



AI speed. Human accountability.

CREDSHIELDS SECURITY ASSESSMENT

Smart Contract Audit

ZPZcoin

Prepared for the EZPZ Team

This report presents the findings of a smart contract audit conducted against ZPZ protocol smart contracts (ZPZCoin, EZPZStakingV1, BuybackRevenueSplitter, ZPZCoinDAONew). Findings were identified through manual review of the contract source by the CredShields security team.

● Critical	1	● High	5	● Medium	13	● Low	11	● Info	8
------------	---	--------	---	----------	----	-------	----	--------	---

CLIENT ZPZcoin	SCOPE ZPZ protocol smart contracts (ZPZCoin, EZPZStakingV1, BuybackRevenueSplitter, ZPZCoinDAONew)	LEAD Shashank (shashank@credshields.com)
REFERENCE EZPZ-2026-SC-01		

Table of Contents

01 Executive Summary

02 Vulnerability Summary

03 Detailed Findings

C001 Stake-and-unstake round trip mints freed sell credit

● CRITICAL

H001 Protected sellers dump the full balance through unregistered pairs

● HIGH

H002 One-token transfer enrolls any address as a protected seller

● HIGH

H003 Unstake and restake each year restarts the 17 percent band

● HIGH

H004 The burn floor halts every burn and every buyback at launch

● HIGH

H005 Vesting marks beneficiaries protected before any basis exists, freezing them entirely

● HIGH

M001 Anti-bot limits throttle every staking payout and every pair buy

● MEDIUM

M002 A forfeited maturity bonus strands treasury capital in the shared pool

● MEDIUM

M003 Permissionless checkpoint rolls let a caller choose the TWAP window on both oracles

● MEDIUM

M004 Buybacks are sandwichable in the default configuration

● MEDIUM

M005 The buyback halt guard checks the wrong quantity, and the reserve locks at the floor

● MEDIUM

M006 A whitelisted recipient disables every sender-side transfer limit

● MEDIUM

M007 The launch-window cap counts protocol transfers as market sales

● MEDIUM

M008 One deployer key holds every privileged role across all three contracts

● MEDIUM

M009 The token contract itself can be enrolled as a protected seller

● MEDIUM

M010 Sell ceilings recompute from a live basis while sold counters stay cumulative

● MEDIUM

M011 The vesting origin is caller-supplied and unbounded, so a schedule can be fully vested at creation

● MEDIUM

M012	A whitelisted stake and exit cycle moves the reward treasury out	● MEDIUM
M013	Rebinding the controller blocks every exit that carries credit	● MEDIUM
L001	A whitelisted stake drains the shared reward pool	● LOW
L002	The whitelisted staking interval inverts and the program is unenterable	● LOW
L003	An all-or-nothing maturity claim above the per-transfer ceiling locks the release permanently	● LOW
L004	Resetting the DEX launch time re-indexes every wallet's daily sell meter	● LOW
L005	The TWAP output floor ignores the pool's own price impact	● LOW
L006	Protection hooks fail open and pay out without sell caps	● LOW
L007	A whitelisted reward claim enrolls the claimer and freezes the wallet	● LOW
L008	Snapshot voting power outlives the stake entirely	● LOW
L009	The APY reserve rate ignores rewards claimed and staked again inside a year	● LOW
L010	A protected wallet that stakes all its credit cannot move any token	● LOW
L011	The maturity release schedule is anchored in the past	● LOW
I001	A third buyback inside ten minutes reverts on an immature checkpoint	● INFO
I002	Skipped reward payments are not signalled, and APY accrues with no funding check	● INFO
I003	Burns bypass the blacklist and the trading switch	● INFO
I004	Proposal finalisation depends on the proposer, and cancellation carries no time bound	● INFO
I005	The participation quorum counts opposing votes, so one wallet clears it alone	● INFO
I006	The early-unstake penalty cited as the flash-stake mitigation does not exist	● INFO
I007	The daily outflow basis is re-snapshotted every day, contradicting its documentation	● INFO
I008	previewUnstake contradicts every exit path it is meant to preview	● INFO

04 Assessment Notes

05 Reviewers & Signatories

The picture at a glance

This report summarizes the security audit findings for the ZPZ protocol smart contracts, encompassing ZPZCoin, EZPZStakingV1, BuybackRevenueSplitter, and ZPZCoinDAONew. The engagement identified a total of 38 findings, categorized by severity as follows: 1 Critical, 5 High, 13 Medium, 11 Low, and 8 Informational.

The most severe finding, **[C001] Stake-and-unstake round trip mints freed sell credit**, represents a critical vulnerability that could lead to an unintended inflation of sell credit. High-severity issues, which warrant immediate attention, include **[H001] Protected sellers dump the full balance through unregistered pairs**, **[H002] One-token transfer enrolls any address as a protected seller**, **[H003] Unstake and restake each year restarts the 17 percent band**, **[H004] The burn floor halts every burn and every buyback at launch**, and **[H005] Vesting marks beneficiaries protected before any basis exists, freezing them entirely**. These critical and high findings highlight significant risks, ranging from potential token manipulation and asset freezes to operational halts, and should be prioritized for remediation.

Following the initial assessment, the client addressed the identified issues. Out of the 38 total findings, all 38 have been fixed. This indicates a thorough effort in resolving the reported vulnerabilities, leading to a significantly improved security posture for the ZPZ protocol.

ASSESSMENT OVERVIEW

This engagement involved a comprehensive security audit of the ZPZ protocol smart contracts, specifically examining ZPZCoin, EZPZStakingV1, BuybackRevenueSplitter, and ZPZCoinDAONew. Our review process combined manual code inspection, analysis of the protocol's design and intent, and static analysis tools. The primary goal was to identify potential vulnerabilities, logical flaws, and deviations from best practices that could impact the security, integrity, or functionality of the ZPZ protocol.

In total, the audit identified 38 findings, categorized across various severity levels. We are pleased to report that the EZPZ team has successfully fixed 38 of these issues, which have been retested and verified.

The most severe issue, classified as **Critical**, was **[C001] Stake-and-unstake round trip mints freed sell credit**. This critical vulnerability represented a fundamental flaw where a specific sequence of staking and unstaking operations could lead to the unintended minting of sell credit, potentially causing an uncontrolled increase in sellable tokens and devaluing the underlying asset.

The audit also uncovered five **High** severity findings, several of which stem from inadequate handling of edge cases or insufficient authorization. **[H003] Unstake and restake each year restarts the 17 percent band**, and **[H005] Vesting marks beneficiaries protected before any basis exists, freezing them entirely**, both relate to complex edge case failures that could lead to unintended user behavior or asset immobility. Authorization weaknesses were highlighted

by [H001] Protected sellers dump the full balance through unregistered pairs, which could allow protected sellers to bypass limitations, and [H002] One-token transfer enrolls any address as a protected seller, indicating a low-cost method to gain unintended protected status. Lastly, [H004] The burn floor halts every burn and every buyback at launch, revealed a critical initialization issue preventing core protocol functions from operating as intended immediately after deployment.

Thirteen **Medium** severity findings were identified, spanning several functional areas. A notable theme concerns issues related to rate limiting and authorization logic, as seen in [M006] A whitelisted recipient disables every sender-side transfer limit and [M012] A whitelisted stake and exit cycle moves the reward treasury out, which could lead to unintended fund movements or bypasses of intended restrictions. Other medium findings highlight potential oracle manipulation or incorrect TWAP usage, such as [M003] Permissionless checkpoint rolls let a caller choose the TWAP window on both oracles and [M004] Buybacks are sandwichable in the default configuration. Several medium findings, including [M005] The buyback halt guard checks the wrong quantity, and the reserve locks at the floor, and [M013] Rebinding the controller blocks every exit that carries credit, indicate further issues with edge case handling or potential fund lockups.

Finally, the engagement identified eleven **Low** severity findings and eight **Informational** findings. These issues generally point to minor inefficiencies, potential user experience issues, or areas for code optimization and improved documentation. Examples include [L003] An all-or-nothing maturity claim above the per-transfer ceiling locks the release permanently, which could lead to asset inaccessibility under specific conditions, and [I006] The early-unstake penalty cited as the flash-stake mitigation does not exist, highlighting discrepancies between documentation and implementation.

AUDITED SCOPE

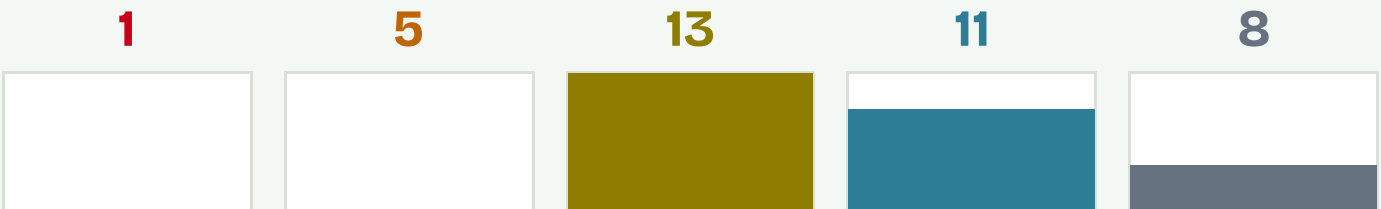
<https://github.com/EZPZTRADING/final-audit/tree/13ad66268cd64e8aeb04df7cb6271f91ff73199d>

RETESTED SCOPE

<https://github.com/EZPZTRADING/final-audit/tree/0ec1a927f941e70aad1affb02f6d45abe30f68ad>

Engagement Details		
Client	Assessment Type	Method
ZPZcoin	Smart Contract Security Audit	Manual code review
Engagement	Report Date	Start Date
ZPZ Protocol Smart Contracts	October 5, 2026	September 29, 2026
End Date	Retest Date	Reference
October 5, 2026	October 15, 2026	EZPZ-2026-SC-01
Total Findings	Lead	
38	Shashank (shashank@credshields.com)	

Finding Severity Distribution





CRITICAL



HIGH



MEDIUM



LOW



INFO

Every finding at a glance

Ordered by severity. Follow an ID for the full write-up.

ID	FINDING	SEVERITY	CATEGORY	STATUS
C001	Stake-and-unstake round trip mints freed sell credit	● CRITICAL	Failure to Handle Edge Cases ([SCWE-083] (https://scs.owasp.org/SCWE/SCSVS-COMP/SCWE-083/))	✓ Fixed
H001	Protected sellers dump the full balance through unregistered pairs	● HIGH	Lack of Rate Limiting ([SCWE-077] (https://scs.owasp.org/SCWE/SCSVS-DEFI/SCWE-077/))	✓ Fixed
H002	One-token transfer enrolls any address as a protected seller	● HIGH	Insufficient Authorization Checks ([SCWE-016] (https://scs.owasp.org/SCWE/SCSVS-AUTH/SCWE-016/))	✓ Fixed
H003	Unstake and restake each year restarts the 17 percent band	● HIGH	Failure to Handle Edge Cases ([SCWE-083] (https://scs.owasp.org/SCWE/SCSVS-COMP/SCWE-083/))	✓ Fixed
H004	The burn floor halts every burn and every buyback at launch	● HIGH	Ether Locked Due to Missing Withdrawal Path ([SCWE-140] (https://scs.owasp.org/SCWE/SCSVS-GOV/SCWE-140/))	✓ Fixed
H005	Vesting marks beneficiaries protected before any basis exists, freezing them entirely	● HIGH	Failure to Handle Edge Cases ([SCWE-083] (https://scs.owasp.org/SCWE/SCSVS-COMP/SCWE-083/))	✓ Fixed
M001	Anti-bot limits throttle every staking payout and every pair buy	● MEDIUM	Lack of Rate Limiting ([SCWE-077] (https://scs.owasp.org/SCWE/SCSVS-DEFI/SCWE-077/))	✓ Fixed

ID	FINDING	SEVERITY	CATEGORY	STATUS
M002	A forfeited maturity bonus strands treasury capital in the shared pool	● MEDIUM	Failure to Handle Edge Cases ([SCWE-083] (https://scs.owasp.org/SCWE/SCSVS-COMP/SCWE-083/))	✔ Fixed
M003	Permissionless checkpoint rolls let a caller choose the TWAP window on both oracles	● MEDIUM	Insufficient TWAP Window or Single Observation ([SCWE-113] (https://scs.owasp.org/SCWE/SCSVS-ORACLE/SCWE-113/))	✔ Fixed
M004	Buybacks are sandwichable in the default configuration	● MEDIUM	Insufficient TWAP Window or Single Observation ([SCWE-113] (https://scs.owasp.org/SCWE/SCSVS-ORACLE/SCWE-113/))	✔ Fixed
M005	The buyback halt guard checks the wrong quantity, and the reserve locks at the floor	● MEDIUM	Failure to Handle Edge Cases ([SCWE-083] (https://scs.owasp.org/SCWE/SCSVS-COMP/SCWE-083/))	✔ Fixed
M006	A whitelisted recipient disables every sender-side transfer limit	● MEDIUM	Lack of Rate Limiting ([SCWE-077] (https://scs.owasp.org/SCWE/SCSVS-DEFI/SCWE-077/))	✔ Fixed
M007	The launch-window cap counts protocol transfers as market sales	● MEDIUM	Lack of Rate Limiting ([SCWE-077] (https://scs.owasp.org/SCWE/SCSVS-DEFI/SCWE-077/))	✔ Fixed
M008	One deployer key holds every privileged role across all three contracts	● MEDIUM	Single Point of Failure in Administrative Key Management ([SCWE-155] (https://scs.owasp.org/SCWE/SCSVS-AUTH/SCWE-155/))	✔ Fixed
M009	The token contract itself can be enrolled as a protected seller	● MEDIUM	Failure to Handle Edge Cases ([SCWE-083] (https://scs.owasp.org/SCWE/SCSVS-COMP/SCWE-083/))	✔ Fixed

ID	FINDING	SEVERITY	CATEGORY	STATUS
M010	Sell ceilings recompute from a live basis while sold counters stay cumulative	● MEDIUM	Lack of Rate Limiting ([SCWE-077] (https://scs.owasp.org/SCWE/SCSVS-DEFI/SCWE-077/))	✔ Fixed
M011	The vesting origin is caller-supplied and unbounded, so a schedule can be fully vested at creation	● MEDIUM	Unvalidated Constructor Parameters ([SCWE-145] (https://scs.owasp.org/SCWE/SCSVS-ARCH/SCWE-145/))	✔ Fixed
M012	A whitelisted stake and exit cycle moves the reward treasury out	● MEDIUM	Lack of Rate Limiting ([SCWE-077] (https://scs.owasp.org/SCWE/SCSVS-DEFI/SCWE-077/))	✔ Fixed
M013	Rebinding the controller blocks every exit that carries credit	● MEDIUM	Ether Locked Due to Missing Withdrawal Path ([SCWE-140] (https://scs.owasp.org/SCWE/SCSVS-GOV/SCWE-140/))	✔ Fixed
L001	A whitelisted stake drains the shared reward pool	● LOW	Failure to Handle Edge Cases ([SCWE-083] (https://scs.owasp.org/SCWE/SCSVS-COMP/SCWE-083/))	✔ Fixed
L002	The whitelisted staking interval inverts and the program is unenterable	● LOW	Failure to Handle Edge Cases ([SCWE-083] (https://scs.owasp.org/SCWE/SCSVS-COMP/SCWE-083/))	✔ Fixed
L003	An all-or-nothing maturity claim above the per-transfer ceiling locks the release permanently	● LOW	Ether Locked Due to Missing Withdrawal Path ([SCWE-140] (https://scs.owasp.org/SCWE/SCSVS-GOV/SCWE-140/))	✔ Fixed
L004	Resetting the DEX launch time re-indexes every wallet's daily sell meter	● LOW	Lack of Rate Limiting ([SCWE-077] (https://scs.owasp.org/SCWE/SCSVS-DEFI/SCWE-077/))	✔ Fixed

ID	FINDING	SEVERITY	CATEGORY	STATUS
L005	The TWAP output floor ignores the pool's own price impact	● LOW	Insufficient TWAP Window or Single Observation ([SCWE-113] (https://scs.owasp.org/SCWE/SCSVS-ORACLE/SCWE-113/))	✔ Fixed
L006	Protection hooks fail open and pay out without sell caps	● LOW	Swallowed Revert Reasons ([SCWE-121] (https://scs.owasp.org/SCWE/SCSVS-CODE/SCWE-121/))	✔ Fixed
L007	A whitelisted reward claim enrolls the claimer and freezes the wallet	● LOW	Insufficient Authorization Checks ([SCWE-016] (https://scs.owasp.org/SCWE/SCSVS-AUTH/SCWE-016/))	✔ Fixed
L008	Snapshot voting power outlives the stake entirely	● LOW	Failure to Handle Edge Cases ([SCWE-083] (https://scs.owasp.org/SCWE/SCSVS-COMP/SCWE-083/))	✔ Fixed
L009	The APY reserve rate ignores rewards claimed and staked again inside a year	● LOW	Failure to Handle Edge Cases ([SCWE-083] (https://scs.owasp.org/SCWE/SCSVS-COMP/SCWE-083/))	✔ Fixed
L010	A protected wallet that stakes all its credit cannot move any token	● LOW	Lack of Rate Limiting ([SCWE-077] (https://scs.owasp.org/SCWE/SCSVS-DEFI/SCWE-077/))	✔ Fixed
L011	The maturity release schedule is anchored in the past	● LOW	Failure to Handle Edge Cases ([SCWE-083] (https://scs.owasp.org/SCWE/SCSVS-COMP/SCWE-083/))	✔ Fixed
I001	A third buyback inside ten minutes reverts on an immature checkpoint	● INFO	Insufficient TWAP Window or Single Observation ([SCWE-113] (https://scs.owasp.org/SCWE/SCSVS-ORACLE/SCWE-113/))	✔ Fixed

ID	FINDING	SEVERITY	CATEGORY	STATUS
I002	Skipped reward payments are not signalled, and APY accrues with no funding check	● INFO	Failure to Handle Edge Cases ([SCWE-083] (https://scs.owasp.org/SCWE/SCSVS-COMP/SCWE-083/))	✔ Fixed
I003	Burns bypass the blacklist and the trading switch	● INFO	Insufficient Authorization Checks ([SCWE-016] (https://scs.owasp.org/SCWE/SCSVS-AUTH/SCWE-016/))	✔ Fixed
I004	Proposal finalisation depends on the proposer, and cancellation carries no time bound	● INFO	Failure to Handle Edge Cases ([SCWE-083] (https://scs.owasp.org/SCWE/SCSVS-COMP/SCWE-083/))	✔ Fixed
I005	The participation quorum counts opposing votes, so one wallet clears it alone	● INFO	Failure to Handle Edge Cases ([SCWE-083] (https://scs.owasp.org/SCWE/SCSVS-COMP/SCWE-083/))	✔ Fixed
I006	The early-unstake penalty cited as the flash-stake mitigation does not exist	● INFO	Poor Governance Documentation ([SCWE-015] (https://scs.owasp.org/SCWE/SCSVS-GOV/SCWE-015/))	✔ Fixed
I007	The daily outflow basis is re-snapshotted every day, contradicting its documentation	● INFO	Lack of Rate Limiting ([SCWE-077] (https://scs.owasp.org/SCWE/SCSVS-DEFI/SCWE-077/))	✔ Fixed
I008	previewUnstake contradicts every exit path it is meant to preview	● INFO	Failure to Handle Edge Cases ([SCWE-083] (https://scs.owasp.org/SCWE/SCSVS-COMP/SCWE-083/))	✔ Fixed

38 findings across the assessment

Findings are ordered by severity.

C001 • FAILURE TO HANDLE EDGE CASES ([SCWE-083]
(<https://scs.owasp.org/SCWE/SCSVS-COMP/SCWE-083/>))

● CRITICAL

✔ Fixed

Stake-and-unstake round trip mints freed sell credit

DESCRIPTION

`ZPZCoin` caps protected sellers at 1% per day and 10% per month of a tracked `freedAmount` credit, representing tokens the protocol genuinely released. The staking contract treats this credit asymmetrically on the two sides of a stake. On entry, `_consumeMarketFreedAmount` consumes only `min(amount, freed)` and skips entirely when credit is zero. On exit, every hook records the FULL principal as new freed credit: `withdrawUnbondedPrincipal`, `_earlyExitWhitelisted`, `_earlyWithdrawWhitelistedPrincipal` and `_claimMaturityRelease`.

A protected wallet holding $1e18$ of credit stakes $10,000,000e18$ of AMM-bought tokens it is not permitted to sell, then withdraws: entry consumes $1e18$, exit credits the full $10,000,000e18$, and the daily sell allowance rises from $0.01e18$ to $100,000e18$. That is a $10,000,000x$ inflation covering the entire staked balance in a single cycle. The whitelisted program removes even the 14-day wait: `stakeWhitelisted` followed by `unstake` in the same block credits the full principal instantly, after which the laundered balance can leave the market in two or three transactions.

AFFECTED CODE

- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/NewSTaking.sol#L1500>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/NewSTaking.sol#L882>

IMPACT

The insider dump-protection the token exists to enforce is defeated. Any protected wallet converts its entire market-bought balance into sell-cap basis, taking 14 days on the regular program or a single block on the whitelisted program, so the daily and monthly caps grow to cover the attacker's whole balance and the inflated credit turns into an unpaced market exit that harms every other holder.

REMEDIATION

Store, per stake at creation time, the exact freed credit that stake consumed on entry, and record only that stored amount, never the full principal, on every exit hook (unbonded withdrawal, early exit, early withdrawal and maturity release). Remove the early return that skips consumption when pre-

stake credit is zero, and scale whitelisted credit by completed lock time so an immediate early exit records nothing, keeping consumed and re-credited amounts conserved on all four paths.

RETEST

Fixed by adding the per-stake ledger `stakeFreedCredit[stakeId]`, which records `min(staked, existing credit)` at creation so an exit restores only that amount. Verified fixed in `0912cf6`.

Protected sellers dump the full balance through unregistered pairs

DESCRIPTION

The protected daily (1%) and monthly (10%) sell caps run only when the recipient is in the owner-maintained `automatedMarketMakerPairs` registry, and pairs are created permissionlessly, so the registry can never be complete. The fallback credit ceiling in `_propagateMarketProtectionOnWalletTransfer` is gated on `!recipientProtected`, and the propagation sets that flag on any recipient, a pair contract included, that receives at least `MIN_PROTECTED_WALLET_TRANSFER` (1e18), so the seller can set the flag that disables the ceiling.

A protected seller holding 10,000,000e18 with 1,000,000e18 of freed credit is paced at 10,000e18 per day, and a direct dump of the balance reverts `ProtectedTransferExceedsFreedCredit`. Transferring 1e18 to any unregistered venue marks that venue protected; the remaining 9,999,999e18 then passes to the same venue with `recipientProtected` true, so the ceiling is skipped and no protected cap applies, leaving only a swap to realise the proceeds. The two transfers are separated by the 30-second cooldown, and the venue is left marked protected, so buys from it revert once its parked credit drains.

The second transfer reverts `DailySellLimitExceeded` while the launch-window cap is active, since that cap is not registry-gated and limits every transfer to 1% of balance per day for the 60 days after `dexLaunchTime` is set; it passes once the window closes, and immediately where `dexLaunchTime` is never set, which is the default. Propagation also reverts while the seller has sold on a registered pair in the current month. The reachable venues are every liquid pair the owner has not registered: other quote assets, V3 pools, other V2 forks, and the canonical pair before registration.

AFFECTED CODE

- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/ZPZCoin.sol#L1400>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/ZPZCoin.sol#L1557>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/ZPZCoin.sol#L1580>

IMPACT

A protected seller converts its entire allocation to market value in two transactions instead of a paced ten-month schedule, causing the price collapse the caps exist to prevent. Inside the launch window the bypass is throttled, not closed: the seller's rate becomes 1% of balance rather than 1% of freed credit. The pair used is left permanently unable to serve buys.

REMEDIATION

Enforce the protected sell limits on every outgoing transfer from a protected seller regardless of the recipient registry, and treat any contract recipient holding both pool tokens as a market venue.

Remove the recipient-protected exemption from the credit ceiling for contract recipients so a pre-marked pair cannot be used as an uncapped exit.

RETEST

Fixed by removing the AMM-registry gate from the protected caps, so they apply to every outbound transfer from a protected seller. Verified fixed in `a3c212b` .

One-token transfer enrolls any address as a protected seller

DESCRIPTION

`_propagateMarketProtectionOnWalletTransfer` marks any recipient of credit-backed tokens as a protected seller with the transferred credit as its cap basis, so a 1e18 transfer enrolls an arbitrary address. The only exemption covers the controller as both caller and recipient, which is a staking deposit; a payout out of staking has `msg.sender == controller` but `to` set to the user, so it matches nothing. The sender needs at least 1e18 of freed credit: less trips `MIN_PROTECTED_WALLET_TRANSFER`, more than its own credit trips `value > senderFreedAmount`. Credit is granted only by the owner or the controller but propagates on every such transfer, letting one holder seed any number of sending addresses, and the transfer reverts while the sender has sold on a registered pair in the current month.

Once the staking contract is enrolled, every payout exceeding its remaining credit reverts `ProtectedTransferExceedsFreedCredit`, so stakers cannot call `withdrawUnbondedPrincipal` or `claimReward`, and the owner's `recoverTokens` and `withdrawRewardPool` fail the same way through `_safeTransferOutExact`. The same seed against `BuybackRevenueSplitter` blocks `rescueToken` on ZPZ. Against a user wallet the victim's balance becomes immovable beyond the credit received. Whitelisting the target does not help: propagation carries no whitelist exemption, and the cap function's whitelist early return requires `!isProtectedSeller`, so marking a whitelisted pair re-attaches the launch-window cap to it.

The same end state also arises without an attacker. The deposit exemption at `ZPZCoin.sol:1537` matches only when the controller is both the caller and the recipient, so whenever `marketProtectionController` does not point at the staking contract, an ordinary deposit fails to match it and enrolls the staking contract itself after which the payout failures above occur from a configuration error alone.

Recovery takes two owner calls, since clearing the flag leaves the residual-credit arm active and the credit must also be zeroed through `recordStakeConsumption`; the sending address can then be blacklisted, but propagation lets the attacker repeat from a fresh one. Vesting releases, maturity releases and already-protected recipients are unaffected, each marking the recipient before transferring; whitelisted early exits break only when an APY reward is payable, since `_payRewardIfAvailable` transfers before the marking.

AFFECTED CODE

- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/ZPZCoin.sol#L1580>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/ZPZCoin.sol#L1537>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/ZPZCoin.sol#L1557>

IMPACT

One transaction from any address holding freed credit blocks principal withdrawals and reward claims for every staker outside the perimeter, and kills all buys on any AMM pair enrolled before registration;

registering it afterwards does not cure the condition, because propagation is skipped only when the recipient is a registered pair, never the sender. Against a user wallet it strands the entire balance behind a one-token cap basis. Recovery is two owner transactions per victim plus a blacklist of the sending address, and propagation lets the attacker repeat from a fresh one, so remediation is attritional rather than final.

REMEDICATION

Extend the controller exemption to any transfer where `msg.sender` is the market-protection controller, regardless of recipient; that single change unblocks every staking payout. Gate entry into the protected perimeter on an explicit owner allowlist rather than on receipt of a credit-backed transfer, and avoid `extcodesize`-style contract detection, which returns zero during construction and is unreliable for delegated EOAs. Clear `freedAmount` alongside the flag in `setProtectedMarketSeller` so a single call is sufficient to release an address.

RETEST

Fixed by removing recipient enrollment entirely and making system addresses reject protected inbound transfers. Verified fixed in `a8ffeae`.

Unstake and restake each year restarts the 17 percent band

DESCRIPTION

The staking APY pays 17%, 9% and 4% over three consecutive yearly bands keyed to each stake's own start time: `_createStake` stamps `newStake.startTime = currentTime`, and reward settlement computes elapsed time from that stamp. Withdrawing and re-staking the same principal restarts the clock in the 17% band, and the only friction is the 14-day unbonding cooldown.

For the same principal over the same three-year wall clock, holding continuously earns $0.17P + 0.09P + 0.04P = 0.300P$. Churning at each year boundary (unstake, wait 14 days, restake) earns two full 17% cycles plus a pro-rated third, roughly $0.497P$: a 65% increase for about nine transactions and 28 idle days. Every churning staker draws about 1.65 times the designed reward from the shared `_rewardPoolBalance`, and once the pool empties every remaining staker's `claimReward` reverts with an insufficient reward pool error.

AFFECTED CODE

- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/NewSTaking.sol#L706>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/NewSTaking.sol#L1034>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/NewSTaking.sol#L1058>

IMPACT

The reward pool drains roughly 1.65 times faster than designed, so long-term stakers lose their accrued rewards to revert-doomed claims once churners exhaust the pool, and the protocol's cost of capital is materially higher than its tokenomics model assumes.

REMEDIATION

Select the APY band from a per-wallet cumulative staking tenure that unstaking and restaking do not reset, so the 17% band applies once per wallet per lifetime principal, or apply an APY penalty on exit before the final band completes.

RETEST

Fixed by moving the APY bands from per-stake anniversaries to one global calendar schedule. Verified fixed in `a8ffeae`.

The burn floor halts every burn and every buyback at launch

DESCRIPTION

`_update` rejects any burn that would take total supply below `BURN_FLOOR`, which is set to 5,000,000,000e18. `INITIAL_SUPPLY` is 1,000,000,000e18, so from deployment the floor sits four billion tokens above actual supply and every burn reverts `BurnFloorBreached`, including a one-wei burn. This disables `ERC20Burnable.burn`, `burnFrom`, and `burnForPlatform`.

The same constant disables the buyback. `BuybackRevenueSplitter.executeBuybackAndBurn` ends with `IZPZBurnable(zpzToken).burn(zpzBought)`, so the splitter's entire purpose is unreachable until supply grows fivefold. Because the maximum supply reachable through the documented emission schedule (initial supply plus the development and treasury allocations) stays below the floor, the condition is not merely a launch-window problem.

AFFECTED CODE

- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/ZPZCoin.sol#L506>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/ZPZCoin.sol#L56>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/BuybackRevenueSplitter.sol#L374>

IMPACT

The deflationary mechanism the protocol advertises cannot run. Platform burns, user burns and buyback-and-burn all revert from deployment, so collected buyback revenue accumulates with no way to convert it into supply reduction.

REMEDIATION

Set `BURN_FLOOR` below the reachable supply under the emission schedule, or make the floor take effect only once supply has exceeded it. Fix this before or together with the defects it currently masks, all of which are unreachable while every buyback reverts and activate the moment the floor is corrected: the missing blacklist check on the burn path, the permissionless checkpoint roll that lets a caller choose the averaging window, the sandwichable output floor in the default configuration, the halt guard that compares the wrong quantity against the burn floor, and the unspaced checkpoint roll that strands a third buyback inside ten minutes.

RETEST

Fixed by restoring `INITIAL_SUPPLY` to 1,000,000,000e18 and making the floor conditional, so it binds only once live supply reaches it. Measured at 0ec1a92: initialization leaves 9,000,000,000 of lifetime capacity,

every documented issuance phase mints, both mint-funded vesting branches create their full allocations, a burn succeeds at the 1,000,000,000 launch supply, and every positive burn reverts at exactly the floor. This is the second remedy the finding offered. Sections 5, 7.3 and 7.4 of the whitepaper still describe burns as disabled below the floor and need amending. Verified fixed in 0ec1a92.

Vesting marks beneficiaries protected before any basis exists, freezing them entirely

DESCRIPTION

`protectedMarketSellers` and `freedAmount` are set and cleared by different code. The propagation sets the flag and moves the basis; `_recordFreedAmount` also sets the flag; but no path clears the flag when the basis reaches zero, and the only writers of false are the owner-only setters. The arithmetic is what makes a stale flag total rather than partial. A protected sender whose recipient is not already protected faces two bounds at once: value must not exceed `freedAmount` [from] and value must be at least `MIN_PROTECTED_WALLET_TRANSFER` of `1e18` (`ZPZCoin.sol`, constant), and for any basis in `[0, 1e18)` no value satisfies both. The only escapes are a recipient that is already protected and a registered AMM pair, and `automatedMarketMakerPairs` is empty in the shipped state. Three first-party paths put wallets into exactly that band before any credit arrives: `createDevelopmentVesting` and `createTreasuryVesting` both mark the beneficiary when the schedule is written, and credit only arrives on the first release. `_createMaturityRelease` marks the payee before the claim that would fund it. The Sepolia deployment script marks the development wallet this way.

AFFECTED CODE

- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/ZPZCoin.sol#L1557>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/ZPZCoin.sol#L1577>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/ZPZCoin.sol#L915>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/ZPZCoin.sol#L971>

IMPACT

The victim is left with a ZPZ balance it cannot send anywhere, and what it loses the use of is its own pre-existing ZPZ, not just tokens the protocol issued: `ZPZCoin.sol:1558` reverts every transfer to an unprotected address, `ZPZCoin.sol:1468` computes a zero daily cap so no AMM sale passes, and `ZPZCoin.sol:506` rejects a burn in every reachable supply state, leaving only an already-protected recipient as a legal destination. The state does not expire and does not self-heal, because the only writers of false are owner-only, so recovery is one token-owner transaction per wallet and the owner is the only party who can perform it. The protocol inflicts this on itself: its own development and treasury wallets are frozen against unprotected destinations for the entire 180-day cliff, and every matured whitelisted staker is frozen from the moment `_createMaturityRelease` marks it until its first release credits a basis. Where an attacker is involved it has spent `1e18` of its own freed credit, permanently, and holding nothing. Where no attacker is involved, on the vesting and maturity paths, nobody gains anything at all and the loss of access is purely self-inflicted.

REMEDIATION

Clear the flag when the basis reaches zero: in `_propagateMarketProtectionOnWalletTransfer` and in `_consumeFreedAmount`, set `protectedMarketSellers [account] = false` when the updated `freedAmount` is zero and the account holds no other claim. Stop marking a wallet before it has a basis. Move the marking to the first release, and move `NewSTaking.sol` to the first claim. Reconcile the two bounds so they cannot be jointly unsatisfiable: allow a transfer of exactly `freedAmount [from]` when that is below `MIN_PROTECTED_WALLET_TRANSFER`, rather than requiring both a floor and a ceiling that can cross.

RETEST

Fixed by enrolling the beneficiary on an actual release rather than on schedule creation. Verified fixed in `390d071`.

Anti-bot limits throttle every staking payout and every pair buy

DESCRIPTION

`ZPZCoin._update` keys its cooldown and transaction counter on the transfer sender: `cooldownPeriod = 30`, `maxTransactionsPerPeriod = 3` and `transactionPeriod = 300` are the initialization defaults. Every staking payout is a transfer whose `from` is the staking contract, so all users of both programs share a single budget of three payouts per 300 seconds, each at least 30 seconds apart. Every AMM buy is a transfer whose `from` is the pair, so all traders on that pair share three buys per 300 seconds chain-wide. `maxTransactionAmount` additionally reverts any single payout above 50,000,000e18, and `maxWalletAmount` stops new stakes once the staking contract's balance reaches 500,000,000e18.

An attacker holding a dust stake can pin the shared payout budget by claiming a few wei of reward three times per window, reverting every other user's payout; and because a blocked payout rolls back the whole calling transaction, each block also fails the entire exit. The throttle applies only while neither the staking contract nor the recipient is token-whitelisted, since the anti-bot block nests under `!whitelisted[from]` then `!whitelisted[to]`.

The wallet cap also applies to market pairs: once an unwhitelisted pair holds 500,000,000e18, every further sell into it reverts `MaxWalletExceeded`, and the `automatedMarketMakerPairs` registry is never consulted in the anti-bot block. Only `script/deploy.sepolia.js` whitelists the staking contract; `deploy.new.js`, `deploy.EZPZStakingDAO.js` and `deploy.DAO.js` contain no whitelist calls, so the shipped default configuration launches with the throttle active.

AFFECTED CODE

- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/ZPZCoin.sol#L483>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/ZPZCoin.sol#L495>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/ZPZCoin.sol#L469>

IMPACT

At most three reward, principal or maturity payouts complete per five minutes across the entire staking user base, buys on each pair are throttled to three per five minutes for every trader, payouts above 50,000,000e18 can never execute, and total staking deposits are capped at 500,000,000e18. Any single user can weaponise the shared budget into targeted griefing of every other user's withdrawal. The failure is loud (the fourth buy or payout at launch reverts immediately) and a single owner `setWhitelist` call permanently disables the throttle for an infrastructure address, so the issue is bounded to an availability window rather than a persistent fault.

REMIEDIATION

Exempt transfers sent by the market-protection controller and by registered market-maker pairs from the sender-side anti-bot checks, or whitelist these infrastructure addresses in a code-enforced way at deployment rather than as a revocable post-deploy setting. Consider keying per-user limits on the human counterparty, the recipient for payouts and the router caller for buys, rather than on the contract that moves the tokens. Note that whitelisting a pair also exempts every sell into it from the per-transaction cap, so the whitelist fix must be paired with protected-cap enforcement on the seller.

RETEST

Fixed by exempting controller payouts and deposits in `_update`, and skipping the shared cooldown when the sender is a registered pair. Verified fixed in `a8ffae`.

A forfeited maturity bonus strands treasury capital in the shared pool

DESCRIPTION

`_reserveMaturityBonus` funds any shortfall in the 100% whitelisted bonus by pulling tokens from `rewardTreasury`, but `_forfeitMaturityBonus` returns the whole reserved amount to the communal `_rewardPoolBalance` with no record of which portion the treasury supplied. A whitelisted staker can stake 50,000,000e18, early-exit in the next block, recover the principal in full because the early exit carries no penalty, and leave 50,000,000e18 of treasury capital sitting in the shared reward pool.

The owner's recovery routes are both conditional. `recoverTokens` counts `_rewardPoolBalance` inside its reserved sum (`NewSTaking.sol`), so it reports no surplus. `withdrawRewardPool` works only while `_activeStakeCount == 0` (`NewSTaking.sol`), and `_createStake` increments that counter unconditionally with no minimum enforced beyond `_minStakeAmount`, so a single minimum-size stake from any address closes that route permanently 1,000e18 under the repository's deploy scripts, or one wei where `_minStakeAmount` is left unset. The treasury capital then pays APY to all stakers, including the wallet that moved it there, with no path back. The gate is not specific to bonus-derived funds: because `_createStake` increments the counter on every stake (`NewSTaking.sol`), any surplus the owner holds in the reward pool including a simple overfunding is equally unreclaimable for as long as a single stake stays open.

AFFECTED CODE

- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/NewSTaking.sol#L1905>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/NewSTaking.sol#L1232>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/NewSTaking.sol#L1839>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/NewSTaking.sol#L713>

IMPACT

Any whitelisted wallet moves up to one bonus-worth of treasury capital into the communal reward pool at the cost of gas alone, and that capital becomes unrecoverable as soon as any address keeps a stake open, since `recoverTokens` counts the pool as reserved and `withdrawRewardPool` requires a zero active-stake count. The drain is bounded and self-limiting rather than open-ended: the released bonus refills the pool, so a further treasury pull requires the pool to empty again. The actor gains nothing directly, recovering only its own principal; the harm is the permanent loss of owner discretion over the relocated capital, which thereafter pays APY to every staker including the wallet that moved it.

REMIEDIATION

Track per stake the portion of the bonus the treasury funded and return exactly that portion to `rewardTreasury` on forfeit rather than to the reward pool. Replace the `_activeStakeCount == 0` gate on `withdrawRewardPool` with a solvency check against `_accruedRewardLiability`, and enforce a non-zero floor in `setMinStakeAmount` so the gate cannot be held open by a dust stake.

RETEST

Fixed by recording the funding treasury per stake and crediting a forfeited bonus to `treasuryRefunds` for delivery by `withdrawTreasuryRefund`. Verified fixed in `390d071`.

Permissionless checkpoint rolls let a caller choose the TWAP window on both oracles

DESCRIPTION

`updateTwapObservation` is unprivileged and rolls the buyback's TWAP anchor whenever ten minutes have passed since the last roll (`BuybackRevenueSplitter.sol`). The documented defence assumes an attacker must sustain a manipulation across the whole averaging window, but the attacker chooses when that window begins. Moving the pair price and rolling the checkpoint in the same block stores a clean cumulative at the manipulation start, after which holding the manipulated price for the ten-minute minimum makes the window average equal the manipulated spot exactly, because the delta divided by elapsed time contains nothing but the manipulated price.

The effect is directly observable: with the honest checkpoint in place a manipulated pool cannot clear the floor and the buyback reverts `Insufficient ZPZ bought`, whereas after the attacker re-anchors the window the identical trade executes. On a fourfold adverse price move, a 100,000-unit buyback whose floor should sit near 95,000 ZPZ sees that floor collapse to 23,750 roughly three quarters of the buyback's value available to the attacker.

The same defect exists twice, and only one instance is currently blocked. `updateNativeTwapObservation` (`BuybackRevenueSplitter.sol`) is unprivileged on identical terms and anchors the window used by `convertNativeRevenueToPaymentToken`, which carries no burn-floor guard `_requireBuybackNotHalted` is called from `executeBuybackAndBurn` alone (`BuybackRevenueSplitter.sol`). The buyback vector is therefore unreachable while the burn floor sits above total supply, because every call reverts `At burn floor; buyback halted` before any oracle logic runs, whereas the native-conversion vector is live as soon as `nativePaymentOraclePair` is configured.

On both paths the caller controls neither the trigger nor the timing, because `executeBuybackAndBurn` (`BuybackRevenueSplitter.sol`) and `convertNativeRevenueToPaymentToken` (`BuybackRevenueSplitter.sol`) are `onlyOwner`: the manipulated price must be held against arbitrage from the checkpoint roll until the owner happens to execute, at least the ten-minute minimum later.

AFFECTED CODE

- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/BuybackRevenueSplitter.sol#L713>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/BuybackRevenueSplitter.sol#L542>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/BuybackRevenueSplitter.sol#L726>

IMPACT

Owner-executed buybacks can be made to execute at an attacker-held manipulated price after ten minutes of preparation instead of the sustained multi-window manipulation the design assumes, transferring a large fraction of each buyback's value. Exposure is bounded by the per-call slippage ceiling and by the reserve available to each call, and requires the relevant oracle pair to be configured. The buyback vector additionally waits on the burn floor being corrected; the native-conversion vector does not.

REMIEDIATION

Restrict both checkpoint rolls to the owner or a timelocked keeper, and require the stored checkpoint to be older than a configurable multiple of the minimum window before it can be replaced. A stale checkpoint is not a second exposure here: with the honest checkpoint left in place the averaged floor is dominated by real history and a manipulated buyback fails closed, so the roll is what the attack needs.

RETEST

Fixed by making both observation rolls `onlyOwner` . Verified fixed in `a3c212b` .

Buybacks are sandwichable in the default configuration

DESCRIPTION

`BuybackRevenueSplitter` derives its on-chain output floor from `priceOraclePair`, but nothing configures that address at construction it is written only by the owner calling `setPriceOraclePair` (`BuybackRevenueSplitter.sol`). While it is unset, `executeBuybackAndBurn` computes its floor solely through `_liveQuoteFloor` (`BuybackRevenueSplitter.sol`), because the TWAP refinement sits behind an `if (priceOraclePair != address(0))` guard (`BuybackRevenueSplitter.sol`).

`_liveQuoteFloor` calls `router.getAmountsOut` a read of the same pool state, in the same block, that the swap will execute against (`BuybackRevenueSplitter.sol`). The contract's own documentation concedes the consequence: the live quote "is itself read from the same, same-block pool state the swap executes against, so on its own it does NOT protect against an active sandwich attacker" (`BuybackRevenueSplitter.sol`). The floor therefore moves with the manipulation instead of resisting it, and `maxSlippageBp` bounds nothing, because its five percent is taken from a quote already read out of the manipulated pool: against a fourfold adverse move the enforced floor falls from 95,000 to 23,750 rather than holding. The only protection that survives is the caller-supplied `minZpzOut`, retained by `if (minOut > floor) floor = minOut` (`BuybackRevenueSplitter.sol`) which is precisely the input the NatSpec says this floor exists to backstop against "a careless/compromised owner setting `minZpzOut` too low". The backstop is inert in exactly the case it was written for. This is the state every deployment starts in and it persists until the owner both configures the pair and waits out `MIN_TWAP_WINDOW`.

The exposure is latent while the burn floor sits above total supply, because `_enforcedMinOut` is reached at `BuybackRevenueSplitter.sol`, after the halt guard at reverts every call. It is also the only sandwich vector here that is atomic: a searcher front-runs and back-runs the owner's buyback inside one block, with no sustained manipulation required.

AFFECTED CODE

- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/BuybackRevenueSplitter.sol#L462>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/BuybackRevenueSplitter.sol#L480>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/BuybackRevenueSplitter.sol#L638>

IMPACT

Every buyback executed before the owner configures the oracle pair and warns it can be sandwiched atomically, and the loss is bounded by pool depth and attacker capital rather than by `maxSlippageBp`, since the enforced floor is derived from the manipulated quote itself. A buyback submitted with a low or zero

`minZpzOut` has no effective floor at all. Because the vulnerable state is the default and needs no attacker setup beyond observing the mempool, exposure begins at deployment rather than at a misconfiguration.

REMEDIATION

Require `priceOraclePair` at construction, or make `executeBuybackAndBurn` revert while it is unset, matching `convertNativeRevenueToPaymentToken`, which already fails closed on the equivalent check (`BuybackRevenueSplitter.sol:411`). The same contract handles the same risk in opposite directions; aligning the buyback path with the native path removes the gap. Until the oracle is live, route buybacks through a private mempool and always supply a `minZpzOut` computed off-chain.

RETEST

Fixed by requiring a mature TWAP reference per hop and bounding spot against it, and by making `_enforcedMinOut` revert while `priceOraclePair` is unset. Measured at `0ec1a92`: unset oracle reverts Oracle pair not set, immature reverts TWAP observation not ready, mature executes at launch supply, front-run pool reverts Execution price below TWAP. Verified fixed in `0ec1a92`.

The buyback halt guard checks the wrong quantity, and the reserve locks at the floor

DESCRIPTION

`_requireBuybackNotHalted` asserts `zpzToken.totalSupply() > burnFloor` (BuybackRevenueSplitter.sol) and runs before the swap (BuybackRevenueSplitter.sol). The operation it protects is `IZPZBurnable(zpzToken).burn(zpzBought)` (BuybackRevenueSplitter.sol), which the token permits only when `totalSupply() - zpzBought ≥ BURN_FLOOR` (ZPZCoin.sol). Whenever the purchase exceeds the headroom above the floor the guard passes, the reserve is debited, the approval is set, the swap executes against the pool, and the burn then reverts the entire transaction.

The result is that the final approach to the burn floor is a hard denial rather than a graceful partial burn. There is no clamp on `amountIn`, no partial-burn fallback and no path that retains the surplus, so the operator cannot determine off-chain how much is safely burnable and each attempt costs a full swap's gas before reverting.

Reaching the floor strands the reserve permanently. `collectRevenue` is unprivileged and credits ten percent of every payment to `buybackReserve` (BuybackRevenueSplitter.sol) with no burn-floor check, while the only path that spends it is `executeBuybackAndBurn` (BuybackRevenueSplitter.sol) which is exactly what halts once `totalSupply()` is no longer above the floor. `rescueToken` then excludes the reserve from recovery, computing `recoverable = balance - buybackReserve` (BuybackRevenueSplitter.sol), so the balance has no exit at all and keeps growing for as long as anyone calls `collectRevenue`. This mirrors the native-side lock the report records separately, on the payment-token side.

The failure mode is currently masked by the burn floor sitting above total supply, in the same way the missing blacklist check on the burn path is: with supply at 1,000,000,000e18 against a 5,000,000,000e18 floor, `_requireBuybackNotHalted` reverts "At burn floor; buyback halted" before any swap is attempted. It becomes reachable the moment the floor is corrected, so remediate it alongside the burn-floor correction.

AFFECTED CODE

- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/BuybackRevenueSplitter.sol#L442>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/BuybackRevenueSplitter.sol#L374>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/ZPZCoin.sol#L506>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/BuybackRevenueSplitter.sol#L314>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/BuybackRevenueSplitter.sol#L314>

IMPACT

As supply nears the burn floor every buyback reverts after the swap has already executed, so each attempt costs a full swap's gas and the operator has no on-chain signal of how much is actually burnable; that half is recoverable by shrinking `amountIn` until the purchase fits the headroom. Once supply reaches the floor the reserve is stranded outright: the buyback is permanently halted, `collectRevenue` continues to credit it, and `rescueToken` is coded to exclude it, so collected revenue accumulates with no path out and no way to convert it into supply reduction.

REMEDIATION

Compute the burnable headroom as `totalSupply() - BURN_FLOOR` and either clamp `amountIn` so the resulting purchase fits inside it, or burn `min(zpzBought, headroom)` and retain the remainder for the next execution. Separately, allow `rescueToken` to reach `buybackReserve` once the buyback is halted, or stop crediting the reserve while `totalSupply()` is at the floor, so revenue cannot accumulate behind a path that can never run again.

RETEST

Fixed by clamping the burn to the available headroom, `burned = min(zpzBought, totalSupply - BURN_FLOOR)`, so the swap no longer reverts on the burn and the excess is retained. `rescueToken` also gained a halted-recovery branch, so the reserve can leave to the treasury. Verified fixed in `a908e33`.

A whitelisted recipient disables every sender-side transfer limit

DESCRIPTION

The anti-bot block in `_update` is nested: the outer condition tests the sender (`!isVestingRelease && !whitelisted[from]`, ZPZCoin.sol) and the inner one tests the recipient (`!whitelisted[to]`, ZPZCoin.sol). Because both must hold for the block to execute, a whitelisted **recipient** alone suppresses every sender-side control `maxTransactionAmount`, `maxWalletAmount`, the cooldown and the per-period transaction counter are all skipped even when the sender is an ordinary, non-whitelisted holder.

The consequence is that the remediation for the staking-payout and pair-buy throttles creates this bypass. Whitelisting an AMM pair so staking payouts and pair buys are not throttled simultaneously means every seller transferring into that pair escapes the transfer limits entirely. An ordinary holder moved 200,000,000e18 to a whitelisted recipient in four consecutive transfers inside one block four times the 50,000,000e18 per-transfer ceiling in aggregate, with no cooldown observed between them and `transactionCount` and `lastTransactionTime` for that sender were never written at all. A single transfer above the ceiling is equally unchecked on this path.

AFFECTED CODE

- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/ZPZCoin.sol#L466>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/ZPZCoin.sol#L467>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/ZPZCoin.sol#L469>

IMPACT

Any address the owner whitelists becomes an unlimited drain for every other holder: transfers into it ignore the per-transaction ceiling, the wallet ceiling, the cooldown and the frequency cap. Because whitelisting the AMM pair is the natural fix for the payout and buy throttling, applying that fix hands every seller a complete bypass of the anti-bot system.

REMEDIATION

Evaluate the sender-side controls on the sender alone and the recipient-side control (`maxWalletAmount`) on the recipient alone, rather than gating all four behind both parties. Exempt infrastructure addresses from the specific limit they need relieved instead of from the whole block.

RETEST

Fixed by no longer skipping the sender-side controls when only the recipient is whitelisted. Verified fixed in 390d071.

The launch-window cap counts protocol transfers as market sales

DESCRIPTION

During the sixty days after `dexLaunchTime` is set, `_enforceDailySellLimit` caps every wallet's total daily outflow at 1% of its day-opening balance (`DAILY_SELL_LIMIT_BP = 100`, ZPZCoin.sol) and counts every outgoing transfer, including staking deposits, with no controller exemption. A deposit passes only when `amount ≤ balance / 100`, so clearing the 10,000,000e18 whitelisted minimum (`WHITELIST_MIN_STAKE`, NewStaking.sol) in one transaction requires a balance of at least 1,000,000,000e18.

No wallet reachable through ordinary transfers can hold that: `maxWalletAmount` is 500,000,000e18 (ZPZCoin.sol), whose 1% daily allowance is 5,000,000e18 half the minimum stake. The ceiling is not absolute, since the initial mint bypasses the check entirely (`from = address(0)` skips the whole block, so the initial recipient holds 1,000,000,000e18 while un-whitelisted) and a whitelisted sender can push any balance into a non-whitelisted wallet but neither route is available to an ordinary participant. The whitelisted three-year program is therefore mathematically unreachable for any non-token-whitelisted user for the entire launch window. Whitelisting the staking contract does not help, because the throttled party is the depositor, not the recipient, and `dexLaunchTime` cannot be unset once written (`setDexLaunchTime` requires a non-zero value), so the only workaround is whitelisting each depositor individually.

Two preconditions apply. The cap is dormant until the owner calls `setDexLaunchTime`, which no deployment script in the repository does, so exposure begins at launch rather than at deployment. And the whitelisted half is latent while the whitelisted stake bounds remain inverted, because `stakeWhitelisted` reverts on them before the transfer is attempted; the regular-staking throttle is live regardless, so the finding holds on that half alone.

The same cap applies in the opposite direction, to payouts. On the staking contract's first outgoing transfer of the day `_enforceDailySellLimit` sets `dailySellLimitBasis[staking] = balanceOf(staking)` and accumulates every subsequent payout into one `soldToday[staking]` counter (ZPZCoin.sol), so all users of both programs share a single daily ceiling of one percent of the pooled balance. With 200,000,000e18 held by the contract, combined withdrawals, reward claims and tranche claims across every user are capped at 2,000,000e18 per day, after which each further payout reverts `DailySellLimitExceeded` until the next day.

Whitelisting the staking contract clears the payout half, because there the throttled party is the contract itself, but the mitigation is not durable. The whitelist early return requires `!isProtectedSeller` (ZPZCoin.sol:1396), so the ceiling re-engages the moment the contract is marked a protected seller which any holder of freed credit can arrange with a single transfer at the one-token minimum. Inside the launch window that seed transfer is itself subject to the one percent cap, so the sender needs roughly one hundred times the seed in balance, but the cost remains trivial. Whitelisting is therefore only a durable mitigation once entry into the protected perimeter is gated.

The maturity-bonus treasury top-up is caught the same way. `_reserveMaturityBonus` covers any shortfall by pulling from `rewardTreasury` (NewStaking.sol), issuing `transferFrom(rewardTreasury, stakingContract, deficit)`, so the recipient is the controller and the transfer is protocol-internal at both ends. The protected-seller credit ceiling correctly ignores it `msg.sender` and `to` both equal the controller,

so `_propagateMarketProtectionOnWalletTransfer` returns at `ZPZCoin.sol` before reaching the ceiling, and a protected treasury funds the bonus normally. The launch-window cap is a separate path and does apply: it carries no controller exemption, and its whitelist early return requires `!isProtectedSeller` (`ZPZCoin.sol`), so a treasury enrolled automatically by holding a vesting allocation (`ZPZCoin.sol`) or by ever having been credited freed tokens (`ZPZCoin.sol`) cannot escape by being token-whitelisted either. A treasury holding 50,000,000e18 can fund at most 500,000e18 of bonus per day, so any whitelisted stake whose deficit exceeds that reverts `DailySellLimitExceeded` and `stakeWhitelisted` fails for reasons the staker neither controls nor can observe, with a revert reason pointing at the token's sell limits rather than the funding path that tripped them.

IMPACT

Normal users cannot enter the flagship whitelisted staking program during the launch window, and all regular staking is limited to one percent of balance per day, degrading launch economics and concentrating participation among token-whitelisted insiders. On the payout side the same cap throttles aggregate withdrawals and rewards to one percent of staked balance per day across the entire user base, so at realistic participation most users' payouts revert late in each day an availability ceiling that worsens as the pool grows in users rather than in daily percentage.

REMIEDIATION

Exempt transfers where the market-protection controller is the sender or the recipient from the launch-window outflow cap in `_enforceDailySellLimit`, so neither deposits nor payouts are counted as market sales. One exemption covering both directions resolves all three paths, including the maturity-bonus top-up. Surface the treasury's remaining daily allowance through `whitelistedStakeStatus` so the condition is visible before a user spends gas. Note this is a separate change from the anti-bot exemption that resolves the maturity-claim ceiling and the shared sender budget: the two caps live in different functions, and fixing one does not fix the other.

RETEST

Fixed by exempting deposits into the controller and registered system contracts from `_enforceDailySellLimit`, placed after the protected-cap check. Verified fixed in `a3c212b`.

One deployer key holds every privileged role across all three contracts

DESCRIPTION

The audited contracts each install whatever owner their initializer is handed, and the only script that wires the full stack hands all of them the same externally owned account, taken from a single `PRIVATE_KEY` and never transferred. `script/deploy.sepolia.js:40` passes `[deployer.address, deployer.address]` to `ZPZCoin.initialize`, making one EOA both `owner()` and the recipient of the entire `1e27 INITIAL_SUPPLY`. `ZPZCoin.initialize` passes the same address as `EZPZStakingV1`'s `initialOwner`, and `NewSTaking.sol` installs it as owner while `NewSTaking.sol` assigns `rewardTreasury = initialOwner` in the same statement block, so the one role whose only protection is being a different party from the owner is the same party by construction rather than by configuration. `deploy the DAO from the same signer and ZPZCoinDAONew.sol` installs `__Ownable_init(msg.sender)`. then has that key install the staking proxy as `marketProtectionController`. None of the three initializers binds the installed owner to anything, no multisig check, no timelock, no separation between the upgrade authority and the operational owner, and `ZPZCoin` additionally lets the owner name a second upgrade controller that is not checked for being a contract. The consequence is not any one row's ceiling but their union, and it composes across contracts because the staking contract's checkpoint ledger is the DAO's only weight source.

AFFECTED CODE

- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/ZPZCoin.sol#L401>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/NewSTaking.sol#L524>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/NewSTaking.sol#L530>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/ZPZCoinDAONew.sol#L248>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/ZPZCoin.sol#L1718>

IMPACT

If that single key is compromised or its holder turns, the attacker walks away with every balance, every stake and the DAO's entire electorate: three transactions replace the three implementations with no timelock, and because `_stakeCheckpoints` is the DAO's only weight source the same key both rewrites the electorate and controls the contract that reads it, the one composition TRUST-6 itself identifies as the route a signalling DAO cannot survive. Stakers lose their principal, holders lose their balances, and voters lose the signal; there is no guardian, no second signature and no delay anywhere on the path, so no party can intervene between the first transaction and the last. The key holder needs no code defect at all, the powers are the documented ones, exercised together.

REMEDIATION

Separate the authorities before mainnet and make the separation visible. Transfer each contract's ownership to a distinct multisig and the three upgrade authorities to a timelock controller, so no single key can both rewrite `_stakeCheckpoints` and upgrade the DAO that reads it; set

`rewardTreasury` to an address that is not the staking owner rather than inheriting it. Require a non-EOA for the upgrade controller and add a view that reports `upgradeController`, `marketProtectionController` and the registered `platformBurners` so the configuration is auditable on-chain. Note that `ZPZCoin`'s implementation is 25,373 bytes against EIP-170's 24,576 under the project's own compiler settings, so a replacement implementation cannot currently be deployed without a size reduction.

RETEST

Fixed by gating `_authorizeUpgrade` on a configured contract controller in all three proxies, where zero disables upgrades and a controller must carry code. Staking no longer seeds `rewardTreasury` from the owner, `setRewardTreasury` rejects the owner and the pending owner, and both ownership functions reject the treasury. `getAuthorityConfig` exposes the upgrade authority, the market controller and an enumerable burner registry. Verified fixed in `18646ef`.

The token contract itself can be enrolled as a protected seller

DESCRIPTION

`_releaseVestedTokens` calls the overridden `_update` with `from == address(this)`, so every release runs the propagation; `emergencyWithdraw`'s self-token branch does the same through `IERC20(self).safeTransfer`. The only thing keeping the propagation inert on those paths is the early return, which fires while the token contract is neither protected nor carrying credit, and the propagation has no special case for `address(this)` as a recipient: tests the AMM registry, the controller and the sender, and then reads the recipient's flag. So a `1e18` transfer to the token's own address passes `ZPZCoin.sol` at equality and `ZPZCoin.sol` at the floor, marks the contract and parks `1e18` of basis there. Thereafter the contract's own outbound transfers face the fail-closed gate as an ordinary sender. One adjacent claim does not hold and is not made here: a vesting release does not double-credit the beneficiary. The credit transfer is exactly zero-sum, so any excess basis the beneficiary holds equals what the attacker parked and comes out of the attacker's own credit. That is a donation, not an inflation.

AFFECTED CODE

- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/ZPZCoin.sol#L534>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/ZPZCoin.sol#L1549>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/ZPZCoin.sol#L1580>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/ZPZCoin.sol#L1136>

IMPACT

Three of `ZPZCoin`'s own self-originated paths break for the price of `1e18` of the attacker's freed credit, which the attacker surrenders permanently and cannot recover without the controller or the owner. A funded vesting schedule past its cliff becomes unreleasable once the beneficiary is itself unprotected: the beneficiary cannot receive it because `ZPZCoin.sol:1557` reverts, the owner cannot rescue it because `ZPZCoin.sol:1141-1142` excludes the reserved balance from `emergencyWithdraw`, there is no cancel or reassign function anywhere in the contract (L011), and `ZPZCoin.sol:912`'s allocation counter is never decremented so the headroom is consumed too. The owner's own emergency ZPZ rescue is disabled by a permissionless action with nothing in the event stream saying it happened. The state is owner-recoverable in one call to `setProtectedMarketSeller (address(this), false)`, which is what holds the impact at Medium rather than higher. Until the owner notices, vested tokens that are already due are unreachable by their beneficiary and by the owner alike.

REMIEDIATION

Reject the token's own address as a counterparty in the propagation: return early in `_propagateMarketProtectionOnWalletTransfer` when `from == address(this)` or `to == address(this)`, so self-originated releases and rescues never enter the perimeter and the contract can never be marked. Apply the same exclusion to `marketProtectionController`. Separately, move the self-token branch of `emergencyWithdraw` onto `super._update`, as `_lockVestingAllocation` already does, so the owner's rescue path does not depend on the perimeter state at all.

RETEST

Fixed by covering the token's own address in `isSystemContract` and rejecting protected inbound transfers. Verified fixed in `390d071`.

Sell ceilings recompute from a live basis while sold counters stay cumulative

DESCRIPTION

`_enforceProtectedSellLimits` reads the basis live from `freedAmount` [from] and recomputes both ceilings from it on every transfer, the daily figure and the monthly, while `protectedSoldToday` and `protectedSoldThisMonth` are cumulative and their window anchors advance only on a full elapse. Nothing rescales or debits a counter when the figure that produced it changes, and three different writers move the ceiling under a fixed counter: `setProtectedDailySellLimitBp`, the propagation's credit transfer, and the staking exit's credit through `recordFreedAmount` into `_recordFreedAmount`. The authors wrote a latch against exactly this amplification shape, with its comment naming sell-then-split, and that latch is evaluated only inside the propagation, so the bp setter and the hook writer both bypass it. Part of this is documented. The NatSpec states that the day and month sold counters are intentionally not reset, so the caps recalculate inside the active window against the updated freed basis. That covers the recompute. It does not cover the two writers the latch never sees, which is why we report it as a defect rather than an accepted design choice.

AFFECTED CODE

- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/ZPZCoin.sol#L1452>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/ZPZCoin.sol#L1467>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/ZPZCoin.sol#L1595>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/ZPZCoin.sol#L801>

IMPACT

A protected wallet can move more than the documented ten percent per 30 days inside one unexpired window, with no cumulative bound and the window anchor never advanced, so the protocol's headline metering of the 2,000,000,000 ZPZ development and 1,600,000,000 ZPZ treasury allocations becomes a renewable rate rather than a cap. Through the propagation route the cooperating credit holder loses nothing because `ZPZCoin.sol:1599-1600` is zero-sum; through the staking route the seller supplies its own basis; through the bp setter the owner widens every open window at once with one transaction and no event tying it to the wallets it affects. The loss is borne by every other holder as sell pressure the regime was written to spread over months, and the protocol is left with two sell-limit counters whose denominators can be moved under them by three parties, one of which the contract's own latch was written to stop.

REMEDIATION

Scale the counters when the basis or the basis-point parameter changes: on every write to `freedAmount` , in `_recordFreedAmount` and in the propagation, scale `protectedSoldToday` and `protectedSoldThisMonth` by the ratio of the old basis to the new, or reset the window anchors. Re-key the latch onto the recipient as well as the sender and evaluate it from `_recordFreedAmount` so the staking-credit route cannot widen an open window. Alternatively store the ceilings with the window rather than recomputing them: snapshot the daily and monthly limits when the anchor is set and compare against the snapshot, so a later basis change takes effect in the next window only. Apply the same treatment in `setProtectedDailySellLimitBp` .

RETEST

Fixed by stopping a live ceiling from being enlarged once it is consumed within a window. Verified fixed in `390d071` .

The vesting origin is caller-supplied and unbounded, so a schedule can be fully vested at creation

DESCRIPTION

The contract advertises a 180-day cliff and a 720-day linear term in both its constants (`DEVELOPMENT_CLIFF_DURATION` , `DEVELOPMENT_VESTING_DURATION`) and its `NatSpec`, which describes "the same 6-month cliff and 24-month linear vesting policy". Both are measured from an origin the creating transaction supplies: `createDevelopmentVesting` stores `vesting.startTime = startTime < 1 ? uint48(block.timestamp) : startTime` with no comparison against `block.timestamp` in either direction, and `createTreasuryVesting` stores it identically. `vestedDevelopmentAmount` then measures the cliff and the linear term from that stored origin and returns `vesting.totalAmount` once elapsed reaches the duration, so an origin set 900 days in the past satisfies the entire schedule at creation. Reproduced: a 50,000,000e18 development schedule created with an origin 901 days back reports 50,000,000e18 vested immediately and releases in full in the same block.

The metering that might otherwise bound the released balance does not apply. `_enforceDailySellLimit` routes to the protected windows only when the recipient is a registered AMM pair, and `automatedMarketMakerPairs` is empty in the shipped state, so a wallet-to-wallet move of the released allocation meets no protected cap.

Creation is `onlyOwner` , and an owner able to do this can already mint within `MAX_SUPPLY` , so the defect is not a route to tokens the owner could not otherwise obtain. What it removes is the verifiability of the commitment: the stored origin is the only record, no event carries the policy the schedule was meant to satisfy, and nothing on-chain distinguishes a genuine two-year lockup from one back-dated to release immediately.

AFFECTED CODE

- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/ZPZCoin.sol#L907>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/ZPZCoin.sol#L963>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/ZPZCoin.sol#L1299>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/ZPZCoin.sol#L1306>

IMPACT

A development or treasury allocation published as locked for two years can be fully released in the block its schedule is created, and no on-chain record distinguishes that from an honest schedule. Holders relying on the published dilution timetable have no way to verify it was ever enforced, and the protocol cannot demonstrate after the fact that any allocation was actually locked. No tokens are obtained that the owner could not mint directly; the loss is the credibility of the lockup rather than the supply itself.

REMIEDIATION

Bound the stored origin in both `createDevelopmentVesting` and `createTreasuryVesting`: require `startTime = 0 || startTime ≥ block.timestamp`, and cap how far forward it may be set. If back-dating is genuinely required for an off-chain agreement, make it an explicit, bounded parameter with its own event rather than a silent property of `startTime`, so the deviation is visible on-chain.

RETEST

Fixed by bounding the stored vesting origin against `block.timestamp`. Verified fixed in `390d071`.

A whitelisted stake and exit cycle moves the reward treasury out

DESCRIPTION

The caller must hold a staking whitelist entry, which only the owner grants, so this is a participant of the double-reward programme rather than an arbitrary address. Within that entry there is no further restraint: a whitelisted early exit carries no minimum holding period. `unstake` sends any stake younger than `LOCK_PERIOD` through `_withdrawEarlyPrincipal`, which returns the full principal in the same transaction with no penalty, and `_removeWhitelistedPrincipal` then clears `_userWhitelistedPrincipal` so the per-wallet cap resets and the sequence repeats.

Each cycle moves treasury capital. When the pool surplus is exhausted the reserve pulls the whole bonus from `rewardTreasury` through `_safeTransferInExact`, and on the early exit `_forfeitMaturityBonus` reads the recorded provenance, takes the full contribution back through `_takeTreasuryBonus` and credits `treasuryRefunds[fundingTreasury]` rather than returning it to `_rewardPoolBalance`. Three cycles of a 10,000,000 stake move 30,000,000 from the treasury into `treasuryRefunds`, and the caller's balance is unchanged at 10,000,000 each time, so the only cost is gas. Transfers into and out of the controller are exempt from the token's sell meters, so nothing on the token side slows the loop.

Recovery needs an owner transaction per drain, because `withdrawTreasuryRefund` is the only route out of `treasuryRefunds` and `recoverTokens` counts `treasuryRefundBalance` as reserved. Two cycles against a treasury of 25,000,000 leave it holding 5,000,000 against 20,000,000 of refunds, and an honest whitelisted staker's next `stakeWhitelisted` then reverts inside the transfer that funds the bonus. The owner can stop a known offender with `setWhitelist`, so the exposure is bounded by how quickly the drain is noticed.

AFFECTED CODE

- <https://github.com/EZPZTRADING/final-audit/blob/a908e33/contracts/scan/NewSTaking.sol#L824>
- <https://github.com/EZPZTRADING/final-audit/blob/a908e33/contracts/scan/NewSTaking.sol#L1113>
- <https://github.com/EZPZTRADING/final-audit/blob/a908e33/contracts/scan/NewSTaking.sol#L1171>
- <https://github.com/EZPZTRADING/final-audit/blob/a908e33/contracts/scan/NewSTaking.sol#L1232>
- <https://github.com/EZPZTRADING/final-audit/blob/a908e33/contracts/scan/NewSTaking.sol#L1859>

IMPACT

One whitelisted participant closes the programme to everyone else for the price of gas. The caller risks nothing: principal round-trips in full and the forfeited bonus was funded by the treasury, not by them. The treasury's working balance becomes refund liabilities that only the owner can clear, one transaction at a time, and the caller recreates them faster than the owner clears them. No tokens are stolen and the owner can revoke the whitelist entry once the pattern is seen, so the cost is the programme's availability until that happens.

REMEDICATION

Require a minimum stake age before a whitelisted early exit, so a stake and exit sequence cannot move treasury capital for the price of gas. Charge the forfeiting staker for the treasury funding their exit releases, or return a treasury-funded forfeiture to the reward pool when the stake is younger than that minimum age.

RETEST

Fixed by charging a 1% early-exit penalty on gross whitelisted principal through both exit paths, rounded up so dust withdrawals cannot avoid it, and by reusing current-treasury refunds before drawing fresh capital. A 10,000,000 stake exiting after 30 days keeps 9,900,000, and the fee stays in the reward pool. Section 4.3 of the whitepaper still states that no principal penalty applies. Verified fixed in `18646ef`.

Rebinding the controller blocks every exit that carries credit

DESCRIPTION

Two credit hooks treat a refused write as fatal. `_recordMarketFreedAmount` reverts with `StakingProtectionHookFailed` and `_restorePrincipalCredit` reverts with `StakingCreditRestorationFailed`, each when `_callMarketProtection` returns false, with no fallback and no event. The audited contracts emitted `MarketProtectionHookFailed` at four sites instead and never reverted. A third writer, `_retirePrincipalCredit`, is guarded on the reported accounting version and does nothing when the token offers no accounting, so the writers of the same credit disagree on how to handle a token that refuses the write.

Pointing `marketProtectionController` at a replacement staking contract is enough to trigger it. The outgoing staking proxy is then neither the bound controller nor a registered system contract, so the token's `_checkAuthorizedUpdater` rejects it. Measured after a rebind: `unstake` before maturity and `withdrawPrincipal` revert with `StakingProtectionHookFailed`, and `withdrawUnbondedPrincipal` reverts with `StakingCreditRestorationFailed`. `requestUnstake` still succeeds, so a regular staker passes the cooldown and is stopped only at the withdrawal. A staker whose 10,000,000 principal has already finished its 14-day cooldown cannot withdraw it, and `recoverTokens` counts `_pendingUnbondingBalance` in its reserved sum so the owner cannot move it either.

The condition is reversible and not a permanent loss. Pointing the controller back restores the withdrawal immediately, and `_checkAuthorizedUpdater` also accepts a registered system contract, so registering the outgoing staking proxy in `systemContracts` before the rebind avoids the freeze while the replacement takes the controller slot. Nothing in the code enforces or records that ordering.

AFFECTED CODE

- <https://github.com/EZPZTRADING/final-audit/blob/a908e33/contracts/scan/NewSTaking.sol#L1340>
- <https://github.com/EZPZTRADING/final-audit/blob/a908e33/contracts/scan/NewSTaking.sol#L1407>
- <https://github.com/EZPZTRADING/final-audit/blob/a908e33/contracts/scan/NewSTaking.sol#L1412>
- <https://github.com/EZPZTRADING/final-audit/blob/a908e33/contracts/scan/NewSTaking.sol#L1415>
- <https://github.com/EZPZTRADING/final-audit/blob/a908e33/contracts/scan/ZPZCoin.sol#L870>

IMPACT

An ordinary migration step freezes principal that has already served its cooldown, for every staker who ever held released sell credit, and the owner cannot recover those tokens while the freeze holds. The remedy is an undocumented ordering constraint on two owner transactions, so the failure appears at the moment the team is least likely to be watching the old contract and most likely to read the reverts as a problem with the new one.

REMEDIATION

Record a failed credit restoration instead of reverting the withdrawal, matching the version-guarded behaviour of the retirement path, and emit an event so the shortfall is visible off chain. The principal transfer should not depend on an accounting hook the token may legitimately refuse.

RETEST

Fixed by routing all three credit writers through `_settleOrDeferMarketCredit`, which records a refused write in `pendingMarketCredit` and emits rather than reverting the payout, and by auto-retaining an outgoing deployed controller as a system contract. Owner-only `reconcileMarketCredit` retries or acknowledges the debt. Ten million of credit settled exactly across two early withdrawals against the real token with no deferral. Verified fixed in `18646ef`.

A whitelisted stake drains the shared reward pool

DESCRIPTION

`_reserveMaturityBonus` reserves the 100% whitelisted bonus out of the same `_rewardPoolBalance` that backs all APY claims. Its treasury top-up computes the deficit as `amount - _rewardPoolBalance`, the new bonus only, never the accrued-but-unpaid APY liability, then deducts the full amount. A whitelisted stake at or above the current pool balance therefore zeroes `_rewardPoolBalance` while accrued rewards are still outstanding.

Every regular staker's `claimReward` then hits the `Insufficient reward pool` require and reverts; `claimReward` has no skip path, unlike the exit flows which use `_payRewardIfAvailable`. Claims stay blocked until the owner calls `fundRewardPool` or a whitelisted stake forfeits its bonus. The intent to protect this liability exists elsewhere: `withdrawRewardPool` subtracts the accrued liability before permitting a withdrawal, so the omission in the reservation path is the defect, not the design. No funds are lost: accrued APY is banked in the stake's `accruedReward` on unstake and is never zeroed, and `claimReward` pays it in full from an inactive stake once the pool is funded.

AFFECTED CODE

- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/NewSTaking.sol#L1901>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/NewSTaking.sol#L1910>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/NewSTaking.sol#L1058>

IMPACT

Reward claims for the entire non-privileged staker class are denied until the owner funds the pool, and the condition recurs on every whitelisted stake, so the APY program degrades to availability windows between top-ups until the code is fixed. No funds are lost and principal is never at risk: every accrued reward is recoverable in full once the pool is funded, and the failure is loud rather than silent.

REMEDIATION

Account maturity bonuses in a reserve separate from the APY pool, or make the reservation top the pool up to `amount` plus the accrued reward liability so settled rewards always remain payable, mirroring the protection `withdrawRewardPool` already applies.

RETEST

Fixed by capping the entrant's APY top-up at `Math.mulDiv(amount, APY_RESERVE_PPM, 1000000, Math.Rounding.Ceil)`, its own envelope, so existing principal can no longer enlarge the treasury pull. Measured on the round-4 scenario: after a 76,000,000 regular stake a 10,000,000 whitelisted entry needs 13,498,590 rather than 30,087,874 and succeeds. Four consecutive entries against a 99,000,000 deficit

each paid the same amount and left the shortfall unchanged. The original defect stays closed. Verified fixed in 0ec1a92.

The whitelisted staking interval inverts and the program is unenterable

DESCRIPTION

`stakeWhitelisted` enforces two bounds derived from different sources. The lower bound is `_effectiveWhitelistedMinStake()`, which returns `max(_minStakeAmount, WHITELIST_MIN_STAKE)` and can never fall below the 10,000,000e18 constant. The upper bound is `_effectiveWhitelistedMaxStake()`, which follows the owner-set `_maxStakeAmount` downward with no floor. Any non-zero `_maxStakeAmount` below 10,000,000e18 therefore produces an empty interval (zero means "no maximum" and resolves to `WHITELIST_MAX_STAKE`): `stakeWhitelisted` requires the amount to be at least 10,000,000e18 and simultaneously at most `_maxStakeAmount` whenever that is the stricter bound, so no value satisfies both. Every call reverts, with "Stake amount below minimum" under the threshold and "Stake amount exceeds maximum" at or above it.

No guard catches this. `initialize` checks only `maxStake ≥ minStake` against the caller-supplied global minimum, so a maximum of 10,000e18 against a minimum of 1,000e18 passes while the whitelisted interval it implies is empty. `setMinStakeAmount` and `setMaxStakeAmount` compare against each other but never against `WHITELIST_MIN_STAKE`, and `whitelistedStakeStatus` reports the two bounds without comparing them. The flagship three-year double-bonus program is inert with no on-chain signal that anything is wrong.

AFFECTED CODE

- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/NewSTaking.sol#L1932>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/NewSTaking.sol#L1921>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/NewSTaking.sol#L531>

IMPACT

Under any configuration where `maxStakeAmount` is non-zero and below 10,000,000e18 the whitelisted staking program cannot be entered by anyone. That is the configuration every deployment script in the repository produces: `deploy.new.js` and `deploy.sepolia.js` both pass `maxStakeAmount = 10,000e18` against `minStakeAmount = 1,000e18`. Every attempt reverts, the failure is silent until a user spends gas, and no initializer, setter or status view reports the contradiction. Recovery requires the owner to notice and raise `setMaxStakeAmount` to at least 10,000,000e18; nothing in the deployment flow prompts this.

REMEDIATION

Floor `_effectiveWhitelistedMaxStake()` at `WHITELIST_MIN_STAKE` so the whitelisted interval can never invert, and validate in `initialize`, `setMinStakeAmount` and `setMaxStakeAmount` that the resulting interval is non-empty, reverting rather than silently disabling the program. Have

`whitelistedStakeStatus` surface an explicit unavailable status when the interval is empty, and set the deployment scripts' `maxStakeAmount` to a value consistent with `WHITELIST_MAX_STAKE` .

RETEST

Fixed by normalising the effective whitelist minimum so invalid global bounds cannot disable the programme. Verified fixed in `390d071` .

An all-or-nothing maturity claim above the per-transfer ceiling locks the release permanently

DESCRIPTION

`_claimMaturityRelease` pays the entire outstanding entitlement in one transfer: `claimable = vested - release.releasedAmount`, followed by a single `_safeTransferOutExact(stakingToken, caller, claimable)`. No partial-claim entry point exists: `claimMaturityRelease` takes only a release id, and the matured `unstake` and `withdrawPrincipal` paths reach the same helper. The token rejects any transfer above `maxTransactionAmount`, defaulting to 50,000,000e18, with the staking contract as sender.

Because `claimable` grows monotonically with elapsed tranches and is never reduced except by a successful claim, a staker who lets it cross the ceiling can never claim again. For a 50,000,000e18 whitelisted stake the schedule totals 100,000,000e18: a 15% immediate tranche of 15,000,000e18 and thirteen monthly tranches of 6,538,461e18. After the immediate tranche is paid at `unstake`, skipping six tranches leaves 39,230,769e18 claimable, still below the ceiling; skipping eight leaves 52,307,692e18, and every `claimMaturityRelease` from that point reverts `MaxTransactionExceeded`. At full vest the figure is 85,000,000e18 and the position is permanently unreachable. The variant that defers the exit is worse: `unstake` itself pays the accumulated entitlement, so a staker who does not `unstake` until roughly eight months past the maturity date carries every accumulated tranche in one payout and `unstake` reverts; past thirteen months that payout is the whole doubled principal, so any stake above 25,000,000e18 cannot be exited at all. Unstaking at the maturity date is unaffected, because only the 15% immediate tranche has vested at that moment.

Three preconditions apply. The position must exceed 29,400,000e18: the fully accumulated payout is 1.7P, so only above that figure does it clear the 50,000,000e18 ceiling, and stakes between the 10,000,000e18 minimum and that threshold are never affected however long they are left. The staking contract must not be token-whitelisted, since the anti-bot block nests under `!whitelisted[from]` then `!whitelisted[to]`: `script/deploy.sepolia.js` does whitelist it, while `deploy.new.js`, `deploy.EZPZStakingDA0.js` and `deploy.DA0.js` do not. And the finding is latent while the whitelisted program is unenterable under the shipped stake limits, because no maturity release can exist; it activates the moment that is fixed, so the two must be addressed together.

The same block breaks a second path through a different check: four exit routes transfer twice from the staking contract in one call the APY through `_payRewardIfAvailable`, then principal or the first tranche so the second leg sees zero elapsed time against the 30-second cooldown and reverts `CooldownNotExpired`, rolling back the exit. It fires whenever an APY reward is payable; the staker recovers unaided by claiming separately first.

AFFECTED CODE

- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/NewSTaking.sol#L1636>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/NewSTaking.sol#L1683>

- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/ZPZCoin.sol#L469>

IMPACT

A whitelisted staker holding more than 29,400,000e18 who skips roughly eight monthly claims, or who defers the exit by about eight months past maturity, loses access to between 85% and 100% of their principal and bonus. The amount owed only grows, so the condition never self-heals and no user-side action recovers it: `claimMaturityRelease` takes no amount parameter, so the staker cannot draw a payable slice. Nothing is destroyed: the tokens remain in the contract and are released in full once the owner whitelists the staking contract or raises `maxTransactionAmount`, but the affected user has no way to trigger that recovery.

REMEDIATION

Exempt transfers where the market-protection controller is the sender or recipient from the sender-side anti-bot block in `ZPZCoin._update`, so protocol payouts are not sized against a per-trader cap. This is one added condition and leaves the protected-seller caps untouched, since they run earlier in `_update`; it also resolves the staking payout throttling and the staking deposit ceiling documented elsewhere in this report. An amount parameter on `claimMaturityRelease` does not suffice alone, because `unstake` and `withdrawPrincipal` claim the full entitlement internally.

RETEST

Fixed by exempting controller payouts from the per-transfer ceiling, so the `maxTransactionAmount` check now sits under `!isSenderExempt`. A 50,000,000 stake claimed 52,307,692 in one call, the figure that previously reverted. Verified fixed in `a908e33`.

Resetting the DEX launch time re-indexes every wallet's daily sell meter

DESCRIPTION

`setDexLaunchTime` accepts any non-zero `uint48` and is re-settable at will, with no monotonicity or bounds check. The launch-window meter buckets time as `currentDay = (currentTime - launchTime) / 1 days` and compares it against the stored `lastSellLimitDay[from]`, resetting `soldToday` and re-snapshotting `dailySellLimitBasis` whenever they differ. Per-wallet accounting is therefore keyed to a mutable anchor.

No bad intent is needed to trigger the defect: correcting a mistyped timestamp, or pushing the launch back a day, re-indexes every wallet at once with two opposite outcomes. Where the shifted index differs from a wallet's stored value the meter is zeroed and the basis re-read, granting a fresh 1% allowance with no time elapsed, so a holder who has just sold its full daily limit can immediately sell roughly that much again. Where the shifted index collides with the stored value no reset fires, so a consumed `soldToday` and a stale basis persist into what is effectively a new day and that wallet cannot sell its legitimate allowance. The damage cannot be repaired, because `delete soldToday[from]` is destructive and re-anchoring only triggers a further reset.

AFFECTED CODE

- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/ZPZCoin.sol#L680>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/ZPZCoin.sol#L1408>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/ZPZCoin.sol#L1413>

IMPACT

On the day the anchor moves, per-wallet sell accounting desynchronises from wall-clock time in both directions: some wallets obtain an extra 1%-of-balance tranche on top of what they have already sold, others are locked out of a day's legitimate allowance, and the 1% is computed against a stale balance snapshot in either case. The effect is bounded to roughly one extra tranche per anchor change, applies only inside the sixty-day window, risks no funds and cannot be triggered by a user, but no owner action restores the correct state once it has happened.

REMEDIATION

Make `dexLaunchTime` write-once, or reject any value that changes an already-started window. If re-anchoring must be supported, migrate or invalidate `lastSellLimitDay` and `dailySellLimitBasis` in the same transaction.

RETEST

Fixed by making `setDexLaunchTime` write-once. Verified fixed in `390d071`.

The TWAP output floor ignores the pool's own price impact

DESCRIPTION

`_twapMinOutFromObservation` derives the floor as `Math.mulDiv(amountIn, priceAverageQ112, 1 << 112)` reduced by `maxSlippageBp` (`BuybackRevenueSplitter.sol:579`). That is a linear projection of the time-averaged spot price across the whole input amount, with no constant-product term, so it does not model the price impact the swap itself causes. A flat basis-point allowance cannot cover a term that grows with trade size.

For a constant-product pool a trade of fraction f of the input reserve returns about $0.997f / (1 + 0.997f)$ of the output reserve against a linear f , so at the default five percent allowance the floor becomes unreachable once the trade exceeds roughly 4.96 percent of pool depth. On a balanced pool a trade worth ten percent of depth requires 95,000 units of output while the pool can return only about 90,661, and the swap reverts `Insufficient ZPZ bought` on a pool nobody has touched. The floor sits in a helper shared by both oracles, so `convertNativeRevenueToPaymentToken` carries the identical ceiling; the buyback path is additionally unreachable while the burn floor sits above total supply, whereas the native path is not.

AFFECTED CODE

- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/BuybackRevenueSplitter.sol#L579>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/BuybackRevenueSplitter.sol#L627>

IMPACT

Buybacks above roughly five percent of pool depth revert with no manipulation present, and the revert is the same one a genuine attack produces, so the operator cannot distinguish the two. Sizing trades smaller is a complete workaround and no capital is stranded, which bounds the direct harm to wasted gas on the first attempt. The indirect cost is larger: the natural remedy is raising `maxSlippageBp` toward its 2,000 ceiling, which admits a twenty percent worse execution and so weakens the same floor the manipulation findings rely on.

REMEDIATION

Apply the constant-product formula to the time-averaged price using live reserves before subtracting `maxSlippageBp`, or compare the averaged price against the swap's realised execution price rather than projecting it linearly across `amountIn`.

RETEST

Fixed by sizing the output floor with the constant-product formula instead of projecting the averaged price linearly across the input. Verified by code review in `390d071` .

Protection hooks fail open and pay out without sell caps

DESCRIPTION

The staking contract calls the token's market-protection hooks through `_callMarketProtection`, which returns the raw call's success flag (`NewSTaking.sol`); every caller converts a failure into an event and continues to transfer the principal, reward or tranche in full; the entry-side consumption hook swallows failures the same way (`NewSTaking.sol`). The capability probe `_tokenHasFreedAccounting` checks only `marketProtectionAccountingVersion()` (`NewSTaking.sol`), a `pure` function returning a constant, so the staking contract believes accounting is active even when every state-changing hook reverts.

The trigger is any state in which the token's `marketProtectionController` is not this contract unset at deployment, repointed during a migration, or changed in an upgrade none of which the staking contract detects. Every `markMarketProtectedSeller` call then reverts "Not controller" (`ZPZCoin.sol`), the event is emitted, and whitelisted principal of 10,000,000e18 or more leaves the contract with the recipient unmarked and `freedAmount` zero. The result is not a weakened cap but no cap: such a holder transfers directly to a registered pair with `protectedSoldToday` still at zero, because `_enforceProtectedSellLimits` is never reached.

AFFECTED CODE

- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/NewSTaking.sol#L1559>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/NewSTaking.sol#L1374>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/NewSTaking.sol#L1545>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/ZPZCoin.sol#L775>

IMPACT

Whenever the controller binding is absent or wrong, every staking exit pays out tokens with no dump protection attached, and a recipient of a 10,000,000e18 whitelisted position can sell straight into a registered pair with no daily or monthly cap applied. The failure is silent: payouts succeed normally and the only signal is an event with no on-chain consumer, so unlike the throttling defects, which announce themselves by reverting, this one reports success while doing nothing. Correcting the binding closes the current window but not the defect, since any later repointing reopens it.

REMEDIATION

Fail closed: revert the payout when a protection hook call fails on an exit path instead of emitting an event, and verify at stake time that the token's controller is actually this contract by performing a state-changing no-op check rather than reading a constant version function.

RETEST

Fixed by making the protection hooks fail closed: the four `MarketProtectionHookFailed` emissions were replaced by `StakingProtectionHookFailed` and `StakingCreditRestorationFailed`, so an exit can no longer pay out with no cap attached. Verified fixed in `a908e33`.

A whitelisted reward claim enrolls the claimer and freezes the wallet

DESCRIPTION

`_recordRewardFreedAmount` routes whitelisted-program rewards through `_recordMarketFreedAmount` unconditionally (NewSTaking.sol), and the token's `_recordFreedAmount` sets `protectedMarketSellers[account] = true` whenever it credits (ZPZCoin.sol). The regular-stake branch correctly uses the status-checking `_recordMarketFreedAmountIfProtected`; the whitelisted branch does not.

A wallet holding 1,000,000e18 of freely transferable, market-bought ZPZ stakes the 10,000,000e18 whitelisted minimum and later claims a year's APY of 1,700,000e18. The claim marks it protected with a basis equal to the reward, so the wallet holds 2,700,000e18 against 1,700,000e18 of credit: every transfer above that credit to an unprotected wallet reverts `ProtectedTransferExceedsFreedCredit`, immobilising exactly the 1,000,000e18 it already owned. The reward itself stays liquid, because the credit covers it; the restriction falls entirely on the holdings the program never released.

The condition is self-recoverable but undocumented. Staking the immobilised balance succeeds, because the controller pull is exempt from propagation, and after the fourteen-day unbonding the withdrawal returns credit equal to the full balance, freeing it. Nothing in the contract or its documentation indicates this, and the balance the holder must stake is the one that appears broken. An owner unmark is the only alternative, and the claimer never consented to enrollment or received any warning.

AFFECTED CODE

- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/NewSTaking.sol#L1470>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/ZPZCoin.sol#L1619>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/NewSTaking.sol#L1475>

IMPACT

A routine, protocol-encouraged action immobilises the claimer's entire pre-existing balance behind a reward-sized cap for at least fourteen days, affecting every whitelisted participant who also holds market-bought tokens. No tokens are lost and the holder can recover unaided, but the escape is undocumented and counter-intuitive, so in practice affected users are blocked until support explains it.

REMEDIATION

Route whitelisted reward claims through the status-checking branch so a claim never marks a new wallet protected, or require explicit opt-in to the protected-seller program before the first whitelisted claim is credited.

RETEST

Fixed by no longer enrolling the claimer on a reward claim. Verified fixed in 390d071 .

Snapshot voting power outlives the stake entirely

DESCRIPTION

`createProposal` sets `snapshotTimestamp = currentTime - MIN_VOTING_DELAY`, one hour before creation (ZPZCoinDAONew.sol), and imposes no upper bound on `endTime` beyond a three-day minimum voting period (ZPZCoinDAONew.sol). `vote` resolves power through `stakingContract.getPastStake(voter, proposal.snapshotTimestamp)` (ZPZCoinDAONew.sol), and because `_writeCheckpoint` records history, a voter who fully exits after the snapshot retains undiminished power for the entire voting window.

The sequence is: stake, wait one hour, have a long-dated proposal created, withdraw the capital in full, and cast the snapshotted credits weeks later with no remaining economic exposure. On the regular program the exit costs the fourteen-day unbonding period; through the whitelisted early-exit path the principal returns in the next block, reducing the capital lock to a single block, though that path requires at least 10,000,000e18 and owner whitelisting in the staking program. Proposal eligibility compounds the divergence by reading live stake through `_isEligibility` (ZPZCoinDAONew.sol) while voting reads the snapshot.

Because the snapshot is backdated rather than taken at creation, the same mechanism also admits a stake that was closed *before the proposal existed*. A wallet stakes at T and calls `requestUnstake` at T+300, which writes a zero checkpoint (NewSTaking.sol); a proposal created at T+3600 anchors its snapshot at T, and the checkpoint search returns the pre-exit balance (NewSTaking.sol). The wallet votes its full sqrt-weighted credits having held zero staked tokens from before the proposal was drafted until after voting closed. A creation-time snapshot would have rejected the same sequence with "No stake at snapshot".

AFFECTED CODE

- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/ZPZCoinDAONew.sol#L333>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/ZPZCoinDAONew.sol#L314>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/ZPZCoinDAONew.sol#L400>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/ZPZCoinDAONew.sol#L567>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/NewSTaking.sol#L2195>

IMPACT

Nothing on-chain depends on the outcome. ZPZCoinDAONew contains no low-level call, `executeProposal` sets `proposal.executed` and emits `ProposalSignalFinalized`, that flag is read nowhere else, and the DAO address holds no owner or controller role in the token or staking contracts, so a captured proposal produces an event and no state change anywhere else. Within that ceiling the defect is real and unprivileged: voting power is decoupled from economic exposure, so any wallet can acquire stake, obtain a snapshot, recover its capital in full, and still carry an outcome weeks later with nothing at risk including a

wallet whose stake was already closed before the proposal was drafted. The harm is to the integrity of the published signal, which off-chain processes may treat as genuine community input.

REMEDIATION

Bound the voting window to a fixed maximum, and weigh each vote by the lesser of the snapshotted stake and the stake held at vote time so exiting forfeits power that single change closes both the exit-after-snapshot and the exit-before-creation variants. Read proposal eligibility from the same snapshot voting uses. Moving `snapshotTimestamp` to the creation timestamp is worth doing but fixes only the backdated half.

RETEST

Fixed by flooring vote weight against live stake. Verified fixed in `390d071`.

The APY reserve rate ignores rewards claimed and staked again inside a year

DESCRIPTION

`protectedRewardPoolBalance` fences the reward pool at 326312 parts per million of staked principal. That rate is the programme's total under annual compounding of unpaid rewards: 17%, then 9% on principal plus year one, then 4% on the running total, which is 32.6312% exactly. It holds only while rewards stay unpaid inside the stake that earned them.

`_rewardThrough` charges every stake the calendar year's rate from its own start time, with no regard for stake age, so a reward claimed mid-year and staked again earns that year's rate for the remainder of the year on top of what the original stake keeps earning. Measured over the full three years on a 10,000,000 stake, a holder who never claims receives 32.6122% of principal while a staker who claims once inside year one and stakes the proceeds receives 33.4296%, which is already above the fence. The supremum under any claim and restake frequency is the continuous bound $e^{0.30} - 1$, or 34.9859%, so the fence understates the maximum liability by up to 2.36 percentage points of staked principal. Exact figures move by a few basis points with block timing.

The gap becomes a shortfall because `withdrawRewardPool` is capped at pool balance minus the fence as it stands at that moment, and the withdrawal is irreversible. The fence is a snapshot: a claim lowers the pool and the restake that follows raises `_totalStaked`, so the requirement grows after the owner has already taken everything above the earlier figure. With the pool stripped to the fence immediately after staking, a single mid-year claim and restake leaves one position owed 195,189 that the contract cannot pay: the claim reverts with `StakingInsufficientRewardPool` while the other two positions are paid in full. The fence does rise once the restake lands, which stops further withdrawals, but it recovers nothing already taken.

AFFECTED CODE

- <https://github.com/EZPZTRADING/final-audit/blob/a908e33/contracts/scan/NewSTaking.sol#L1824>
- <https://github.com/EZPZTRADING/final-audit/blob/a908e33/contracts/scan/NewSTaking.sol#L1016>
- <https://github.com/EZPZTRADING/final-audit/blob/a908e33/contracts/scan/NewSTaking.sol#L1030>
- <https://github.com/EZPZTRADING/final-audit/blob/a908e33/contracts/scan/NewSTaking.sol#L1740>
- <https://github.com/EZPZTRADING/final-audit/blob/a908e33/contracts/scan/NewSTaking.sol#L1853>

IMPACT

The owner withdraws tokens the contract reports as surplus, within the guard the contract provides, and the loss falls on the staker who did nothing. Under monthly claim and restake with the owner taking the maximum at every step, the churning staker is paid in full throughout 356 cycles while the staker who never claims is left holding an unpayable claim for the whole of their three-year APY, 3,260,863 on a 10,000,000 stake. The amount is deferred rather than destroyed: the accrued balance survives, no expiry applies, and `fundRewardPool` restores payment in a single owner transaction. The cure is outside the contract's control, because the tokens have already left it and the owner is under no on-chain obligation to return them. The same understated fence also sizes the maturity bonus reserve, so it widens the funding gap on the whitelisted programme.

REMEDIATION

Raise the reserve rate to the continuous bound of 349859 parts per million, the supremum of the compounded programme under any claim and restake frequency. Applied to this commit and re-run under the same monthly churn with maximum owner extraction, every claim then pays and no position is left unfunded. Leave the accrued liability term unchanged.

RETEST

Fixed by raising the reserve envelope from 326312 to 349859 parts per million, the upper bound for arbitrarily frequent claim and restake at the fixed rates, and applying it in the protected balance, the bonus surplus and the incoming-stake fence. Section 4.2 of the whitepaper still gives 32.6312% as the maximum cumulative yield. Verified fixed in `18646ef`.

A protected wallet that stakes all its credit cannot move any token

DESCRIPTION

Staking consumes a protected wallet's free credit and records it as custody, and `_consumeFreedAmount` keeps the protected flag alive whenever that custody is non-zero. The sell ceiling is then computed from the free credit alone, so `_newProtectedCeiling(0, bp)` returns zero and every outgoing transfer reverts, whatever the wallet actually holds.

A wallet with 100,000 of credit and 5,000,000 of tokens that stakes exactly its credit cannot send one wei: `freedAmount` falls to zero, custody holds 100,000, the flag stays set, and a transfer reverts with `ProtectedDailySellLimitExceeded`. The 4,900,000 it never staked is caught as well, because the ceiling is keyed on credit rather than on holdings. `clearEmptyProtection` cannot help, since it requires zero custody and reverts with `NotAuthorized`.

The wallet is not trapped. Credit returns on exit, so a regular staker clears the condition after the fourteen-day cooldown and a whitelisted staker clears it immediately through an early exit, and the owner can lift it with `setProtectedMarketSeller`. The cost falls on the whitelisted case, where the only self-service remedy is an early exit that forfeits the entire maturity bonus. The dust rule sharpens the same edge: a positive basis that rounds below one wei is paced at one wei for the whole window.

AFFECTED CODE

- <https://github.com/EZPZTRADING/final-audit/blob/a908e33/contracts/scan/ZPZCoin.sol#L1519>
- <https://github.com/EZPZTRADING/final-audit/blob/a908e33/contracts/scan/ZPZCoin.sol#L1524>
- <https://github.com/EZPZTRADING/final-audit/blob/a908e33/contracts/scan/ZPZCoin.sol#L1568>
- <https://github.com/EZPZTRADING/final-audit/blob/a908e33/contracts/scan/ZPZCoin.sol#L787>
- <https://github.com/EZPZTRADING/final-audit/blob/a908e33/contracts/scan/NewSTaking.sol#L1379>

IMPACT

A vesting beneficiary or reward recipient who stakes the credit they were released loses the ability to transact at all, including with tokens bought on the market that the credit basis never covered. Nothing in the staking entry points signals this, and the whitelisted staker's choice is to stay immobile for the three-year lock or to forfeit the hundred percent maturity bonus to regain the use of their own balance.

REMEDIATION

Size the protected ceiling from the wallet's holdings as well as its released credit, so a wallet whose credit is in custody is paced rather than stopped. Alternatively surface the consequence at the staking entry point and reject a stake that would consume the caller's entire free credit.

RETEST

Fixed by sizing the basis from the credit ledger alone: `_protectedAllowanceBasis` returns `freedAmount + stakedProtectedCredit`. Measured: a wallet with 1,000,000 of credit opens a 10,000 daily ceiling whether it holds 10,000,000 or 100,000,000, so the tenfold amplification is gone, and staking through the unpaced controller path no longer detaches a frozen ceiling from its basis. A wallet whose credit is all in custody is paced at 1,000 daily rather than frozen at zero, so the original defect stays closed and the section 6 conflict is resolved. Verified fixed in 0ec1a92.

The maturity release schedule is anchored in the past

DESCRIPTION

`_createMaturityRelease` sets `release.startTime = maturityDate`, which is `s.startTime + LOCK_PERIOD` and therefore a timestamp that has already passed by the time the schedule is created. `_vestedMaturityAmount` counts elapsed tranches from that anchor, and `_maturityReleaseTerms` ignores its only parameter, so nothing re-anchors the schedule to the moment it comes into existence. A schedule created late is already partly or wholly elapsed.

The terms are a 15% immediate release followed by thirteen monthly tranches. A staker who exits at the maturity date receives 3,000,000 of a 20,000,000 entitlement and the rest over thirteen claims. The proportion released on the first claim rises with the delay, because the tranche count is already running: 15.0% at the maturity date, 34.6% three months later, 54.2% at six months, 73.8% at nine months, and the whole 20,000,000 in one transaction once thirteen months have passed. No privileged action is involved.

`_claimMaturityRelease` also calls `_recordMarketFreedAmount` for the amount delivered, so the single claim grants the full principal and bonus as sell credit at once. For a protected wallet that converts a paced release into one step change in its sell ceiling.

AFFECTED CODE

- <https://github.com/EZPZTRADING/final-audit/blob/a908e33/contracts/scan/NewSTaking.sol#L1289>
- <https://github.com/EZPZTRADING/final-audit/blob/a908e33/contracts/scan/NewSTaking.sol#L1316>
- <https://github.com/EZPZTRADING/final-audit/blob/a908e33/contracts/scan/NewSTaking.sol#L1556>

IMPACT

No value is gained or lost and the total wait to full receipt is unchanged, so a staker who delays is not paid sooner. What the defect removes is the pacing: the position arrives as one lump at a moment the holder chooses rather than spread across thirteen claims. A matured whitelisted position measured on this commit credited 23,260,925 of sell credit in a single write, which raises a protected wallet's daily and monthly ceilings in one step and removes the throttle the protected-seller model exists to apply to exactly these positions.

REMEDIATION

Anchor the schedule at the later of the maturity date and the current timestamp when the release is created, so the tranche count always starts at zero. Keep the maturity date as the eligibility test for creating the schedule, and treat the creation time as the point from which tranches accrue.

RETEST

Fixed by anchoring a maturity release at the later of the current timestamp and the maturity date, so the tranche count always starts at zero, with the maturity date retained as the eligibility test. A schedule

created thirteen months late now releases only the immediate 15%. Section 4.4 of the whitepaper still describes the schedule as fixed from maturity. Verified fixed in `18646ef` .

A third buyback inside ten minutes reverts on an immature checkpoint

DESCRIPTION

The contract enforces `MIN_TWAP_WINDOW` spacing on its two permissionless checkpoint rolls (BuybackRevenueSplitter.sol:716 and :729) but not on the two internal rolls that follow a successful swap (BuybackRevenueSplitter.sol:381 and :433). Each roll shifts the current checkpoint into the previous slot and writes a fresh one, and the fallback in `_twapMinOutFromObservation` reaches only one generation back (BuybackRevenueSplitter.sol:558).

Three executions inside ten minutes therefore leave both generations young. A buyback at T rolls the checkpoint to T; a second at T+300 finds the latest immature, falls back to the still-mature previous one, succeeds, and rolls again so the previous slot now holds T; a third at T+540 finds the latest immature at 240 seconds and the previous immature at 540 seconds, and reverts `TWAP observation not ready`. The revert fires in `_enforcedMinOut` before the swap, so nothing is spent and nothing is written, and the condition clears on its own once either generation matures. The same unspaced roll applies to `convertNativeRevenueToPaymentToken`, which carries no burn-floor guard, so that half is reachable while the buyback half is not.

AFFECTED CODE

- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/BuybackRevenueSplitter.sol#L381>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/BuybackRevenueSplitter.sol#L716>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/BuybackRevenueSplitter.sol#L433>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/BuybackRevenueSplitter.sol#L558>

IMPACT

No funds, state or security guarantee are affected. An operator batching executions is blocked for under ten minutes and sees `TWAP observation not ready`, which attributes the failure to the oracle rather than to the contract overwriting its own checkpoint on the previous call, leaving no way to diagnose the cause from the error.

REMIEDIATION

Apply the same `MIN_TWAP_WINDOW` spacing check to both internal rolls that the permissionless entry points already apply, so a buyback never overwrites a checkpoint that is still the only mature observation.

RETEST

Fixed by adding `MIN_TWAP_UPDATE_INTERVAL` of twenty minutes and enforcing it on the internal rolls that follow a swap, so a young checkpoint is never shifted into the previous slot. Verified fixed in `a908e33`.

Skipped reward payments are not signalled, and APY accrues with no funding check

DESCRIPTION

`_settleReward` increases `_accruedRewardLiability` by the newly accrued amount unconditionally (NewSTaking.sol), with no check that `_rewardPoolBalance` can cover it, so promised APY may exceed the pool by construction. The whitelisted bonus is escrowed up front by `_reserveMaturityBonus`; ordinary APY never is.

Once liability outruns the pool the two payout routes diverge without saying so. `claimReward` routes through `_payReward` and hard-reverts `require(_rewardPoolBalance ≥ amount, "Insufficient reward pool")` (NewSTaking.sol). The four whitelisted and partial-withdrawal exits `unstake` (NewSTaking.sol), `withdrawPrincipal`, `_earlyExitWhitelisted` and `_earlyWithdrawWhitelistedPrincipal` route through `_payRewardIfAvailable`, which returns zero when the pool is short (NewSTaking.sol) and lets the exit proceed with no event, no revert and no distinguishing return value. The regular `unstake` path does not attempt payment at all, so it is unaffected.

Nothing is lost, taken or left permanently inaccessible. `delete stakes[stakeId].accruedReward` executes only inside `_payReward` (NewSTaking.sol), so a skipped entitlement survives and is paid in full once the pool is funded. The gap is that the holder is given no indication the payment was skipped and has no reason to return for it.

AFFECTED CODE

- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/NewSTaking.sol#L1080>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/NewSTaking.sol#L1011>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/NewSTaking.sol#L1061>

IMPACT

No funds are lost, stolen or permanently stuck. A staker exiting through one of the four affected paths receives principal, sees a successful transaction, and is not told the reward portion paid nothing; absent an event or a distinguishing return value, neither the interface nor an off-chain indexer can surface the shortfall. Separately, the contract accrues APY obligations it has not verified it can fund, so the gap between liability and pool widens with participation rather than being caught at accrual.

REMEDIATION

Emit a distinct event when `_payRewardIfAvailable` skips a payment, and return the unpaid amount to callers rather than folding it into the paid figure, so the shortfall is visible on-chain and in the

interface. Separately, bound accrual against available funding, or surface the liability-to-pool ratio in a view so underfunding is observable before it reaches a payout.

RETEST

Fixed by the observability route the finding offered: `_payRewardIfAvailable` emits `RewardPaymentDeferred` with the unpaid amount and the available pool, and `protectedRewardPoolBalance` surfaces the backing requirement including unsettled obligations, so underfunding is visible before a payout. Accrual is still not bounded against funding, which the client discloses as residual. Verified fixed in `a908e33` .

Burns bypass the blacklist and the trading switch

DESCRIPTION

`_update` runs every blacklist check and the `tradingEnabled` gate inside the transfer branch only: `if (!tradingEnabled) revert, if (blacklisted[from]) revert, if (blacklisted[to]) revert` and `if (blacklisted[msg.sender]) revert` all sit under `if (to != address(0))` (ZPZCoin.sol). A burn is a movement to the zero address, so control falls to the `else` branch, which validates nothing but the burn floor (ZPZCoin.sol). Neither control is enforced on burns.

The consequence with an identifiable victim is third-party: a platform burner holding an allowance granted before the freeze can call `burnForPlatform(victim, amount)` (ZPZCoin.sol) and permanently destroy a blacklisted account's balance. That function checks only `platformBurners[msg.sender]`, then `_spendAllowance` and `_burn`, neither of which consults the blacklist. The realistic trigger is routine rather than malicious a burner operating on standing allowances continues to process an account after it is frozen, because nothing stops it. Separately, `approve` never passes through `_update`, so a blacklisted account can still grant fresh allowances, and a global `tradingEnabled = false` halts transfers while leaving supply reduction available.

The practical weight is limited. Nothing is stolen and no party gains: burning destroys tokens rather than moving them, and it touches neither `freedAmount` nor protected-seller status, so it confers no cap relief or other advantage. A blacklisted holder burning its own balance therefore has no motive beyond denying a seizure. The defect is currently masked by the burn floor sitting above total supply, which makes every burn revert before these checks are reached, and becomes live the moment the floor is corrected.

AFFECTED CODE

- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/ZPZCoin.sol#L451>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/ZPZCoin.sol#L449>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/ZPZCoin.sol#L505>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/ZPZCoin.sol#L1094>

IMPACT

No funds are stolen and no party profits. The documented guarantee is nonetheless inaccurate: a blacklist deployed to mean that an account cannot move its tokens does not prevent those tokens being destroyed, by the account itself or by any platform burner holding a pre-freeze allowance, and an emergency `tradingEnabled = false` does not halt supply changes. The loss scenario is narrow it requires both an owner-appointed burner and an allowance the holder granted earlier but the destruction it causes is permanent.

REMIEDIATION

Move the blacklist and `tradingEnabled` checks above the transfer/burn branch split in `_update` so they apply to every value movement including burns, or add explicit guards to `burnForPlatform` and the public burn entry points. Remediate together with the burn-floor correction, never after it.

RETEST

Fixed by moving the `tradingEnabled` gate and the three blacklist checks out of the transfer branch, so a burn to the zero address is now covered by both. Verified fixed in `a908e33` .

Proposal finalisation depends on the proposer, and cancellation carries no time bound

DESCRIPTION

`executeProposal` is gated by `onlyAdminOrProposer` (ZPZCoinDAONew.sol), so a proposal that has passed can be finalised only by the proposer or the owner. A proposer who dislikes the result therefore needs to do nothing at all: declining to call `executeProposal` leaves the proposal unfinalised indefinitely, with no event and no record of the decision.

Against that baseline, the missing time bound on cancellation adds one thing. Every other lifecycle transition is time-guarded `vote` by the voting window (ZPZCoinDAONew.sol:384), `executeProposal` by `block.timestamp ≥ proposal.endTime` (ZPZCoinDAONew.sol) while `cancelProposal` (ZPZCoinDAONew.sol:514) checks only that the proposal exists, is not already cancelled and is not executed. A proposer can therefore let the vote run, observe a qualified passing result, and set `isCanceled` afterwards, which also forecloses the owner's path to finalisation rather than merely withholding their own. The NatSpec discloses the owner's guardian-cancel as a deliberate centralisation tradeoff (ZPZCoinDAONew.sol) but says nothing about either power the proposer holds.

AFFECTED CODE

- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/ZPZCoinDAONew.sol#L514>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/ZPZCoinDAONew.sol#L472>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/ZPZCoinDAONew.sol#L516>

IMPACT

Nothing on-chain depends on the outcome: a finalised proposal sets `proposal.executed` and emits `ProposalSignalFinalized`, that flag is read nowhere else, and the DAO holds no privileged role in the token or staking contracts. The practical effect of the missing bound is also narrower than it appears, since the proposer can already moot any result by declining to execute it. What cancellation adds is permanence it blocks the owner from finalising as well and it does so after voters have committed, with their recorded participation discarded and no re-open path.

REMEDIATION

Reject cancellation once `block.timestamp ≥ proposal.endTime`, mirroring the gate `executeProposal` already applies, and make `executeProposal` permissionless once the thresholds are met so finalisation does not depend on the proposer's cooperation. If the guardian power is to be retained past that point, split it onto a separate owner-only path so the proposer's cancel and the owner's guardian-cancel are distinct and separately auditable.

RETEST

Fixed by making `executeProposal` permissionless and adding `block.timestamp < proposal.endTime` to `cancelProposal`, so a proposer can neither withhold finalisation nor cancel a passed vote. Verified fixed in `a908e33`.

The participation quorum counts opposing votes, so one wallet clears it alone

DESCRIPTION

`executeProposal` measures turnout as `totalCast = proposal.votesFor + proposal.votesAgainst` (ZPZCoinDAONew.sol) and compares that sum against `minParticipationVotes` (ZPZCoinDAONew.sol), so votes cast *against* a proposal count toward the quorum that lets it pass. Nothing locks a voter to one direction: `support` is a per-call parameter applied at ZPZCoinDAONew.sol, while the budget is tracked as a single direction-agnostic total in `votesPerProposal` (ZPZCoinDAONew.sol) and surfaced as one shared remaining balance by `getVoteStatus`.

A single wallet therefore manufactures its own quorum. With the deployment parameters the repository ships `minParticipationVotes = 100` (`script/deploy.DA0.js`, `script/deploy.sepolia.js`) and `votingThreshold = 1000e18` a wallet holding 10,000e18 of stake receives `sqrt(1e22) / TOKEN_DECIMAL_SCALE = 100` credits. Casting 99 FOR and then 1 AGAINST makes `totalCast = 100`, which meets the minimum, while `votesFor * 2 = 198 > 100` satisfies the majority test at ZPZCoinDAONew.sol; 99 FOR alone would miss the quorum by one. Finalisation needs a second party, since `vote` rejects the proposer (ZPZCoinDAONew.sol) and `executeProposal` is gated on proposer-or-owner, but no additional stake is required. The same arithmetic lets a hostile holder lift an ally's under-subscribed proposal over the bar: 70 genuine FOR votes fall below the quorum of 100, and 39 AGAINST votes carry the total to 109, after which `140 > 109` passes.

The denominator a proportional quorum would need is already maintained and discarded

`proposalTotalVotePower` is incremented by each voter's snapshotted credits (ZPZCoinDAONew.sol) and read nowhere.

AFFECTED CODE

- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/ZPZCoinDAONew.sol#L477>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/ZPZCoinDAONew.sol#L480>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/ZPZCoinDAONew.sol#L429>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/ZPZCoinDAONew.sol#L410>

IMPACT

Nothing on-chain depends on the outcome: the DAO contains no low-level call, `executeProposal` sets `proposal.executed` and emits `ProposalSignalFinalized`, that flag is read nowhere else, and the DAO holds no privileged role in the token or staking contracts. Within that ceiling the defect is real and needs no privilege. The participation floor provides no assurance of engagement a single wallet meeting the voting threshold satisfies it by voting on both sides of its own proposal, at a cost of 10,000e18 of stake under the shipped configuration and, counter-intuitively, votes cast against a proposal help it pass, so an opponent

can lift an under-subscribed proposal over the bar that was about to reject it. The harm is to the integrity of the published signal, which off-chain processes may treat as genuine community engagement.

REMIEDIATION

Count only `votesFor` toward `minParticipationVotes`, or express the quorum as a proportion of `proposalTotalVotePower` so turnout is measured against eligible power rather than an absolute that either side can inflate. Separately, record a voter's chosen direction on first vote and reject later votes in the opposite direction, so one wallet cannot occupy both sides of the same proposal.

RETEST

Fixed by measuring the participation floor against `votesFor` alone instead of `votesFor + votesAgainst`, so votes against a proposal no longer count toward the quorum that passes it. Verified fixed in `a908e33`.

The early-unstake penalty cited as the flash-stake mitigation does not exist

DESCRIPTION

The published audit response accepts two DAO risks on the strength of a five percent early-unstake penalty: against flash-loan voting-power inflation it states that a penalty applies if a user unstakes before maturity and that this makes flash staking impractical, and against multi-address Sybil voting it repeats that an attacker would incur the penalty on each exit.

No such penalty exists in `NewSTaking.sol`. The code states the opposite in four places: normal stakes have no early-unstake penalty under the unbonding model (`NewSTaking.sol`), the `penalty` return value is documented as always zero and retained only for interface compatibility (`NewSTaking.sol`), whitelisted stakes have no early penalty (`NewSTaking.sol`), and `withdrawPrincipal` is documented as withdrawing without a penalty (`NewSTaking.sol:781`). `previewUnstake` `delete`s the field it returns (`NewSTaking.sol:1731`), and every exit path returns principal in full. The only real frictions are the fourteen-day `UNBONDING_COOLDOWN` and forfeiture of the whitelisted maturity bonus neither named in the response, and neither of which costs a voter anything if they never intended to hold to maturity.

AFFECTED CODE

- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/NewSTaking.sol#L1714>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/NewSTaking.sol#L1731>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/NewSTaking.sol#L637>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/NewSTaking.sol#L781>

IMPACT

Two accepted governance risks rest on a control that is not present, so the documented security posture overstates the protocol's actual defences. The underlying snapshot-voting exposure remains unmitigated by anything the response names. Nothing on-chain is affected; the defect is in the risk record the client and its reviewers rely on.

REMEDIATION

Update the audit response to cite the mechanisms that actually exist the fourteen-day unbonding cooldown and whitelisted bonus forfeiture and re-run the flash-stake and Sybil acceptances against those frictions, which impose no proportional cost on a voter who exits immediately. If the acceptance genuinely depended on a five percent penalty, implement it rather than restating it.

RETEST

Fixed by correcting the risk record rather than the code: `Updated_Audit_Report` lines 23 and 45 now state there is no principal penalty, name the fourteen-day unbonding cooldown and bonus forfeiture as the real frictions, and withdraw the claim that either prevents borrowed-capital voting. The Sybil exposure is retained as an explicitly accepted choice. Verified fixed in `a908e33` .

The daily outflow basis is re-snapshotted every day, contradicting its documentation

DESCRIPTION

`dailySellLimitBasis` is documented as "opening balance used as a wallet's immutable launch-day outflow basis" (ZPZCoin.sol). The enforcement path does not treat it that way: on every day rollover `_enforceDailySellLimit` rewrites it from the wallet's current balance (ZPZCoin.sol), so the basis is neither immutable nor anchored to the launch day.

A wallet that accumulates tokens during the sixty-day window therefore receives a larger one percent allowance each day, computed from a balance that may be many times its launch-day holding. The control is looser than its own documentation describes, and any risk assessment reading the NatSpec would conclude that a holder's launch-window outflow is bounded by their balance at launch when it is in fact bounded by their balance each morning.

AFFECTED CODE

- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/ZPZCoin.sol#L166>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/ZPZCoin.sol#L1415>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/ZPZCoin.sol#L1411>

IMPACT

Launch-window outflow is bounded by each day's opening balance rather than the launch-day balance, so a wallet acquiring tokens during the window gains a growing allowance. The divergence misleads anyone relying on the stated semantics.

REMEDIATION

Either set the basis once when the window opens and leave it fixed, matching the documentation, or correct the NatSpec to describe a daily-rebasing allowance.

RETEST

Fixed by correcting the documentation rather than the behaviour: the NatSpec now reads "Pre-outflow balance snapshot for the current launch-relative day" instead of describing an immutable launch-day basis. Verified fixed in `a908e33`.

previewUnstake contradicts every exit path it is meant to preview

DESCRIPTION

`previewUnstake` exists so an interface can show what an exit will pay, and it diverges from the transaction on all three of its branches. For a regular stake it returns `toUser = principal` (`NewSTaking.sol`) while `unstake` routes to `_requestRegularUnstake`, which transfers nothing and queues a fourteen-day cooldown. Once `requestUnstake` has cleared `s.active` (`NewSTaking.sol`) the view's `if (!s.active)` return `(0, 0, 0)` guard (`NewSTaking.sol`) reports that the staker holds nothing for the whole unbonding window and permanently afterwards, even though `pendingUnbondingPrincipal` holds the full amount and `pendingReward` still reports the reward side of the same stake (`NewSTaking.sol`). For a whitelisted pre-maturity exit it returns early with `reward = 0` (`NewSTaking.sol`). That is defensible on the documented contract, since the NatSpec scopes `reward` to the maturity bonus "excluding separately claimable APY" and an early exit forfeits that bonus but it understates what the transaction actually delivers, because `_earlyExitWhitelisted` settles and pays the accrued APY in the same call rather than leaving it to be claimed separately. For a ten-million-token stake held one full band that is 1,700,000e18 paid by the exit and reported by the preview as nothing, so the qualifier the NatSpec relies on does not describe this path.

The `penalty` return value is `delete`d unconditionally (`NewSTaking.sol`) and is never non-zero on any path.

AFFECTED CODE

- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/NewSTaking.sol#L1724>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/NewSTaking.sol#L1733>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/NewSTaking.sol#L1767>
- <https://github.com/EZPZTRADING/final-audit/blob/13ad66268cd64e8aeb04df7cb6271f91ff73199d/contracts/scan/NewSTaking.sol#L2027>

IMPACT

Any frontend built on the contract shows users a payout they will not receive, hides one they will, and misstates a mid-unbonding position by the full principal for fourteen days and beyond. Funds remain claimable; the harm is user confusion and support burden on every unstake.

REMEDIATION

Replace the `!s.active` early return with a `pendingUnbondingPrincipal` lookup so the view reflects what `withdrawUnbondedPrincipal` will pay, settle-preview the APY for whitelisted early exits, and either populate `penalty` or remove it from the signature.

RETEST

Fixed by returning `pendingUnbondingPrincipal` for a ready unbonding instead of zero, and by including the funded APY a whitelisted early exit pays. Verified fixed in `a908e33`.

DISCLOSURE

The Reports provided by CredShields are not an endorsement or condemnation of any specific project or issue and do not guarantee the security of any specific project. The contents of this report are not intended to be used to make decisions about buying or selling tokens, products, services, or any other assets and should not be interpreted as such.

Emerging technologies such as Smart Contracts and Solidity carry a high level of technical risk and uncertainty. CredShields does not provide any warranty or representation about the quality of code, the business model or the proprietors of any such business model, or the legal compliance of any business. The report is not intended to be used as investment advice and should not be relied upon as such.

CredShields Audit team is not responsible for any decisions or actions taken by any third party based on the report.

Reviewers & Signatories

Shreyas Koli

Security Auditor II

Prasad Kuri

Security Auditor



Shashank

Co-founder & CEO · CredShields

Security analyst and penetration tester with **12 years of experience** in pentesting and **5 years** in managing bug bounty programs. Listed in **Google, Facebook, Apple**, and 30+ websites' hall of fame for reporting valid security issues.

CVE-2017-7063

CVE-2017-7062

CVE-2017-2458



Aditya Dixit

Audit & Research Lead · CredShields

Senior security researcher and penetration tester with **10+ years** in cybersecurity. Led pentest teams at **Cobalt and HackerOne**; previously at Binary.com. Executed **500+ pentests** across banking, fintech, payments, healthcare, telecom, and media, covering web, mobile (iOS/Android), network, cloud, code review, and smart contracts.

OSCP

CREST CPSA

AWS Security Specialty

AWS Cloud Practitioner

Qualys VMDR



EZPZ-2026-SC-01 · Confidential · © 2026 CredShields Technologies Pte. Ltd.

credshields.com · info@credshields.com

