

January 30th 2026

Axis

Comprehensive Security Assessment

Table of Contents

Disclaimer

1. Introduction

- 1.1. Executive Summary
- 1.2. Project Timeline
- 1.3. Scope
- 1.4. Overview of Findings

2. Findings

- 2.1. [M-1] Cooldown Can Be Shortened for Existing Locked Assets
- 2.2. [L-1] Blacklist Can Be Frontrun to Transfer Out Before Restriction
- 2.3. [L-2] setOperator() with Same Address Downgrades Delegation
- 2.4. [L-3] Minting-Contract Rotation Breaks Delegation Until Re-Set
- 2.5. [I-1] Remainder Dust Always Routes to Last Custodian
- 2.6. [I-2] safeIncreaseAllowance Can Overflow on Re-Approval
- 2.7. [I-3] Virtual-Share Offset Outdated Comment
- 2.8. [I-4] AxisUSDRateProvider Rate Is Manipulable via Direct Transfers
- 2.9. [I-5] Cannot Revoke Allowances from Current Minting Contract

Disclaimer

This security assessment represents a time-boxed security review using tooling and manual review methodologies. Our findings reflect our comprehensive evaluation of the materials provided in-scope and are specific to the commit hash referenced in this report.

The scope of this security assessment is strictly limited to the code explicitly specified in the report. External dependencies, integrated third-party services, libraries, and any other code components not explicitly listed in the scope have not been reviewed and are excluded from this assessment.

Any modifications to the reviewed codebase, including but not limited to smart contract upgrades, protocol changes, or external dependency updates will require a new security assessment, as they may introduce considerations not covered in the current review.

In no event shall Plainshift's aggregate liability for all claims, whether in contract or any other theory of liability, exceed the Services Fee paid for this assessment. The client agrees to hold Plainshift harmless against any and all claims for loss, liability, damages, judgments and/or civil charges arising out of exploitation of security vulnerabilities in the contracts reviewed.

By accepting this report, you acknowledge that deployment and implementation decisions rest solely with the client. Any reliance upon the information in this report is at your own discretion and risk. This disclaimer is governed by and construed in accordance with the laws specified in the engagement agreement between Plainshift and the client.

1. Introduction

Plainshift is a full-stack security firm built on the “shift left” security philosophy. We often work with teams early in the product development process to bring security to a greater organizational range than just smart contracts. From the web app, to fuzzing/formal verification, to a team’s operational security, full-stack security can only be achieved by first understanding there is no “scope” to fully protect the users that trust you.

We’re here to meaningfully revolutionize how teams approach security and guide them towards a holistic approach rather than the single sided approach so prevalent today.

Learn more about us at <https://plainshift.io>.

1.1. Executive Summary

Plainshift was tasked with reviewing Axis’s synthetic dollar and staking protocol between January 26th and January 30th, 2026. The codebase is forked from Ethena. The core contracts form a cohesive system: `AxisUSD` and `AxisUSDMinting` handle minting and collateral routing, `StakedAxisUSD` provides the ERC-4626 staking vault with vesting logic, `StakedAxisUSDV2` adds a cooldown-based withdrawal flow backed by `AxisUSDSilo`, `AxisUSDRateProvider` exposes a rate view for integrations, `SingleAdminAccessControl` centralizes admin permissions, and `StakingRewardsDistributor` manages reward distribution and operator delegation. Overall the codebase is very solid. We ultimately confirmed 1 medium, 3 low, and 5 informational findings. The only medium-severity finding is a fairness issue in the cooldown flow; the remaining items are low or informational and mainly relate to edge-case behavior, operational assumptions, or documentation clarity.

1.2. Project Timeline

Date	Phase
January 26th, 2026	Kickoff
January 30th, 2026	Audit End
January 30th, 2026	Delivery

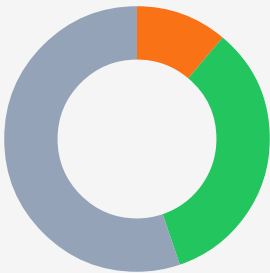
1.3. Scope

Repositories	https://github.com/Axis-Coordinate/contracts
Version	98849c540e5f51aef666a4651d7728087211a030
Contracts	src/
Type	Solidity
Platform	EVM

1.4. Overview of Findings

Our comprehensive review yielded 0 high, 1 medium, 3 low and 5 informational issues.

Severity/Impact Level	Count
High	0
Medium	1
Low	3
Informational	5



2. Findings

2.1. [M-1] Cooldown Can Be Shortened for Existing Locked Assets

Target	src/StakedAxisUSDV2.sol
Severity	● Medium
Category	Fairness Issue

2.1.1. Description

`cooldownAssets()` and `cooldownShares()` overwrite `cooldownEnd` on each call and accumulate `underlyingAmount`. If governance reduces `cooldownDuration`, a user can perform a tiny cooldown transaction to reset `cooldownEnd` to the new, shorter value and thus unlock all previously locked assets earlier than intended. This is a fairness issue because only users who know the trick will do it and have an advantage, while other users still have to go through the original, longer cooldown.

The issue is more theoretical than exploitable in realistic scenarios. The point is to have well-defined behavior when governance reduces `cooldownDuration`: users should have equal advantage whether they know this trick or not.

2.1.2. Impact

When governance shortens the cooldown, users who are aware of this behavior can fast-track withdrawal of previously cooled funds, defeating the intended liquidity friction. The effect does not apply when the cooldown is increased.

2.1.3. Recommendation

Change the logic so that the cooldown functions only extend cooldown and never shorten.

2.1.4. Remediation

Verified as fixed in commit `e45b8b6a6857c49ae840da11814738de5657b2ed`.

2.2. [L-1] Blacklist Can Be Frontrun to Transfer Out Before Restriction

Target	<code>src/StakedAxisUSD.sol</code> , <code>src/StakedAxisUSDV2.sol</code>
Severity	● Low
Category	Design Assumption

2.2.1. Description

Blacklisting is enforced on-chain only after `addToBlacklist()` is mined. A user who is not yet fully restricted can observe the pending blacklist transaction and submit a transfer before it is included. The transfer succeeds because `_update()` only blocks transfers after the address has `FULL_RESTRICTED_STAKER_ROLE`. This means there is an inherent mempool frontrun window in which tokens can be moved out prior to restriction.

2.2.2. Impact

In practice, a targeted user can often evade a pending blacklist by transferring out before the blacklist transaction lands. This reduces the effectiveness of `redistributeLockedAmount()` against an active adversary and is a limitation of public mempool execution.

2.2.3. Recommendation

Treat this as an operational limitation. If strict enforcement is required, use private transaction channels or pre-emptive restriction flows (e.g., pausing transfers first, or executing the blacklist through a private RPC).

2.2.4. Remediation

Acknowledged.

2.3. [L-2] `setOperator()` with Same Address Downgrades Delegation

Target	<code>src/StakingRewardsDistributor.sol</code>
Severity	● Low
Category	Logic Error

2.3.1. Description

`setOperator()` always removes the previous delegation and re-initiates it as `PENDING`, even if `_newOperator` equals the current `operator`. This temporarily breaks signature validation until the operator re-confirms delegation.

```
1 | mintContract.removeDelegatedSigner(operator);  
2 | mintContract.setDelegatedSigner(_newOperator);
```

2.3.2. Impact

An accidental call with the current operator causes temporary mint/redeem downtime until the operator re-confirms delegation.

2.3.3. Recommendation

Add an equality guard: `if (_newOperator == operator) return;`

2.3.4. Remediation

Verified as fixed in commit `0af398a7ab0aa2639d498b233cb3b0bd34bc88ae`.

2.4. [L-3] Minting-Contract Rotation Breaks Delegation Until Re-Set

Target	src/StakingRewardsDistributor.sol
--------	-----------------------------------

Severity	● Low
----------	-------

Category	Logic Error
----------	-------------

2.4.1. Description

Delegated signer state is stored in each `AxisUSDMinting` instance. `setMintingContract()` only updates the pointer and does not re-establish the operator delegation on the new contract.

```
1 | mintContract = _newMintingContract;
```

2.4.2. Impact

After rotating the minting contract, operator signatures fail until delegation is explicitly re-set and confirmed, causing avoidable downtime.

2.4.3. Recommendation

Re-establish delegation as part of `setMintingContract()` (e.g., call `setOperator(operator)` after the update) or add a separate migration step that is required before resuming operations.

2.4.4. Remediation

Acknowledged.

2.5. [I-1] Remainder Dust Always Routes to Last Custodian

Target	src/AxisUSDMinting.sol
Severity	● Informational
Category	Dust Amount

2.5.1. Description

`_transferCollateral()` splits collateral using integer division and sends the entire remainder to the last custodian in the route. Because route order is caller-supplied, the last custodian receives a systematic rounding advantage.

```

1  uint256 amountToTransfer = (amount * ratios[i]) / ROUTE_REQUIRED_RATIO;
2  // ...
3  uint256 remainingBalance = amount - totalTransferred;
4  if (remainingBalance > 0) {
5      token.safeTransferFrom(benefactor, addresses[addresses.length - 1], remainingBalance);
6  }
```

2.5.2. Impact

Over many mints, rounding dust accumulates in the last custodian, creating a deterministic bias in custody allocation. This is small per mint but can matter at scale or during reconciliation.

2.5.3. Recommendation

Document the rounding bias or distribute the remainder more fairly (e.g., assign remainder to the largest ratio, rotate the recipient, or compute the last leg as `amount - sum(previous)` with a fixed, policy-defined ordering).

2.5.4. Remediation

Acknowledged.

2.6. [I-2] `safeIncreaseAllowance` Can Overflow on Re-Approval

Target	<code>src/StakingRewardsDistributor.sol</code>
--------	--

Severity	● Informational
----------	-----------------

Category	Configuration
----------	---------------

2.6.1. Description

`approveToMintContract()` uses `safeIncreaseAllowance(..., type(uint256).max)`. If called when an allowance is already non-zero, the addition can overflow under Solidity 0.8 checks and revert, blocking re-approval workflows.

```
1 | IERC20(_assets[i]).safeIncreaseAllowance(address(mintContract), type(uint256).max);
```

2.6.2. Impact

Owner operations that re-run approvals can revert, causing avoidable administrative downtime.

2.6.3. Recommendation

Use `forceApprove` to set allowances to a known value, or set to zero then approve max:

```
1 | IERC20(asset).forceApprove(spender, 0);
2 | IERC20(asset).forceApprove(spender, type(uint256).max);
```

2.6.4. Remediation

Verified as fixed in commit `973c65872166e1f8092984d38332cf8107570568`.

2.7. [I-3] Virtual-Share Offset Outdated Comment

Target	<code>src/StakedAxisUSD.sol</code> , <code>src/StakedAxisUSDV2.sol</code>
--------	---

Severity	● Informational
----------	-----------------

Category	Configuration Error
----------	---------------------

2.7.1. Description

`StakedAxisUSDV2` documentation claims it inherits a virtual-share offset of 3, but neither `StakedAxisUSD` nor `StakedAxisUSDV2` overrides `_decimalsOffset()`. In OpenZeppelin ERC-4626, the default offset is 0, which yields only 1 virtual share and 1 virtual asset. This is intended, as the contract uses `MIN_SHARE` to prevent inflation attacks. The issue is just an outdated comment.

2.7.2. Impact

Outdated comment can cause confusion.

2.7.3. Recommendation

Update documentation/comments to match the actual behavior.

2.7.4. Remediation

Verified as fixed in commit `7111dce2d527fd9bafff88610dbfb8c1cf2e29dd`.

2.8. [I-4] AxisUSDRateProvider Rate Is Manipulable via Direct Transfers

Target	src/AxisUSDRateProvider.sol
Severity	● Informational
Category	Calculation Error

2.8.1. Description

AxisUSDRateProvider computes the rate as `totalAssets() / totalSupply()` from `StakedAxisUSDV2`. Because `totalAssets()` is based on the vault's raw token balance, any direct transfer to the vault increases `totalAssets()` without minting shares, which inflates the reported rate.

```

1  function getRate() external view returns (uint256) {
2      uint256 _totalSupply = stakedAxisUSD.totalSupply();
3      if (_totalSupply == 0) {
4          return 0;
5      } else {
6          return (stakedAxisUSD.totalAssets() * 1 ether) / _totalSupply;
7      }
8  }
```

Since donations are permissionless, the rate can be arbitrarily moved upward by transferring AxisUSD directly to the vault. This does not create profit by itself but makes the rate unreliable for any downstream usage that assumes it is resistant to manipulation.

2.8.2. Impact

Any integration using `getRate()` for pricing, collateral valuation, or accounting can be mispriced by a third party. Even if not directly exploitable today, it creates an oracle footgun and can lead to future integration risk.

2.8.3. Recommendation

Use an “accounted assets” value that only changes on trusted flows (deposit, withdraw, and reward distribution), and base `getRate()` on that value instead of raw balance. Example pattern:

```
1  uint256 public accountedAssets;
2
3  // increment on deposit/reward transfer, decrement on withdraw
4  function _afterDeposit(uint256 assets) internal {
5      accountedAssets += assets;
6  }
7
8  function _afterWithdraw(uint256 assets) internal {
9      accountedAssets -= assets;
10 }
11
12 function getRate() external view returns (uint256) {
13     uint256 supply = stakedAxisUSD.totalSupply();
14     if (supply == 0) return 0;
15     return (accountedAssets * 1 ether) / supply;
16 }
```

At minimum, document `getRate()` as donation-manipulable so it is not used as a price oracle.

2.8.4. Remediation

Verified as fixed in commit `1fa87957ce9cef2b2c76e69d4920398f7af71d04` .

2.9. [I-5] Cannot Revoke Allowances from Current Minting Contract

Target	src/StakingRewardsDistributor.sol
Severity	● Informational
Category	Incident Response

2.9.1. Description

`revokeApprovals()` explicitly reverts when `_target` equals the active `mintContract`, which prevents the owner from zeroing allowances for the current minting contract. If the active minting contract is misconfigured or compromised, allowances remain live until the owner switches to a new minting contract.

```

1  function revokeApprovals(address[] memory _assets, address _target) external onlyOwner {
2      if (_target == address(mintContract)) {
3          revert InvalidAddressCurrentMintContract();
4      }
5      // ...
6  }
```

2.9.2. Impact

Operational response is delayed during an incident: the owner cannot immediately revoke the largest and most sensitive allowances (the active `mintContract`). This increases the blast radius for a compromised minting contract.

2.9.3. Recommendation

Allow revoking the current minting contract's allowances in emergencies. For example, add an owner-only function to force-approve zero for the active `mintContract`, or remove the `_target == mintContract` restriction.

2.9.4. Remediation

Acknowledged.