

# **Smart contract audit**

## **EthGild**

# Content

<b>Project description</b>	<b>3</b>
<b>Executive summary</b>	<b>3</b>
<b>Reviews</b>	<b>3</b>
<b>Technical analysis and findings</b>	<b>5</b>
<b>Security findings</b>	<b>6</b>
**** Critical	6
*** High	6
** Medium	6
* Low	6
Informational	6
I01 - Unused and Implicitly Visible sAuthorizee State Variable	6
I02 - Gas optimization	6
<b>General Risks</b>	<b>8</b>
<b>General Recommendations</b>	<b>9</b>
<b>Approach and methodology</b>	<b>10</b>



## Project description

This repository implements a decentralized financial system that leverages receipt-based vaults to manage tokenized assets. It provides mechanisms for minting ERC20 shares and ERC1155 receipt NFTs, enabling transparent tracking of deposits and withdrawals. The system supports dynamic share-to-asset ratios, off-chain asset integration, and price oracle-based calculations. It incorporates features like certification, asset confiscation, and fine-grained access control to ensure compliance and trust. The design emphasizes flexibility for issuers to maintain asset pegs and profitability while offering robust auditability and user protections.

## Executive summary

<b>Type</b>	Vault
<b>Languages</b>	Solidity
<b>Methods</b>	Architecture Review, Manual Review, Unit Testing, Functional Testing, Automated Review
<b>Documentation</b>	README.md
<b>Repository</b>	<a href="https://github.com/gildlab/ethgild">https://github.com/gildlab/ethgild</a>

## Reviews

<b>Date</b>	<b>Commit</b>
23/04/25	a86f472875258485fea3e512d5ec344ca4948cd4
28/05/25	a9901296f208b77c780b0cde4658ad1cf46bcd5c

## Scope

<b>Contracts</b>
src/concrete/receipt/Receipt.sol
src/concrete/receipt/ERC20PriceOracleReceipt.sol
rain.math.fixedpoint/lib/LibFixedPointDecimalArithmeticOpenZeppelin.sol



rain.math.fixedpoint/lib/format/LibFixedPointDecimalFormat.sol
src/concrete/vault/ERC20PriceOracleReceiptVault.sol
src/concrete/oracle/SceptreStakedFlrOracle.sol
src/concrete/oracle/TwoPriceOracleV2.sol
src/concrete/oracle/FtsoV2LTSFeedOracle.sol
rain.flare/lib/lts/LibFtsoV2LTS.sol
src/abstract/PriceOracleV2.sol
rain.flare/lib/sflr/LibSceptreStakedFlare.sol
src/abstract/ReceiptVault.sol
rain.factory/blob/main/src/concrete/CloneFactory.sol
rain.flare/blob/main/src/err/ErrFtso.sol
rain.flare/blob/main/src/lib/registry/LibFlareContractRegistry.sol
src/concrete/vault/OffchainAssetReceiptVault.sol
src/concrete/authorize/OffchainAssetReceiptVaultAuthorizerV1.sol
src/abstract/OwnerFreezable.sol



## Technical analysis and findings

Critical	0
High	0
Medium	0
Low	0
Informational	0

## Security findings

### \*\*\*\* Critical

No critical severity issue found.

### \*\*\* High

No high severity issue found.

### \*\* Medium

No Medium severity issue found.

### \* Low

No Low severity issue found.

## Informational

### I01 - Unused and Implicitly Visible sAuthorizee State Variable

The state variable sAuthorizee is declared in the OffchainAssetReceiptVaultAuthorizerV1 contract but is never used in the contract's logic. This introduces unnecessary complexity and may confuse developers or auditors reviewing the code.

Additionally, the variable has default visibility (internal), which can lead to ambiguity. Explicit visibility should be specified to improve code clarity and maintainability.

**Path:** src/concrete/authorize/OffchainAssetReceiptVaultAuthorizerV1.sol

**Recommendation:** If sAuthorizee is not required for the current or future functionality of the contract, consider removing it to simplify the code.

**Found in:** a86f472

**Status:** Fixed

### I02 - Gas optimization

Consider assigning the parameter certifyUntil to certifyStateChange.newCertifiedUntil instead of state variable sCertifiedUntil in the certify() function to avoid unnecessary multiple reads from storage.

**Path:** src/concrete/vault/OffchainAssetReceiptVault.sol : certify();



**Recommendation:** Replace assignment to local variable `certifyUntil`.

**Found in:** `a86f472`

**Status:** Fixed



## General Risks

- According to the logic of the `deposit()` function, each new depositor receives a new ID for every new deposit.  
At the same time, the `redeposit()` function allows issuing receipts with the same ID to different recipients. For example, both Alice and Bob could end up holding receipts with the same ID.
- The `receiptInformation()` function in the Receipt contract can be invoked at any time by any user. However, there is no specific event type associated with this function. This can lead to potential confusion when reading and interpreting events, as it may not be clear which function invocation triggered a particular event.
- The LibFtsoV2LTS library contains hardcoded feed IDs for various cryptocurrency/USD pairs. While these addresses are assumed to be correct, they may need to be updated if the project is deployed on another blockchain or if the feed addresses change. Relying on hardcoded addresses can lead to issues if the addresses are incorrect or need to be modified, as it requires changes to the source code and redeployment of the contract.
- In a few upgradeable contracts, the constructor is used to initialize immutable variables. Since the value of immutable variables is stored in the bytecode, it will be shared among all proxies pointing to a given contract instead of being stored in each proxy's storage.
- Most calculations in the project assume that tokens have 18 decimals. While this is a common standard, it is not universally true. Tokens with different decimal places may lead to incorrect calculations and potential vulnerabilities.
- The `confiscateShares` and `confiscateReceipt` functions allow confiscators to forcibly remove ERC20 shares and ERC1155 receipts from users. While these functions are designed to bypass transfer restrictions, they pose a significant risk to user assets if misused or exploited.
- The `ownerFreezeAlwaysAllowFrom()` and `ownerFreezeAlwaysAllowTo()` functions of the OwnerFreezable contract allow to set a timestamp (`protectUntil`) until which an address cannot be removed from the allowed list. The functions do not validate the maximum possible time that the owner can set. This can lead to the owner accidentally specifying a large number and the third-party address being added to the allowed list forever.








## General Recommendations

- To enhance the flexibility and granularity of access control, consider including information about whether a deposit is a redeposit or a regular deposit in the data sent to the authorizer. This would allow the authorizer to differentiate between the two scenarios and apply specific rules or restrictions as needed.

## Approach and methodology




To establish a uniform evaluation, we define the following terminology in accordance with the OWASP Risk Rating

Methodology:

	<b>Likelihood</b> Indicates the probability of a specific vulnerability being discovered and exploited in real-world scenarios
	<b>Impact</b> Measures the technical loss and business repercussions resulting from a successful attack
	<b>Severity</b> Reflects the comprehensive magnitude of the risk, combining both the probability of occurrence (likelihood) and the extent of potential consequences (impact)

Likelihood and impact are divided into three levels: High H, Medium M, and Low L. The severity of a risk is a blend of these two factors, leading to its classification into one of four tiers: Critical, High, Medium, or Low.

When we identify an issue, our approach may include deploying contracts on our private testnet for validation through testing. Where necessary, we might also create a Proof of Concept PoC to demonstrate potential exploitability. In particular, we perform the audit according to the following procedure:

	<b>Advanced DeFi Scrutiny</b> We further review business logics, examine system operations, and place DeFi-related aspects under scrutiny to uncover possible pitfalls and/or bugs
	<b>Semantic Consistency Checks</b> We then manually check the logic of implemented smart contracts and compare with the description in the white paper.
	<b>Security Analysis</b> The process begins with a comprehensive examination of the system to gain a deep understanding of its internal mechanisms, identifying any irregularities and potential weak spots.



