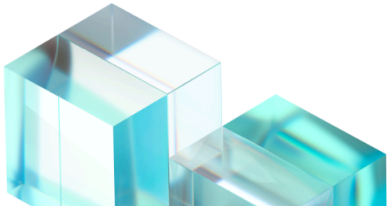
LinumLabs

AmanaHQ- Audit

Prepared by Linum Labs AG

2025-03-04

linumlabs.com
info@linumlabs.com
+1 (650) 534-2011



linumlabs.com
info@linumlabs.com
+1 (650) 534-2011

Executive Summary

This document outlines the security review of the AmanaHQ smart contracts. Linum Labs Auditing has identified several vulnerabilities that should be addressed before deploying smart contracts to any live chain. The report outlines these vulnerabilities and advises on how to fix them.

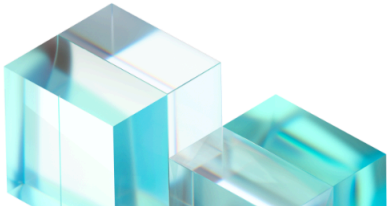
Protocol Summary

Overview

The Amana platform provides users with a robust, decentralized system to optimize their investments through yield-bearing crypto vaults. The project leverages cutting-edge blockchain interoperability to enable cross-chain operations, ensuring accessibility and scalability.

Audit Scope

../contracts/AmanaVaultBase.sol
../contracts/AmanaConnectedChainVault.sol
../contracts/AmanaZetachainVault.sol
../contracts/GasTank.sol
../contracts/ERC4626RewardsUpgradeable.sol
../contracts/Treasury.sol
../contracts/PriceOracle.sol



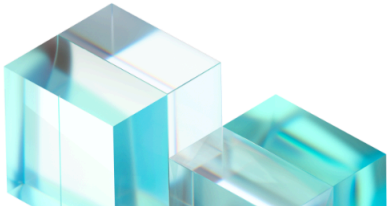
linumlabs.com
info@linumlabs.com
+1 (650) 534-2011

../contracts/WithdrawalReceiver.sol
../contracts/strategies/ERC20_4626_Strategy.sol
../contracts/strategies/ERC20_StrategyParent.sol
../contracts/strategies/Eth_4626_Strategy.sol
../contracts/strategies/EthStrategyParent.sol
../contracts/strategies/MockERC4626ZetachainStrategy.sol
../contracts/strategies/StrategyParent.sol
../contracts/strategies/SwapHelperLibEddy.sol

Audit Results

Summary

Repository	https://github.com/AmanaDefi/Amana-monorepo
Commit	3b72de8...
Timeline	January 24th - February 5th



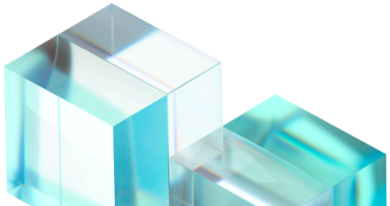
linumlabs.com
info@linumlabs.com
+1 (650) 534-2011

Issues Found

Bug Severity	Count
Critical	3
High	3
Low	8
Informational	1
Total Findings	15

Summary of Issues

Description	Severity	Status
Inconsistent ERC-20 boolean returns	Critical	Resolved
Incorrect slippage calculation: Missing token decimals	Critical	Resolved
Missing access control in *onRevert()* function	Critical	Resolved
Possible sandwich attacks in *invest()* and *withdraw()* functions	High	Resolved
Incorrect assumption about ERC-20 approve return value	High	Resolved
ERC4626RewardsUpgradeable is vulnerable to sandwich attacks due to missing slippage protection	High	Resolved
State *maxStaleness* is never used	Low	Resolved
Function *setMaxStaleness()* lacks access control	Low	Resolved



linumlabs.com
info@linumlabs.com
+1 (650) 534-2011

Gas limit restriction in *transfer()* function can cause failed transactions	Low	Resolved
Implement two-step governance	Low	Resolved
Inconsistent balance check	Low	Resolved
Deprecated definitions in imported *UniversalContract* library	Low	Acknowledged
Hardcoded address	Low	Partially Resolved, Acknowledged
Missing events for state changes	Low	Resolved
Redundant *emergencyWithdrawETH()* function	Informational	Resolved

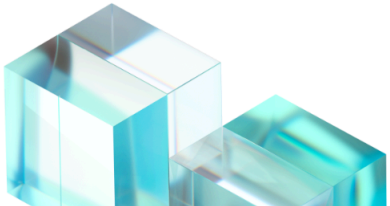
Issues

Critical Severity

1. Inconsistent ERC-20 boolean returns

Description: The ERC-20 standard (EIP-20) recommends that the *transfer()* and *transferFrom()* functions return a boolean value indicating success or failure. However, not all ERC-20 tokens adhere to this recommendation. Some tokens omit the return value entirely, while others might return different data types or behave inconsistently. Relying on the boolean return value can lead to unexpected behavior and potential vulnerabilities when interacting with non-compliant tokens.

Potential Risk: Even if a token transfer succeeds, if the token doesn't return a boolean, your contract might incorrectly interpret the missing return value as false



and revert the transaction or take other incorrect actions. This can lead to denial-of-service or prevent legitimate operations.

Suggested Mitigation: Use OpenZeppelin's *SafeERC20* library.

AmanaHQ: Fixed in PR

Linum Labs: Verified

Related Issue: https://github.com/AmanaDefi/Amana-monorepo/issues/21

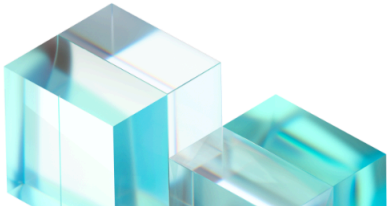
2. Incorrect slippage calculation: Missing token decimals

Description: The *calculateMinAmountOut()* function does not account for the decimal places of the input and output tokens when calculating slippage. This can lead to significant inaccuracies in the computed minimum amount, especially when dealing with tokens with different decimal places.

Potential Risk: The lack of decimal handling in the slippage calculation can result in:

- Incorrect Minimum Amounts: The calculated *minAmountOut* might significantly differ from the intended value, leading to unexpected behavior for users.
- Failed Transactions: If the calculated *minAmountOut* is too low, the swap transaction might revert due to exceeded slippage tolerance.
- Financial Loss: Inaccurate slippage calculations can create opportunities for arbitrage or other exploits, potentially leading to user financial losses.

Suggested Mitigation: Modify the *calculateMinAmountOut()* function to incorporate the decimal places of both the input and output tokens in the slippage calculation. Fetch the decimals using:



```
Unset
IERC20(token).decimals();
```

AmanaHQ: Fixed in PR

Linum Labs: Verified

Related Issue: https://github.com/AmanaDefi/Amana-monorepo/issues/26

3. Missing access control in *onRevert()* function

Description: The *onRevert()* function handles reverts from the Gateway contract. However, it lacks any access control mechanism. This means anyone can call this function, providing arbitrary *RevertContext* data. The function attempts to recover funds if a withdrawal fails on the destination chain by decoding revert data, swapping tokens, and attempting a withdrawal back to the source chain. This approach minimizes the risk of lost funds and ensures a robust fallback mechanism. However, without access control, significant risks are created.

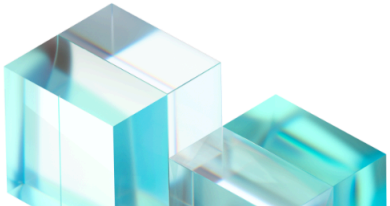
Potential Risk: Attackers can call *onRevert()* with crafted *RevertContext* data, potentially triggering incorrect logic and causing loss of funds or other unintended consequences. They could manipulate the *_crossChainTxId* and *revertMessage* to their advantage.

Suggested Mitigation: Restrict access to the *onRevert()* function to only the Gateway contract. This can be achieved using the *onlyGateway* modifier.

AmanaHQ: Fixed in PR

Linum Labs: Verified

Related Issue: https://github.com/AmanaDefi/Amana-monorepo/issues/31



linumlabs.com
info@linumlabs.com
+1 (650) 534-2011

High Severity

4. Possible sandwich attacks in *invest()* and *withdraw()* functions

Description: The *invest()* and *withdraw()* functions are vulnerable to sandwich attacks due to the lack of slippage protection. These attacks exploit the ability of a malicious actor to manipulate the price of the *inputToken* relative to the *receiptToken* around the execution of a victim's transaction.

- In the *invest()* function, the attacker can front-run a user's transaction to inflate the price of the *inputToken*, causing the user to receive fewer *receiptToken* shares for their deposit.
- In the *withdraw()* function, the attacker can front-run a user's transaction to deflate the price of the *inputToken*, causing the user to receive less *inputToken* for their redeemed *receiptToken* shares.

Potential Risk:

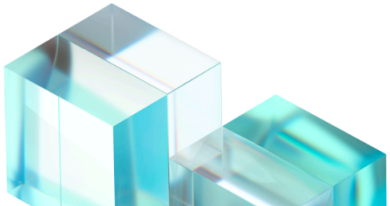
- Loss of Funds (Direct or Indirect): Users can lose value due to unfavorable exchange rates caused by the sandwich attack. When they invest, they receive fewer shares. When they withdraw, they receive less *inputToken* for their shares.
- Profit for Attacker: The attacker profits from the price manipulation.

Suggested Mitigation: Implement slippage control in both functions by allowing the user (via the vault) to specify a minimum number of shares or *inputToken* they are willing to accept.

AmanaHQ: Fixed in Commit

Linum Labs: Verified

Related Issue: https://github.com/AmanaDefi/Amana-monorepo/issues/30



linumlabs.com
info@linumlabs.com
+1 (650) 534-2011

5. Incorrect assumption about ERC-20 approve return value

Description: The code assumes that the ERC-20 *approve()* function will always return true on success and false on failure. This assumption is incorrect. Some ERC-20 tokens, especially older ones or those with custom implementations, may not return a boolean value at all or may return a boolean with different semantics. Relying on the return value of *approve()* can lead to unexpected behavior and potential vulnerabilities. If the approve call succeeds but doesn't return true (or returns nothing), the require statement will revert the transaction, even though the approval might have been successful. Conversely, if the *approve()* call fails but returns true (or returns nothing), the transaction will proceed, potentially leading to a situation where the contract attempts to transfer tokens without proper approval.

Potential Risk: If the *approve()* call succeeds but doesn't return true, the transaction will revert, even though the approval might have been successful. This can disrupt the contract's intended functionality.

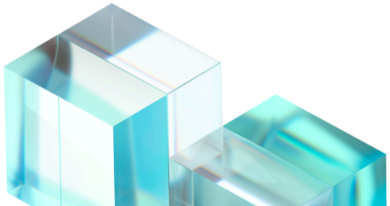
Suggested Mitigation:

- The most robust and preferred approach is to check the allowance before and after the *approve()* call. This verifies that the allowance has been set correctly, regardless of the *approve()* return value.
- The OpenZeppelin *SafeERC20* library handles ERC-20 interactions securely, including the approve function. *SafeERC20* is highly recommended as it prevents many common ERC-20 issues and simplifies development.

AmanaHQ: Fixed in PR

Linum Labs: Verified

Related Issue: https://github.com/AmanaDefi/Amana-monorepo/issues/29.



linumlabs.com
info@linumlabs.com
+1 (650) 534-2011

6. *ERC4626RewardsUpgradeable* is vulnerable to sandwich attacks due to missing slippage protection

Description: The *ERC4626RewardsUpgradeable* contract, while extending the ERC4626 standard with reward functionality, lacks crucial slippage protection during deposits, mints, withdrawals, and redeems. This absence makes users highly susceptible to sandwich attacks, a form of front-running exploit where malicious actors manipulate the exchange rate between the vault's shares and underlying assets to profit at the expense of legitimate users.

Potential Risk: Users may receive fewer shares on deposits/mints or fewer assets on withdrawals/redeems than expected, resulting in direct financial losses.

Suggested Mitigation: Implement slippage protection by adding *minShares* (for deposits/mints) and *minAssets* (for withdrawals/redeems) parameters. Before any asset transfers or share manipulation, verify that the calculated shares/assets are greater than or equal to the user's specified minimum; revert if the slippage tolerance is exceeded.

AmanaHQ: Fixed in Branch

Linum Labs: Verified

Related Issue: https://github.com/AmanaDefi/Amana-monorepo/issues/34

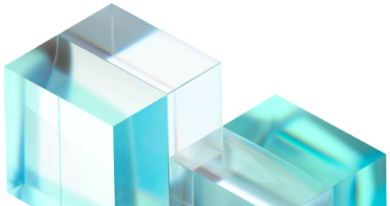
Low Severity

7. State *maxStaleness* is never used

Description: The state variable *maxStaleness* is declared in the *PriceOracle.sol* contract but is never read or used within any of the contract's functions.

Potential Risk: The primary risk is that the intended staleness check, which *maxStaleness* was presumably meant to implement, is not being performed.

Suggested Mitigation:



linumlabs.com
info@linumlabs.com
+1 (650) 534-2011

- Implement staleness checks: If *maxStaleness* was intended to be used, then the contract logic needs to be updated to incorporate it.
- Remove the unused variable: If *maxStaleness* is unnecessary, the simplest solution is to remove the declaration entirely. This will save gas and eliminate a source of potential confusion.

AmanaHQ: Fixed in PR

Linum Labs: Verified

Related Issue: https://github.com/AmanaDefi/Amana-monorepo/issues/19

8. Function *setMaxStaleness()* lacks access control

Description: The *setMaxStaleness()* function, intended to modify the *maxStaleness* state variable, lacks proper access control. This means that anyone can call this function and change the value of *maxStaleness*, potentially disrupting the intended behavior of the price oracle.

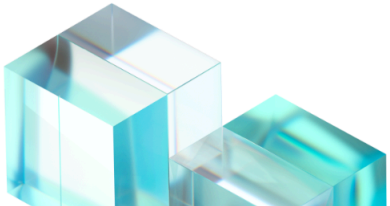
Potential Risk: The primary risk is an unauthorized modification of the *maxStaleness* value. Because *setMaxStaleness()* lacks access control, anyone can call it and change the value.

Suggested Mitigation: The most common and effective mitigation is restricting access to *setMaxStaleness()* using an access control mechanism. These could include using an owner/admin variable or implementing widely used libraries such as OpenZeppelin's Ownable or Ownable2Step.

AmanaHQ: Fixed in PR

Linum Labs: Verified

Related Issue: https://github.com/AmanaDefi/Amana-monorepo/issues/20



9. Gas limit restriction in *transfer()* function can cause failed transactions

Description: The Solidity *transfer()* function, used for sending Ether, forwards a fixed amount of gas (2300 gas units as of the Byzantium hard fork) to the recipient contract's *fallback()* or *receive()* function. While *transfer()* has a low base gas cost, the limited gas forwarding can cause issues when interacting with contracts requiring more gas for their execution. If the recipient contract's logic (e.g., within its fallback/receive function) consumes more than 2300 gas units, the transaction will revert due to an out-of-gas error, even if the sender provided a sufficient gas limit for the overall transaction.

Potential Risk: The fixed gas limit of *transfer()* creates the risk of failed Ether transfers when the recipient is a contract with more complex logic in its fallback or receive function.

Suggested Mitigation: The recommended mitigation is to avoid using *transfer()* for interactions with contracts whenever the recipient contract may require more than the fixed gas limit. Instead, use the low-level *call()* function, which allows for the specification of the amount of gas to be moved forward.

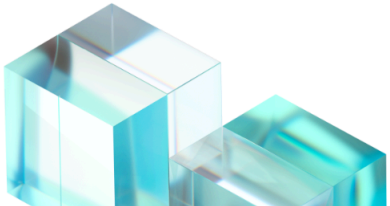
AmanaHQ: Fixed in Commit

Linum Labs: Verified

Related Issue: https://github.com/AmanaDefi/Amana-monorepo/issues/22

10. Implement two-step governance

Description: The current governance transfer mechanism in multiple contracts uses a single-step process to update the governance address. This approach poses security risks as it allows for immediate and potentially irreversible changes to the governance control. A more secure and recommended approach is a two-step process involving a pending governance address and a separate acceptance step.



linumlabs.com
info@linumlabs.com
+1 (650) 534-2011

Potential Risk: The single-step governance transfer process creates the primary risk of accidental changes to the governance address. A simple typo or mistake when setting the new governance address could lead to unintended and potentially irreversible loss of control. While malicious takeovers are also a concern, they are not directly mitigated by the two-step process unless a time delay or veto mechanism is added. The two-step process primarily protects against unintentional errors.

Suggested Mitigation: Implement a two-step governance transfer process using a *pendingGovernance* address and separate functions for setting and accepting the new governance. This can easily be achieved through Openzeppelin's *Ownable2Step* library.

AmanaHQ: Fixed in PR, as well as PR

Linum Labs: Verified

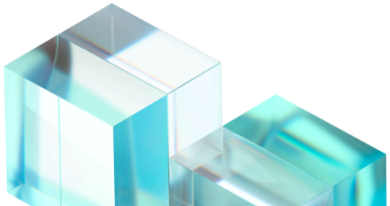
Related Issue:

- https://github.com/AmanaDefi/Amana-monorepo/issues/36
- https://github.com/AmanaDefi/Amana-monorepo/issues/23

11. Inconsistent balance check

Description: The *getGas()* and *withdrawGasToken()* functions use an if statement to check for sufficient balance before transferring ZRC20 tokens. This check is too restrictive.

```
Unset
contracts/GasTank.sol
if (balance < amount)
```

linumlabs.com
info@linumlabs.com
+1 (650) 534-2011

Potential Risk: The overly restrictive balance check in *getGas()* and *withdrawGasToken()* prevents authorized vaults (in the case of *getGas()*) and the contract owner (in the case of *withdrawGasToken()*) from withdrawing the full amount of ZRC20 tokens available to the contract.

Suggested Mitigation: Update the balance check too:

```
Unset
if (balance <= amount)
```

AmanaHQ: Fixed in Commit

Linum Labs: Verified

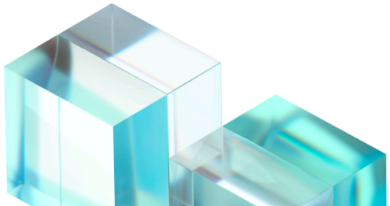
Related Issue: https://github.com/AmanaDefi/Amana-monorepo/issues/25

12. Deprecated definitions in imported *UniversalContract* library

Description: The *AmanaVaultBase* contract imports and inherits the *UniversalContract* library from the ZetaChain protocol contracts. This library contains deprecated definitions for *zContext* and *zContract*, marked for removal once the v2 SystemContract is upgraded or unused.

Potential Risk:

- Technical Debt: The deprecated code in the imported library adds complexity and makes the project harder to understand and maintain. It also confuses developers about which definitions to use.
- Potential for Accidental Usage: Developers working on your contract might accidentally use the deprecated *zContext* or *zContract* if they are not careful.



Suggested Mitigation: Implement the mitigation strategy recommended by the ZetaChain team.

AmanaHQ: They have discussed this closely with the ZetaChain team and received the following response: “Hey! These interfaces will remain in the codebase, because even though we're migrating to Gateway/onCall, there are still some scenarios where onCrossChainCall might be executed, for example when you deposit directly to a TSS address/ERC-20 custody. We're not advertising these features, because we want to migrate devs to Gateway, but they're still there for backwards compatibility. And since these are just interfaces, they don't actually have any code associated with them.”

Linum Labs: Acknowledged

Related Issue: https://github.com/AmanaDefi/Amana-monorepo/issues/32

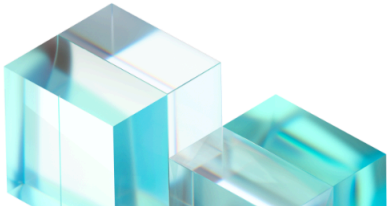
13. Hardcoded addresses

Description: Multiple contracts contain hardcoded or static addresses for critical dependencies.

Potential Risk:

- The contract can only function on the chain where the hardcoded addresses are valid. Deploying to a new chain requires modifying the code and redeploying.
- Manually changing addresses increases the risk of introducing errors, which could lead to contract failures or security vulnerabilities.

Suggested Mitigation: The most common and recommended approach is to use the constructor to pass addresses as arguments during deployment.



AmanaHQ: Partially fixed in Commit, these vault contracts are only deployed on one chain—the Zeta chain. Omnichannel functionality is achieved by deploying strategies on different chains.

Linum Labs: Verified

Related Issue: https://github.com/AmanaDefi/Amana-monorepo/issues/35

14. Missing events for state changes

Description: Several contracts are missing event emissions for significant state changes. This makes it difficult to track and audit contract behavior, debug issues, and integrate with off-chain applications that rely on event data. This is a general recommendation to improve all contracts in the codebase.

Potential Risk: Off-chain applications (like transaction explorers, monitoring tools, and automated processes) often rely on events to track contract activity. Missing events hinder or prevent this integration.

Suggested Mitigation: Implement comprehensive event emissions in all contracts to cover all relevant state changes.

AmanaHQ: Fixed in PR

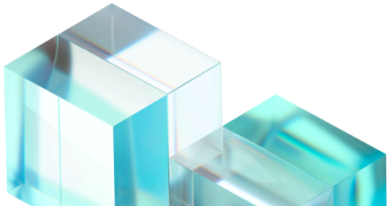
Linum Labs: Verified

Related Issue: https://github.com/AmanaDefi/Amana-monorepo/issues/37

Informational Severity

15. Redundant *emergencyWithdrawETH()* function

Description: The *emergencyWithdrawETH()* function is intended to allow the contract owner to withdraw ETH in an emergency. However, the contract will never hold any ETH without a payable function, a receive function, or a fallback function.



linumlabs.com
info@linumlabs.com
+1 (650) 534-2011

Therefore, this function is currently redundant and will always revert. There's no way for the contract ever to receive ETH.

Potential Risk: Gas wasted on deployment and potential execution. The function adds unnecessary gas costs for deployment. While the gas cost of calling the function would be relatively low (and would revert anyway), it is still a tiny waste. More importantly, it adds to the contract's deployed bytecode size, increasing deployment costs.

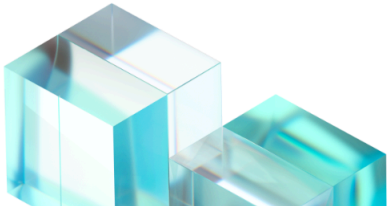
Suggested Mitigation:

- Remove the function: The simplest and most effective solution is to remove the *emergencyWithdrawETH()* function entirely. This eliminates redundancy, avoids confusion, and saves gas costs. If the contract is not designed to hold ETH, it shouldn't have a function to withdraw ETH.
- Implement a receive function: If there's a legitimate reason for the contract to receive ETH, implement a *receive()* function (or a *fallback()* function if you need to handle other data as well) marked as payable. This clarifies how and when the contract is intended to receive ETH. Then, the *emergencyWithdrawETH()* function becomes meaningful.

AmanaHQ: Fixed in PR

Linum Labs: Verified

Related Issue: https://github.com/AmanaDefi/Amana-monorepo/issues/28



linumlabs.com
info@linumlabs.com
+1 (650) 534-2011

Disclaimer

This report is based on the materials and documentation provided to Linum Labs Auditing for the purpose of conducting a security review, as outlined in the Executive Summary and Files in Scope sections. It's important to note that the results presented in this report may not cover all vulnerabilities. Linum Labs Auditing provides this review and report on an as-is, where-is, and as-available basis. By accessing and/or using this report, along with associated services, products, protocols, platforms, content, and materials, you agree to do so at your own risk. Linum Labs Auditing disclaims any liability associated with this report, its content, and any related services and products to the fullest extent permitted by law. This includes implied warranties of merchantability, fitness for a particular purpose, and non-infringement. Linum Labs Auditing does not warrant, endorse, guarantee, or assume responsibility for any third-party products or services advertised or offered through this report, its content, or related services and products. Users should exercise caution and use their best judgment when engaging with third-party providers. It's important to clarify that this report, its content, access, and/or usage thereof, including associated services or materials, should not be considered or relied upon as financial, investment, tax, legal, regulatory, or any other form of advice.