PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For YoProtocol

21 November 2025

paladinsec.co

info@paladinsec.co

Table of Contents

Disclaimer

Paladin Blockchain Security ("Paladin") has conducted an independent audit to verify the integrity of and highlight any vulnerabilities or errors, intentional or unintentional, that may be present in the codes that were provided for the scope of this audit. This audit report does not constitute agreement, acceptance or advocation for the Project that was audited, and users relying on this audit report should not consider this as having any merit for financial advice in any shape, form or nature. The contracts audited do not account for any economic developments that may be pursued by the Project in question, and that the veracity of the findings thus presented in this report relate solely to the proficiency, competence, aptitude and discretion of our independent auditors, who make no guarantees nor assurance that the contracts are completely free of exploits, bugs, vulnerabilities or deprecation of technologies. Further, this audit report shall not be disclosed nor transmitted to any persons or parties on any objective, goal or justification without due written assent, acquiescence or approval by Paladin.

All information provided in this report does not constitute financial or investment advice, nor should it be used to signal that any persons reading this report should invest their funds without sufficient individual due diligence regardless of the findings presented in this report. Information is provided 'as is', and Paladin is under no covenant to the completeness, accuracy or solidity of the contracts audited. In no event will Paladin or its partners, employees, agents or parties related to the provision of this audit report be liable to any parties for, or lack thereof, decisions and/or actions with regards to the information provided in this audit report.

Cryptocurrencies and any technologies by extension directly or indirectly related to cryptocurrencies are highly volatile and speculative by nature. All reasonable due diligence and safeguards may yet be insufficient, and users should exercise considerable caution when participating in any shape or form in this nascent industry.

The audit report has made all reasonable attempts to provide clear and articulate recommendations to the Project team with respect to the rectification, amendment and/or revision of any highlighted issues, vulnerabilities or exploits within the contracts provided. It is the sole responsibility of the Project team to sufficiently test and perform checks, ensuring that the contracts are functioning as intended, specifically that the functions therein contained within said contracts have the desired intended effects, functionalities and outcomes of the Project team.

Paladin retains the right to re-use any and all knowledge and expertise gained during the audit process, including, but not limited to, vulnerabilities, bugs, or new attack vectors. Paladin is therefore allowed and expected to use this knowledge in subsequent audits and to inform any third party, who may or may not be our past or current clients, whose projects have similar vulnerabilities. Paladin is furthermore allowed to claim bug bounties from third-parties while doing so.

1 Overview

This report has been prepared for YoProtocol on the Ethereum network. Paladin provides a user-centred examination of the smart contracts to look for vulnerabilities, logic errors or other issues from both an internal and external perspective.

1.1 Summary

Project Name	YoProtocol
URL	https://www.yo.xyz/
Platform	Ethereum
Language	Solidity
Preliminary Contracts	https://github.com/yoprotocol/core/tree/876663f7a5b73d0fb343b841f0ed45cbc988aff5/src
Resolution #1	https://github.com/yoprotocol/core/commit/266360516a3d094433ee3eff5bbe8f037ecd0eff

1.2 Contracts Assessed

Name	Contract	Live Code Match
`YoEscrow`		
`YoGateway`		
`YoVault`		
`YoRegistry`		
`YoSecondaryVault`		

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
Governance	2	-	-	2
High	0	-	-	-
Medium	0	-	-	-
Low	7	1	-	6
Informational	10	-	-	10
Total	**19**	**1**	**0**	**18**

Classification of Issues

Severity	Description
Governance	Issues under this category are where the governance or owners of the protocol have certain privileges that users need to be aware of, some of which can result in the loss of user funds if the governance's private keys are lost or if they turn malicious, for example.
High	Exploits, vulnerabilities or errors that will certainly or probabilistically lead towards loss of funds, control, or impairment of the contract and its functions. Issues under this classification are recommended to be fixed with utmost urgency.
Medium	Bugs or issues with that may be subject to exploit, though their impact is somewhat limited. Issues under this classification are recommended to be fixed as soon as possible.
Low	Effects are minimal in isolation and do not pose a significant danger to the project or its users. Issues under this classification are recommended to be fixed nonetheless.
Informational	Consistency, syntax or style best practices. Generally pose a negligible level of risk, if any.

1.3.1 YoEscrow

ID	Severity	Summary	Status
01	INFO	ETH dust is not withdrawable	ACKNOWLEDGED

1.3.2 YoGateway

ID	Severity	Summary	Status
02	LOW	Gateway `minAssetsOut` check compares against gross (pre-fee) instead of net assets	ACKNOWLEDGED

1.3.3 YoVault

ID	Severity	Summary	Status
03	GOV	Operator can fulfill requests with arbitrary exchange rate	ACKNOWLEDGED
04	GOV	Queued redeems use current fee configuration instead of fee at request time	ACKNOWLEDGED
05	LOW	Zero address receiver can lead to irrecoverable pending requests	✓ RESOLVED
06	LOW	No way to enforce "instant-only" redemption (frontrunnable downgrade to queued redeem)	ACKNOWLEDGED
07	LOW	First depositor inflation	ACKNOWLEDGED
08	LOW	If fee recipient is equal to zero, users are still charged	ACKNOWLEDGED
09	LOW	The maximal amount is actually not allowed due to an off by one bug	ACKNOWLEDGED
10	INFO	Missing dedicated fee events makes fee accounting unclear	ACKNOWLEDGED
11	INFO	`maxWithdraw` advertises liquidity while `withdraw()` is permanently disabled (EIP-4626 mismatch)	ACKNOWLEDGED
12	INFO	No slippage-checked mint in `YoGateway`	ACKNOWLEDGED
13	INFO	Documentation mentions 24-hour redemption limit, but code has no timeout or expiry	ACKNOWLEDGED
14	INFO	Documentation mismatch: "private functions" section contains internal	ACKNOWLEDGED
15	INFO	Batch manage lacks array length checks	ACKNOWLEDGED
16	INFO	cancel returns shares to receiver instead of original owner	ACKNOWLEDGED
17	INFO	Division by zero when updating price with zero total supply	ACKNOWLEDGED

1.3.4 YoRegistry

No issues found.

1.3.5 YoSecondaryVault

ID	Severity	Summary	Status
18	LOW	`YoSecondaryVault` mis-scales `lastPricePerShare` for non-18 decimal assets	ACKNOWLEDGED
19	INFO	`initializeV2` is public, thus anyone can set the initial PPS or during upgrade	ACKNOWLEDGED

2 Findings

2.1 YoEscrow

YoEscrow is a simple helper contract that holds ERC20 tokens on behalf of a single vault. It is deployed with a fixed `VAULT` address, and only that vault can pull tokens out.

2.1.1 Privileged Functions

- `withdraw`

2.1.2 Issues & Recommendations

Issue #01	ETH dust is not withdrawable
Severity	INFORMATIONAL
Description	`withdraw()` allows to retrieve any asset balance as long as it is an ERC20 and transfer it to the calling vault, which can be done using the `manage()` call. However, the function does not comply with potential ETH balances, leaving it on the escrow. On the other hand, since there is no native support of ETH deposits, it is unlikely to become a problem.
Recommendation	Consider adding the ability to sweep ETH (or any other native funds) as well along the ERC20s.
Resolution	ACKNOWLEDGED The team commented that the escrow is used for withdrawing ERC20 tokens and ETH is never sent to it intentionally.

2.2 YoGateway

`YoGateway` is a simple entry point that routes deposits and redemptions to approved YO vaults. It pulls assets or shares from the user, calls the target vault's ERC4626 methods, and emits partner-attribution events.

Instant redemptions pay out immediately; queued ones are handled later by the vault. The gateway itself holds no funds or logic beyond safe transfers and vault allow-listing.

2.2.1 Issues & Recommendations

Issue #02	**Gateway `minAssetsOut` check compares against gross (pre-fee) instead of net assets**
Severity	LOW SEVERITY

Description	In an instant redeem, `YoVault.requestRedeem()` returns `assetsWithFee` (the gross amount before exit fee is carved out in `_withdraw`). `YoGateway.redeem()` treats this return as the "assets out" and validates:
	`bool instant = assetsOrRequestId > 0;`
	`if (instant && assetsOrRequestId < minAssetsOut) revert;`
	
	But the user actually receives `net = assetsWithFee - fee`.
	
	Result: if exit fee > 0, the gateway can pass the `minAssetsOut` check even when the actual assets sent to the user are below `minAssetsOut`. Users are under-protected and can be paid less than they explicitly required.

Recommendation	When `instant == true`, validate against the net amount:
	
	Either make `YoVault.requestRedeem()` return the net assets on instant path, or keep the current return, but in YoGateway recompute the net:
	`// query fee from vault or replicate fee formula`
	`uint256 fee = feeOnTotal(assetsOrRequestId, feeOnWithdraw);`
	`uint256 net = assetsOrRequestId - fee;`
	`if (net < minAssetsOut) revert;`

Resolution	ACKNOWLEDGED The `minAssetOut` is passed as argument, so the fee can be calculated off-chain.

2.3 YoVault

`YoVault` is an ERC4626 vault with fees and an asynchronous redeem system. Deposits mint shares net of entry fees, and redemptions either execute instantly or get queued until the operator fulfils them.

Total assets count both the on-chain balance and oracle-reported balances from other chains or strategies, and large price moves automatically trigger a pause. Operators can manage external strategy calls through an authenticated manage function.

The following duplicate issues from previous security reports were found during this audit:

- Operator can change share-to-asset exchange rate (Spearbit)
- Inflation griefing attacks are possible (Spearbit)
- Incorrectly handling the case where fee recipient is not set (Hunter Security)
- Tokens may not be going to respective wallets (Spearbit)

2.3.1 Privileged Functions

- `manage`
- `pause`
- `unpause`
- `fulfillRedeem`
- `cancelRedeem`
- `onUnderlyingBalanceUpdate`
- `updateMaxPercentageChange`
- `updateWithdrawFee`
- `requiresAuth`
- `updateDepositFee`

2.3.2 Issues & Recommendations

Issue #03	Operator can fulfill requests with arbitrary exchange rate
Severity	GOVERNANCE

Location	`function fulfillRedeem(address receiver, uint256 shares, uint256 assetsWithFee) external requiresAuth {`
	`    // Only bounds each independently; no ratio check`
	`    require(pending.shares != 0 && shares <= pending.shares, Errors.InvalidSharesAmount());`
	`    require(pending.assets != 0 && assetsWithFee <= pending.assets, Errors.InvalidAssetsAmount());`
	`    [...]`
	`    _withdraw(address(this), receiver, address(this), assetsWithFee, shares); // ratio unchecked`
	`}`

Description	The arbitrary exchange rate in `fulfillRedeem`, which is callable only by the vault operator, can be used — e.g. 1 share and full amount of funds which would lead to an unfair redemption. While this is an admin only function, hence likelihood of exploit is low, it could become a single point of failure or possible centralization issue.
Recommendation	Enforce the exchange rate on `withdraw` instead of allowing it to be arbitrary. Alternatively, allow users to specify a `minAssetsOut` value (excluding fee) enforced on fulfillment.
Resolution	ACKNOWLEDGED The team commented that this is a duplicate of another report from Spearbit. The operator has no economic incentive to fulfill the redeem order with an incorrect amount.

Issue #04	Queued redeems use current fee configuration instead of fee at request time
Severity	GOVERNANCE
Description	`requestRedeem` records `pending.assets` as the gross amount from `previewRedeem(shares)` at request time. When the operator later calls `fulfillRedeem`, `_withdraw` recalculates the exit fee using the current `feeOnWithdraw` and `feeRecipient` and then sends the net amount. As users cannot cancel their own pending requests, they are fully exposed to whatever fee and `feeRecipient` changes are made after they enter the queue. This allows an authorized operator to: - Increase `feeOnWithdraw` (up to 10%) after users are queued and capture a larger fee on fulfillment. - Toggle `feeRecipient` between a treasury address or `address(0)` to change where the fee value goes, even after users are locked in. - Partially fulfill one request across multiple fee configurations, producing different effective fees for the same user.
Recommendation	Cache fee configuration at the moment of `requestRedeem` and compute the net payout using the snapshotted values, not current ones. Alternatively, let users cancel their pending request.
Resolution	ACKNOWLEDGED The team commented that this is by design. The user receives assets based on the state at the moment of request and not fulfilment.

Issue #05	Zero address receiver can lead to irrecoverable pending requests
Severity	LOW SEVERITY
Description	`requestRedeem` is missing `receiver != address(0)` guard. Zero-address receiver is accepted and used for both asset and share transfers, while standard ERC-20s revert on transfers to 0x0, making pending requests unfulfillable and uncancellable. While the impact is limited and needs the creation of an invalid recipient, this seems to be an undesired smart contract state.
Recommendation	Consider adding `require(receiver != address(0)`.
Resolution	RESOLVED

Issue #06	No way to enforce "instant-only" redemption (frontrunnable downgrade to queued `redeem`)
Severity	LOW SEVERITY
Description	`requestRedeem(shares, receiver, owner)` attempts an instant redeem only if the vault's available balance is high enough. If not, it silently falls back to a queued redeem and returns `REQUEST_ID = 0`. There is no user-side way to demand "instant or revert." This allows a simple frontrun: if the user expects an instant redeem, another borrower/redeemer can drain a small amount of liquidity just before the call. The user is then pushed into a queued redeem without notice, and their shares become locked until an operator fulfills the request later. The cost is opportunity loss as the user's ERC4626 shares can no longer be reused elsewhere.
Recommendation	Add an optional bool requireInstant argument to `requestRedeem` and revert if the vault cannot satisfy the redemption immediately when the flag is set. This makes user intent explicit and prevents unexpected queued redemptions caused by miner frontrunning.
Resolution	ACKNOWLEDGED The team commented that throwing when instant is not available is against their system design.

Issue #07	First depositor inflation
Severity	LOW SEVERITY
Description	`YoVault` inherits `ERC4626Upgradeable` but does not protect the first depositor against share-price inflation. With the vault empty, the first user can mint shares at a tiny exchange rate, then an attacker can donate a large amount of the asset before the next deposit executes. This spikes `totalAssets` while `totalSupply` stays almost unchanged, pushing `pricePerShare` to an extreme value. The next user's `previewDeposit` returns ≈ 0 shares, so their deposit executes but mints zero shares - effectively burning their deposit.
Recommendation	Seed the vault at deployment with a meaningful initial deposit.
Resolution	ACKNOWLEDGED The team commented that this is a duplicate issue from another audit, and that they are also far beyond the first depositor and defend against it during deployment.

Issue #08	If fee recipient is equal to zero, users are still charged
Severity	LOW SEVERITY
Description	YoVault's documentation says that if `feeRecipient` is zero, no fees are charged. The code does not follow this behavior. During both deposit and redeem flows, the vault always applies fee math (`_feeOnTotal`). When `feeRecipient == address(0)`, the fee is still deducted from the user (fewer assets out, fewer shares minted), but the fee value just stays inside the vault. This silently taxes users and alters share price, contradicting the intended behavior and breaking user expectations. In other words, `feeRecipient == 0` does not disable fees, it only disables the fee payout. Users still pay the fees.
Recommendation	If `feeRecipient == address(0)`, skip the fee entirely: For deposit, mint shares against the full assets without reducing via `_feeOnTotal`. For `redeem`, send full `assetsWithFee` to the receiver with no deduction. Alternatively, explicitly document that `feeRecipient == 0` means "fees are retained by the vault" instead of "no fee charged" and update previews to reflect this.
Resolution	ACKNOWLEDGED The team commented that this is a duplicate issue from another audit.

Issue #09	The maximal amount is actually not allowed due to an off by one bug
Severity	INFORMATIONAL
Location	`require(newFee < MAX_FEE, Errors.InvalidFee());` `require(newMaxPercentageChange < MAX_PERCENTAGE_THRESHOLD, Errors.InvalidMaxPercentage());`
Description	In YoVault maximum thresholds, the constants define a "maximum," value, but code enforces "strictly less than." Due to this, the maximal value is actually not allowed, despite the comments say "The maximum fee that can be set (...)"
Recommendation	Use <= for comparison or update the comments to reflect the desired state.
Resolution	ACKNOWLEDGED

Issue #10	Missing dedicated fee events makes fee accounting unclear
Severity	INFORMATIONAL
Description	When deposits or redemptions apply fees, the vault either transfers those tokens to `feeRecipient` or retains them inside the contract. However, the contract does not emit any dedicated events for fees. This reduces transparency for users, partners and auditors who rely on clean on-chain logging.
Recommendation	Emit a dedicated event when fees are charged: - `FeePaid(recipient, amount)` when fees are transferred out - `FeeRetained(amount)` when `feeRecipient` is zero and the fee remains in the vault
Resolution	ACKNOWLEDGED

Issue #11	**`maxWithdraw` advertises liquidity while `withdraw()` is permanently disabled (EIP-4626 mismatch)**
Severity	INFORMATIONAL
Description	`YoVault.withdraw()` always reverts, but `maxWithdraw(owner)` returns a positive value when not paused (inherited from OZ). Per EIP-4626, `maxWithdraw` MUST return 0 "whenever withdrawals are entirely disabled" (https://eips.ethereum.org/EIPS/eip-4626). Here, withdrawals are permanently disabled by design, so returning a non-zero `maxWithdraw` doesn't follow EIP4626 spec.
Recommendation	Make `maxWithdraw` always return 0.
Resolution	ACKNOWLEDGED The team commented that the protocol is not fully ERC4626 compatible due to async redemption.

Issue #12	**No slippage-checked mint in `YoGateway`**
Severity	INFORMATIONAL
Description	The gateway exposes a slippage-checked deposit, but there is no equivalent safe path for mint. If a user or integrator wants to mint a fixed amount of shares, they must call `YoVault.mint()` directly and implement their own min-asset-in check. This breaks consistency with how deposit is handled: the gateway is the canonical entrypoint, but does not provide symmetric slippage protection for both flows.
Recommendation	Add a `mint` function to `YoGateway` with a `maxAssetsIn` or similar parameter, reverting if the required asset transfer exceeds user expectation. This keeps the interface symmetric.
Resolution	ACKNOWLEDGED The team commented that they recommend using deposit, which is why they have not done this for mint.

Issue #13	Documentation mentions 24-hour redemption limit, but code has no timeout or expiry
Severity	INFORMATIONAL
Description	The documentation states: "If the target vault does not have sufficient liquidity, the redemption will remain pending for up to 24 hours. Once filled, the vault will send assets directly to the receiver." (https://docs.yo.xyz/technical-guides/yogateway-integration-guide) However, the contracts never enforce a 24-hour limit. `requestRedeem` stores no timestamps and `fulfillRedeem`/`cancelRedeem` have no deadline logic. Pending redemptions can stay open indefinitely, completely controlled by the operator.
Recommendation	Update docs to reflect actual behavior.
Resolution	ACKNOWLEDGED The team commented that their automated system makes sure that all redeem requests are executed in 24 hours.

Issue #14	Documentation mismatch: "private functions" section contains internal
Severity	INFORMATIONAL
Description	In YoVault.sol, the code section prepended with comment "`//============================== PRIVATE FUNCTIONS ==============================`" still contains internal functions. This decreases maintainability.
Recommendation	Move internal functions out of this section.
Resolution	ACKNOWLEDGED

Issue #15	Batch manage lacks array length checks
Severity	INFORMATIONAL

Location	`function manage(address[] calldata targets, bytes[] calldata data, uint256[] calldata values) external ...`
	`{`
	`  uint256 targetsLength = targets.length; // no data/values length checks`
	`  for (uint256 i; i < targetsLength; ++i) {`
	`  [...]`
	`  results[i] = targets[i].functionCallWithValue(data[i], values[i]);`
	`  }`
	`}`

Description	The `manage` function takes multiple arrays as an argument with an assumption that they are of equal length, iterating over one but processing all of them. If the arrays are not equal, it will lead to a late revert, unnecessarily wasting gas.
Recommendation	Require `targets.length == data.length && data.length == values.length`.
Resolution	ACKNOWLEDGED

Issue #16	`cancel` returns shares to receiver instead of original owner
Severity	INFORMATIONAL
Description	On `cancel`, shares are transferred back to the receiver rather than the original owner (contradicting the in-line comment "transfer the shares back to the owner"). This can misroute shares when owner != receiver
Recommendation	Store and return shares to the recorded owner on cancel, or explicitly document/send to owner.
Resolution	ACKNOWLEDGED The team commented that this is a duplicate issue from another audit.

Issue #17	Division by zero when updating price with zero total supply
Severity	INFORMATIONAL
Description	The first oracle update in `onUnderlyingBalanceUpdate` (or any update with no shares minted) attempts to calculate the `newPricePerShare` against zero supply, which leads to an unexpected revert. While there seems to be no direct security impact, an unexpected revert seems to be an unexpected state.
Recommendation	Require the total supply to be greater than zero before updating.
Resolution	ACKNOWLEDGED The team commented that this is a duplicate issue from another audit and that their total supply is far beyond 0.

2.4 YoRegistry

`YoRegistry` is the aggregator of all existing vaults. By adding a vault to a registry it is exposed to the Gateway entrypoint.

2.4.1 Privileged Functions

- `addYoVault`
- `removeYoVault`

2.4.2 Issues & Recommendations

No issues found.

2.5 YoSecondaryVault

`YoSecondaryVault` is a `YoVault` variant that ignores on-chain balances and instead prices deposits and redemptions using an oracle-provided price per share.

It keeps the same fee logic and async redemption queue but uses `lastPricePerShare` for all share/asset conversions. This isolates share pricing from on-chain liquidity changes like donations.

The contract is activated via `initializeV2`, which sets the initial price per share.

The following duplicate issues from previous security reports were found during this audit:

- Non-critical issues and suggestions - Anyone can call `initializeV2` eventually frontrunning the call upon deployment. (Aether Labs)

2.5.1 Privileged Functions

- `onSharePriceUpdate(newSharePrice) - set the oracle price-per-share (requiresAuth)`

2.5.2 Issues & Recommendations

Issue #18	**YoSecondaryVault mis-scales lastPricePerShare for non-18 decimal assets**
Severity	LOW SEVERITY

Description	YoVault v1 defines lastPricePerShare (PPS) on a 1e18 scale:
	`lastPricePerShare = totalAssets * 1e18 / totalSupply;`
	
	YoSecondaryVault (v2) changes how PPS is used and assumes it is scaled by 10 ** decimals():
	`// shares = assets * 10**decimals / lastPricePerShare`
	`_convertToShares: assets.mulDiv(10 ** decimals(), lastPricePerShare, rounding);`
	
	`// assets = shares * lastPricePerShare / 10**decimals`
	`_convertToAssets: shares.mulDiv(lastPricePerShare, 10 ** decimals(), rounding);`
	
	If lastPricePerShare continues to be pushed in the old 1e18 scale (as produced in v1), then for any asset with decimals != 18 the conversions are incorrect by a fixed factor: error factor = 10^(18 - decimals). e.g. USDC (6 dec): 1e12 error.
	
	So, after upgrading / migrating and setting initializeV2(_lastPricePerShare) or calling onSharePriceUpdate(newSharePrice) with a 1e18-scaled value, convertToAssets will vastly overstate assets per share for non-18 dec tokens.
	
	Existing share holders can redeem for orders of magnitude more assets than the vault's real NAV per share, thus draining liquidity / forcing insolvency. And vice versa, deposits would mint too few shares for the assets provided.

Recommendation	Change v2 conversions to use 1e18:
	`_convertToShares: return assets.mulDiv(1e18, lastPricePerShare, rounding);`
	`_convertToAssets: return shares.mulDiv(lastPricePerShare, 1e18, rounding);`
	
	Also add a sanity check in `initializeV2` to validate `lastPricePerShare ≈ totalAssets * 1e18 / totalSupply` (when `supply > 0`).

Resolution	ACKNOWLEDGED The team commented that there is a safeguard in `onSharePriceUpdate` that would prevent this from happening.

Issue #19	**`initializeV2` is public, thus anyone can set the initial PPS or during upgrade**
Severity	INFORMATIONAL
Description	`YoSecondaryVault.initializeV2(uint256 _lastPricePerShare)` is `public reinitializer(2)` with no auth guard. If the proxy upgrade to v2 is not done atomically via `initializeV2`, anyone can front-run and set an arbitrary `lastPricePerShare` and zero out `aggregatedUnderlyingBalances`. A malicious PPS lets an attacker mint underpriced shares (if PPS is set too low), overpay/underpay on redemptions or brick the vault by setting PPS to 0 (division by zero on conversions). This is a one-transaction takeover window at upgrade time.
Recommendation	Make `initializeV2` protected with `requiresAuth` or enforce that upgrades perform `upgradeAndCall` to initialize atomically in a single transaction. Also consider adding require `_lastPricePerShare > 0` and emitting a `V2Initialized(lastPricePerShare, migrator)` event.
Resolution	ACKNOWLEDGED The team commented that this is a duplicate issue from another audit and that they use `upgradeAndCall` thus this can never happen.


PALADIN
BLOCKCHAIN SECURITY