Audit Report

Amulet

v1.0

January 8, 2024

Table of Contents

License

Disclaimer

THE CONTENT OF THIS AUDIT REPORT IS PROVIDED “AS IS”, WITHOUT REPRESENTATIONS AND WARRANTIES OF ANY KIND.

THE AUTHOR AND HIS EMPLOYER DISCLAIM ANY LIABILITY FOR DAMAGE ARISING OUT OF, OR IN CONNECTION WITH, THIS AUDIT REPORT.

THIS AUDIT REPORT WAS PREPARED EXCLUSIVELY FOR AND IN THE INTEREST OF THE CLIENT AND SHALL NOT CONSTRUE ANY LEGAL RELATIONSHIP TOWARDS THIRD PARTIES. IN PARTICULAR, THE AUTHOR AND HIS EMPLOYER UNDERTAKE NO LIABILITY OR RESPONSIBILITY TOWARDS THIRD PARTIES AND PROVIDE NO WARRANTIES REGARDING THE FACTUAL ACCURACY OR COMPLETENESS OF THE AUDIT REPORT.

FOR THE AVOIDANCE OF DOUBT, NOTHING CONTAINED IN THIS AUDIT REPORT SHALL BE CONSTRUED TO IMPOSE ADDITIONAL OBLIGATIONS ON COMPANY, INCLUDING WITHOUT LIMITATION WARRANTIES OR LIABILITIES.

This audit has been performed by

Oak Security GmbH

https://oaksecurity.io/
info@oaksecurity.io

Introduction

Purpose of This Report

Oak Security has been engaged by Gaia Projects Limited to perform a security audit of Amulet.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine smart contract bugs, which might lead to unexpected behavior.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebase Submitted for the Audit

The audit has been performed on the following target:

Repository	https://github.com/Amulet-Dev/protocol
Commit	`10c7320c0fc892b9fa604c38361ffe030fd2aa6e`
Scope	The scope of this audit was limited to the `hub` and `vault` contracts. The following files were **excluded** from the audit: • `contracts/vanilla-cw/amulet-mock-strategy` • `crates/core/strategy` • `crates/core/strategy.rs` • `crates/cosmwasm/iface/amulet-strategy-cw.rs` • `crates/cosmwasm/parse/strategy.rs` • `crates/cosmwasm/vanilla/mock_strategy/*` • `crates/cosmwasm/vanilla/mock_strategy.rs` • `crates/storage/strategy.rs`
Fixes verified at commit	`e6e8e41bee3a0b7fbecdb3bdcbe195b85dc101c1` The following pull requests are also reviewed up to the specified commit hashes: • PR #33: `d81cf39e77c88a8534234dfb611af8e7bebb8a56` • PR #34: `ff8f355a22cc46b66b07684ca9401c15daaaa355` • PR #35: `e8cba37223bfb66f4935e60e3dcc9332c5a6368c` • PR #36: `846b49c9e83d94e64a710f63911f9130ee0b282c` Note that changes to the codebase beyond fixes after the initial audit have not been in the scope of our fixes review.

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line-by-line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
4. Report preparation

Functionality Overview

The audit covers parts of Amulet protocol, a decentralized finance lending system deployed on the Neutron chain. The protocol allows users to earn staking rewards and yields on various crypto assets and secure an advance that self-repays from their future yield.

How to Read This Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending,** **Acknowledged**, or **Resolved**.

Note that audits are an important step to improving the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than in a security audit and vice versa.

Code Quality Criteria

The auditor team assesses the codebase's code quality criteria as follows:

Criteria	Status	Comment
Code complexity	**High**	The project's architecture is novel compared to the other CosmWasm codebases within the ecosystem. The goal of the architecture is to build platform-agnostic smart contracts with as few external dependencies as possible, which creates complexity due to requiring core API traits, wrappers, and parsers to be implemented for handling by the core logic.
Code readability and clarity	**Medium**	Although common programmatic patterns were used, the non-standard architecture made execution paths harder to follow.
Level of documentation	**Medium-High**	Besides the detailed documentation in the repository, the client has also provided a high-level architectural overview and valuable diagrams.
Test coverage	**Medium-High**	`cargo tarpaulin` reports a test coverage of `80.65%`.

Summary of Findings

No	Description	Severity	Status
1	Excess reward distribution due to incorrect reserve shares calculation	Critical	Resolved
2	Repaying and redeeming synthetic tokens does not normalize decimals to primary deposit asset	Critical	Resolved
3	Unimplemented oracle querying feature	Minor	Resolved
4	Potential bypass of access control in the `hub` contract	Minor	Acknowledged
5	Potential out-of-gas in `AllVaults` query	Informational	Acknowledged
6	Smart queries expose insufficient contract storage states	Informational	Acknowledged
7	Duplicated authorization code	Informational	Resolved
8	Repeated validation on callback handling	Informational	Resolved
9	Unoptimized deposit asset validation	Informational	Acknowledged
10	Contracts should implement a two-step ownership transfer	Informational	Acknowledged
11	Overflow checks are not enabled for the release profile	Informational	Acknowledged
12	Misleading comments in the codebase	Informational	Resolved

Detailed Findings

1. Excess reward distribution due to incorrect reserve shares calculation

Severity: Critical

In `crates/core/hub/position.rs:286`, the `reserve_shares_payment` value is calculated by subtracting `amo_shares_allocation` from `total_issued_shares`. This is problematic because `total_issued_shares` denotes the total shares within the protocol, not the excess shares intended to be distributed to depositors.

Since the contract does not hold the actual reward amounts, users will unknowingly withdraw from other depositors' balances once the accrued rewards have been exhausted. Consequently, late-withdrawal users will be unable to withdraw due to insufficient funds, ultimately risking the protocol's solvency.

Recommendation

We recommend modifying the `crates/core/hub/position.rs:287` calculation to subtract `amo_shares_allocation` from `total_payment_shares`.

Status: Resolved

2. Repaying and redeeming synthetic tokens does not normalize decimals to primary deposit asset

Severity: Critical

In `crates/core/hub/position.rs:1221` and lines `1617-1634`, the `amount` parameter denominated in synthetic tokens is used to repay debt and redeem underlying assets. This is problematic because the user's debt and reserve redeemable are denominated in the primary deposit asset, not the synthetic token asset.

Since the synthetic token and primary deposit asset might have different decimal values, the sent amount should be normalized to the correct decimal before decreasing the user's debt or redeeming the underlying assets. For example, the `mint_synthetic` function in `crates/core/hub/position.rs:968-985` normalizes the primary deposit asset decimals to synthetic token decimals before minting them to the user.

Due to this misalignment, users will repay more debt with fewer funds and redeem more underlying assets if the number of synthetic token decimals is higher than that of the primary deposit asset. Conversely, if the synthetic token decimals are fewer than the primary deposit asset, users will still have outstanding debt even after repaying in full and redeeming fewer underlying assets.

Recommendation

We recommend normalizing the synthetic token decimals to match the primary deposit asset decimals during debt repayment and synthetic token redemption.

Status: Resolved

3. Unimplemented oracle querying feature

Severity: Minor

The `advance_fee_rate` function is expected to query the fee rate through the configured oracle address in `crates/cosmwasm/vanilla/hub.rs:153-159`. However, the function seems to be pending development as it only contains an `unimplemented` exception.

Oracle features are usually error-prone and should be reviewed with care. Although not a security issue, this finding has been raised as a minor risk to highlight the dangers of this functionality not undergoing thorough auditing.

Recommendation

We recommend dealing with any pending development and having the full feature undergo an audit.

Status: Resolved

4. Potential bypass of access control in the `hub` contract

Severity: Minor

The `hub` contract has implemented several access controls that the contract admin can manage. These controls are designed to limit certain actions within the contract. However, these restrictions may be bypassed through alternative entry points, thus circumventing the defenses.

Specifically, the `ConvertCredit` message allows users to convert their credit into collateral, subsequently increasing their collateralized debt position. This function might bypass the intended restrictions set by the `VAULT_DEPOSITS_ENABLED` storage state and deposit proxy. For instance, this could occur if the intended function of disabling a deposit is to prevent users from increasing their collateral or if the deposit proxy is specifically designed to limit collateral increases.

Similarly, the `SelfLiquidate` message allows users to fully liquidate their positions, receiving funds in synthetic tokens or primary deposit assets. This function might bypass the intended restrictions for the redeem and mint proxy. For example, if the redeem proxy's

purpose is to ensure that only the proxy can redeem underlying assets or if the mint proxy's purpose is to ensure that only the proxy can mint new synthetic tokens.

Consequently, restrictions set by the contract admin may be circumvented, potentially leading to unintended consequences.

Recommendation

We recommend assessing the entry points to ensure they align with the intended defense mechanisms and restrict them if they unintentionally circumvent established limitations.

Status: Acknowledged

The client states that this is mostly intentional in that once a user has a position, they do not want to impinge on the user's ability to control it. For example, if deposits are disabled after being previously enabled, users with existing credit should be allowed to convert them or self-liquidate them regardless of the current proxy state. This is mostly fine as the access controls (deposit/advance disabled) and proxies can deny the creation of new positions, apart from one case that they see so far, repaying debt into credit and then converting it to collateral, which can be done without an existing position. The client reasons that disabling this by not allowing a debt repayment unless the user's debt is > 0 should be sufficient.

5. Potential out-of-gas in `AllVaults` query

Severity: Informational

In `crates/core/hub/query.rs:75-79`, the `AllVaults` query message iterates through and returns all vaults registered in the `hub` contract. If many vaults are registered, the query might fail due to an out-of-gas error.

Recommendation

We recommend implementing a pagination mechanism to query the vaults in batches.

Status: Acknowledged

The client states that pagination will be added before mainnet deployment, along with other queries.

6. Smart queries expose insufficient contract storage states

Severity: Informational

The `hub` and `vault` contract's query messages expose minimal contract storage states. Important information such as the contract admin address, reserve shares, reserve redeemable, total deposits, and total shares are not exposed through smart queries.

Consequently, users and other smart contracts must perform raw queries to fetch these values, decreasing user experience and tying queries to the underlying storage implementation, which is error-prone.

Recommendation

We recommend implementing more query messages that expose additional contract storage states.

Status: Acknowledged

The client states that smart queries that are not required by the protocol to function or for testing will be added before mainnet deployment.

7. Duplicated authorization code

Severity: Informational

The `set_admin` function includes duplicated code in `crates/core/admin.rs:40-47`, which is the same code as the `authorize` function.

We classify this issue as an informational issue because duplicated code impacts maintainability, which could be error-prone.

Recommendation

We recommend removing the duplicated code and calling the `authorize` function instead.

Status: Resolved

8. Repeated validation on callback handling

Severity: Informational

The `deposit_callback` and `sync_callback` functions validate that the `vault` contract has the strategy and share asset contracts configured in storage at the beginning of their execution in `crates/core/vault/holdings.rs:495-501` and lines `581-583`. These same checks are already performed at the beginning of the deposit and synchronization execution flows. However, it is not possible to have them unset before the reply is handled by the mentioned functions, therefore making them redundant.

A similar instance was found in `crates/core/hub/position.rs:1129`, affecting the validation of the `advance` feature being enabled.

We classify this issue as an informational issue because repeated code unnecessarily increases gas costs.

Recommendation

We recommend removing the redundant code.

Status: Resolved

9. Unoptimized deposit asset validation

Severity: Informational

Validation on the received collateral asset is not done in the `hub` or `vault` contracts. Therefore, if an incorrect asset is submitted, it will cause an error in a later stage of the execution flow, unnecessarily wasting gas.

The following Core messages of the `hub` contract are affected:

- `CoreMsgKind::MintOnBehalf`
- `CoreMsgKind::Mint`
- `CoreMsgKind::Repay`
- `CoreMsgKind::DepositOnBehalf`
- `CoreMsgKind::Deposit`

Recommendation

We recommend checking that the submitted asset matches that of the collateral in one of the early handlers of the `hub` contract, as done in other validation steps.

Status: Acknowledged

The client states that the hub and vault contracts do not know what assets the strategy can accept for deposit. Therefore, they cannot validate assets earlier in a deposit execution chain, i.e., the strategy may be able to convert arbitrary assets into the 'primary deposit asset'.

10. Contracts should implement a two-step ownership transfer

Severity: Informational

The contracts within the scope of this audit allow the current owner to execute a one-step ownership transfer. While this is common practice, it presents a risk for the ownership of the contract to become lost if the owner transfers ownership to the incorrect address. A two-step ownership transfer will allow the current owner to propose a new owner, and then the account that is proposed as the new owner may call a function that will allow them to claim ownership and actually execute the config update.

Recommendation

We recommend implementing a two-step ownership transfer. The flow can be as follows:

1. The current owner proposes a new owner address that is validated.
2. The new owner account claims ownership, which applies the configuration changes.

Status: Acknowledged

11. Overflow checks are not enabled for the release profile

Severity: Informational

The following packages and contracts do not enable `overflow-checks` for the release profile:

- `contracts/vanilla-cw/amulet-hub/Cargo.toml`
- `contracts/vanilla-cw/amulet-mock-strategy/Cargo.toml`
- `contracts/vanilla-cw/amulet-vault/Cargo.toml`
- `crates/core/Cargo.toml`
- `crates/cosmwasm/api/Cargo.toml`
- `crates/cosmwasm/iface/Cargo.toml`
- `crates/cosmwasm/parse/Cargo.toml`
- `crates/cosmwasm/vanilla/Cargo.toml`
- `crates/storage/Cargo.toml`
- `xtask/Cargo.toml`

While enabled implicitly through the workspace manifest, future refactoring might break this assumption.

Recommendation

We recommend enabling overflow checks in all packages, including those that do not currently perform calculations, to prevent unintended consequences if changes are added in future releases or during refactoring. Note that enabling overflow checks in packages other than the workspace manifest will lead to compiler warnings.

Status: Acknowledged

The client states that it is highly unlikely that the crates will be taken out of the context of a workspace and lose the shared release profile. Maintaining this invariant would be preferable to dealing with persistent warnings about ignored release profiles.

12. Misleading comments in the codebase

Severity: Informational

The `hub` contract included detailed comments on the user who is expected to interact with some of the functions. Several of them detailed that the function should return an error if "The sender is not the vault creator or admin".

Although not a security issue, this is misleading for the current implementation because both the vault creator and the admin are the same address. Therefore, referencing the same role twice.

The affected comments can be found in lines `crates/core/hub/config.rs:336`, `362`, `395`, `423`, `451`, `479`, `507`, `535`, `581`, `609`, `637`, `665`, and `693`.

Similarly, the `allocate_dormant_state_surplus` function includes the following comment in `crates/core/hub/position.rs:743`: "Returns the total value of the Reserves payment, if any". This is incorrect as the function only allocated surplus to the treasury contract. Therefore, it only returns `Ok(None)`, and it is not possible to perform any reserve payment.

Recommendation

We recommend modifying the affected comments to reflect the actual implementation.

Status: Resolved