OpenZeppelin | security

Euler Vault Kit Synths Audit

June 14, 2024

Table of Contents

Table of Contents 2

Summary 3

Scope 4

System Overview 5

ESynth 5

IRMSynth 5

EulerSavingsRate 5

PegStabilityModule 6

Security Model and Trust Assumptions 6

Privileged Roles 6

Low Severity 7

L-01 Floating Pragma 7

L-02 Incomplete Docstrings 7

L-03 Missing Docstrings 8

L-04 EulerSavingsRate Contract Can Suffer From Reentrancy 8

L-05 The Rate In IRMSynth Contract Is Decreased by 9% and Not by 10% as Documented 9

L-06 The gulp Function Can Be Used to Push Forward the Interest Distribution Deadline Indefinitely 10

Notes & Additional Information 10

N-01 Function Visibility Overly Permissive 10

N-02 Lack of Security Contact 11

N-03 Inconsistent Use of Named Returns 11

N-04 Not Following Solidity Style Guide 12

N-05 Missing Named Parameters in Mapping 12

N-06 Non-Explicit Imports Are Used 13

N-07 Unused Named Return Variables 13

N-08 Gas and Code Improvements 14

N-09 Incorrect Docstrings 14

N-10 Removal of Addresses from totalSupply Calculation Has Unclear Purpose 14

N-11 Functions Are Updating the State Without Event Emissions 15

Client Reported 16

CR-01 Assets Can be Lost if gulp is Called With Empty Deposit 16

CR-02 Redeem and Withdraw Might Be Prevented by Enabled Controllers 16

Conclusion 17

Summary

Type	DeFi
Timeline	From 2024-05-13 To 2024-05-21
Languages	Solidity

Total Issues	19 (12 resolved, 1 partially resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	0 (0 resolved)
Low Severity Issues	6 (3 resolved)
Notes & Additional Information	11 (7 resolved, 1 partially resolved)
Client Reported Issues	2 (2 resolved)

Scope

We audited the contracts inside the `Synths` directory of the `euler-xyz/euler-vault-kit` repository at commit 82dd718.

In scope were the following files:

```
src
└── Synths
    ├── ERC20Collateral.sol
    ├── ESynth.sol
    ├── EulerSavingsRate.sol
    ├── IRMSynth.sol
    └── PegStabilityModule.sol
```

System Overview

The contracts located in the `Synths` directory implement logic for creating synthetic assets that follow the price of the underlying asset. In addition, a savings vault is implemented which is designed to accrue interest for the depositors.

`ESynth`

The `ESynth` contract is an ERC-20-compatible contract that integrates with EVC (Ethereum Vault Connector), allowing it to be used as collateral within the Euler ecosystem by executing account status checks on transfers and burns. The contract defines a mapping of minters, specifying the amount of tokens each can mint. The minters and their configurations can be added and modified by the owner.

The synthetic assets held by the synthetic asset contract itself can be allocated to or deallocated from the attached synthetic asset vault by the owner. Since protocol deposits into synthetic vaults are not backed by any collateral and are not in circulation, they are excluded from the total supply calculation. This is achieved within the allocate functionality which adds the vault to the list of ignored addresses or can be specified by the owner using a separate function.

`IRMSynth`

The `IRMSynth` contract is a custom interest rate model contract that is intended to be used by the synthetic asset vaults. It is designed to adjust the interest rate based on the current price of the asset relative to the target quote. If the synthetic asset trades above or at the target quote, the interest rate is decreased by 10% compared to the previous rate. If the synthetic asset trades below the target quote, the interest rate is increased by 10%.

`EulerSavingsRate`

The `EulerSavingsRate` contract is an ERC-4626-compatible vault that integrates with EVC, allowing the deposit of the underlying assets to receive interest. It can be used as collateral by

other vaults since all withdrawals, redemptions, and transfers execute account status checks using EVC to ensure the health of the account.

`PegStabilityModule`

The `PegStabilityModule` contract is responsible for supporting the peg of the `ESynth` token it integrates with. It has minting rights on the target `ESynth` contract and allows swaps from the underlying token to the synthetic token using a defined conversion price.

Security Model and Trust Assumptions

Privileged Roles

The owner defined in the `ESynth` contract can:

- set the minting capacity for a minter.
- deposit cash from the `ESynth` contract into the attached vault.
- withdraw cash from the attached vault to the `ESynth` contract.
- burn cash deposited into the `ESynth` contract.
- add an account to the list of accounts to ignore for the total supply.
- remove an account from the list of accounts to ignore for the total supply.

The minters configured by the owner of the `ESynth` contract can mint synthetic tokens up to the limit defined by the owner.

Low Severity

L-01 Floating Pragma

Pragma directives should be fixed to clearly identify the Solidity version with which the contracts will be compiled.

Throughout the codebase, there are multiple floating pragma directives:

- `ERC20Collateral.sol` has the `solidity ^0.8.0` floating pragma directive.
- `ESynth.sol` has the `solidity ^0.8.0` floating pragma directive.
- `EulerSavingsRate.sol` has the `solidity ^0.8.0` floating pragma directive.
- `IRMSynth.sol` has the `solidity ^0.8.0` floating pragma directive.
- `PegStabilityModule.sol` has the `solidity ^0.8.0` floating pragma directive.

Consider using fixed pragma directives.

Update: *Acknowledged, will resolve. The Euler team stated:*

> *Will be fixed pre-deployment.*

L-02 Incomplete Docstrings

Throughout the codebase, there are several instances of incomplete docstrings:

- Missing return value documented for the `isIgnoredForTotalSupply` function in `ESynth.sol`.
- Missing `owner` parameter documented for the `withdraw` function in `EulerSavingsRate.sol`.
- Missing `owner` parameter documented for the `redeem` function in `EulerSavingsRate.sol`.
- Missing return value documented for the `interestAccrued` function in `EulerSavingsRate.sol`.
- Missing return value documented for the `getESRSlot` function in `EulerSavingsRate.sol`.

Consider thoroughly documenting all functions/events (and their parameters or return values) that are part of a contract's public API. When writing docstrings, consider following the Ethereum Natural Specification Format (NatSpec).

Update: *Resolved in pull request #1.*

L-03 Missing Docstrings

Throughout the codebase, there are multiple code instances that do not have docstrings:

- The `minters` and `ignoredForTotalSupply` state variables, and the `MinterCapacitySet` event in `ESynth.sol`
- The state variables, the `maxRedeem` function, and the `maxWithdraw` function in `EulerSavingsRate.sol`
- The state variables, the `computeInterestRate` function, the `computeInterestRateView` function, and the `getIRMData` function in `IRMSynth.sol`
- The state variables in `PegStabilityModule.sol`

Consider thoroughly documenting all functions (and their parameters) that are part of any contract's public API. Functions implementing sensitive functionality, even if not public, should be clearly documented as well. When writing docstrings, consider following the Ethereum Natural Specification Format (NatSpec).

Update: *Resolved in pull request #14.*

L-04 `EulerSavingsRate` Contract Can Suffer From Reentrancy

In the `EulerSavingsRate` contract, the `_deposit` and `_withdraw` function overrides of OpenZeppelin's ERC-4626 implementation manually account for the `_totalAssets` internal variable before calling their `super` counterparts. According to the counterparts' documentation, state changes must occur after a `transferFrom` and before a `transfer` when dealing with ERC-777 tokens. The `transferFrom` operations are triggered by deposits, while transfers are triggered by withdrawals.

In general, any hook-enabled token used with the `EulerSavingsRate` contract should be handled carefully as it is uncertain whether state changes must be made before and/or after any potential trigger of a hook. This is because a malicious actor can reenter the contract and

exploit recursive calls, leaving the contract in a compromised state. While the current implementation uses the `nonReentrant` modifier to prevent reentrancy attacks, it is still vulnerable to read-only reentrancy, where the call to the `totalAssets` function returns the value of assets as if the deposit has occurred, whereas in reality the assets have not been transferred yet.

Consider either thoroughly documenting that the contract is not capable of handling ERC-777 tokens or updating the implementation to handle such tokens. Keep in mind that updating the total assets amount after transferring tokens during deposit might expose the contract to additional side effects, like the `convertToShare` function not using the proper `totalSupply`.

Update: *Resolved in pull request #13. The Euler team stated:*

> *Fixed while also fixing CR-02.*

L-05 The Rate In `IRMSynth` Contract Is Decreased by 9% and Not by 10% as Documented

The `_computeRate` function in the `IRMSynth` contract is designed to adjust the rate based on the comparison between the current quote and the target quote. Specifically, the function should increase the rate by 10% if the current quote is less than the target quote and decrease the rate if the current quote is equal to or bigger than the target quote. However, there is an issue with the decrease calculation: it multiplies the rate by `1.0e18 / 1.1e18`, leading to an approximate 9% decrease instead of the intended 10% adjustment.

Consider updating the documentation to thoroughly describe the calculations made.

Update: *Acknowledged, not resolved. The Euler team stated:*

> *This is intended behaviour. The whitepaper is referring to a (scaled) change in log-percentage. When discussing relative changes in prices or rates, this is often implicit because of the assumption these values follow geometric Brownian motion. Note that when computed in this way, the relative changes are symmetric.*

L-06 The `gulp` Function Can Be Used to Push Forward the Interest Distribution Deadline Indefinitely

The `gulp` function allows spreading donations to the vault as interest. This is done by updating `interestSmearEnd` to a time of two weeks from now and increasing the value of `interestLeft` by the amount of the donation. The issue is that the function can be called even when there has been no donation, resulting in repeated updates to `interestSmearEnd` and modification of the deadline by two weeks, even when there is nothing to smear.

The `documentation` states:

> On gulp any interest which has not been distributed is smeared for an additional two weeks, in theory this means that interest could be smeared indefinitely by continuously calling gulp, in practice it is expected that the interest will keep accruing, negating any negative side effects which may come from the smearing mechanism.

However, this is only partially true. While it is true that interest will keep accruing, it is also true that any `interestLeft` that has not been accrued yet would be spread over two weeks even if the time left to accrue it at 100% is less than this period.

Consider documenting that as long as such an attack is carried out, undistributed interest is indeed affected (even if not indefinitely).

Update: *Acknowledged, will resolve. The Euler team stated:*

> *Will clarify docs pre-deployment.*

Notes & Additional Information

N-01 Function Visibility Overly Permissive

Throughout the codebase, there are various functions with unnecessarily permissive visibility:

- The `isIgnoredForTotalSupply` function in `ESynth.sol` with `PUBLIC` visibility could be limited to `EXTERNAL`.

- The `getAllIgnoredForTotalSupply` function in `ESynth.sol` with `PUBLIC` visibility could be limited to `EXTERNAL`.

To better convey the intended use of functions and to potentially realize some additional gas savings, consider changing a function's visibility to be only as permissive as required.

Update: *Resolved in pull request #5.*

N-02 Lack of Security Contact

Providing a specific security contact (such as an email or ENS name) within a smart contract significantly simplifies the process for individuals to communicate if they identify a vulnerability in the code. This practice is quite beneficial as it permits the code owners to dictate the communication channel for vulnerability disclosure, eliminating the risk of miscommunication or failure to report due to a lack of knowledge on how to do so. In addition, if the contract incorporates third-party libraries and a bug surfaces in those, it becomes easier for their maintainers to contact the appropriate person about the problem and provide mitigation instructions.

Throughout the codebase, there are contracts that do not have a security contact:

- The `ERC20Collateral` abstract contract.
- The `ESynth` contract.
- The `EulerSavingsRate` contract.
- The `IRMSynth` contract.
- The `PegStabilityModule` contract.

Consider adding a NatSpec comment containing a security contact above each contract definition. Using the `@custom:security-contact` convention is recommended as it has been adopted by the OpenZeppelin Wizard and the ethereum-lists.

Update: *Resolved at commit 34b1716.*

N-03 Inconsistent Use of Named Returns

To improve the readability of the contract, use the same return style in all functions. Throughout the codebase, there are multiple instances where contracts have inconsistent usage of named returns in their functions:

- In `ESynth` contract of `ESynth.sol`

- In `IRMSynth` contract of `IRMSynth.sol`

Consider being consistent with the use of named returns throughout the codebase.

Update: *Resolved in pull request #6.*

N-04 Not Following Solidity Style Guide

Throughout the codebase, there are multiple contracts that deviate from the Solidity Style Guide.

Some contracts have inconsistent ordering of functions:

- The `ESynth` contract in `ESynth.sol`
- The `EulerSavingsRate` contract in `EulerSavingsRate.sol`
- The `IRMSynth` contract in `IRMSynth.sol`

Moreover:

- Internal variables like the `ignoredForTotalSupply` variable of the `ESynth` contract or `irmStorage` variable of the `IRMSynth` contract have no prefix `_`.
- All the immutable variables in the `IRMSynth` contract and in the `PegStabilityModule` should be named using the UPPER_CASE format.

To improve the project's overall legibility, consider standardizing the ordering throughout the codebase and following the variable naming system that is recommended by the Solidity Style Guide (order of functions, prefix of internal variables, constant variables).

Update: *Acknowledged, not resolved. The Euler team stated:*

> *We will keep it as it is to not introduce a large difference from previously audited code.*

N-05 Missing Named Parameters in Mapping

Since Solidity 0.8.18, developers can utilize named parameters in mappings. This means mappings can take the form of `mapping(KeyType KeyName? => ValueType ValueName?)`. This updated syntax provides a more transparent representation of a mapping's purpose.

The `minters` state variable in the `ESynth` contract does not have named parameters.

Consider adding named parameters to the mappings to improve code clarity and readability.

Update: *Resolved in pull request #7.*

N-06 Non-Explicit Imports Are Used

The use of non-explicit imports in the codebase can decrease code clarity and may create naming conflicts between locally defined and imported variables. This is particularly relevant when multiple contracts exist within the same Solidity file or when inheritance chains are long.

Within `IRMSynth.sol`, global imports are being used.

Following the principle that clearer code is better code, consider using the named import syntax (`import {A, B, C} from "X"`) to explicitly declare which contracts are being imported.

Update: *Resolved in pull request #8.*

N-07 Unused Named Return Variables

Named return variables are a way to declare variables that are meant to be used within a function's body for the purpose of being returned as that function's output. They are an alternative to explicit in-line `return` statements.

Within `ESynth.sol`, there are unused named return variables:

- The `success` return variable in the `addIgnoredForTotalSupply` function
- The `success` return variable in the `removeIgnoredForTotalSupply` function

Consider either using or removing any unused named return variables.

Update: *Acknowledged, not resolved. The Euler team stated:*

> *We will keep it as it is.*

N-08 Gas and Code Improvements

There are instances in the codebase that might benefit from a refactor to improve gas consumption and overall code quality:

- In line 59 of the `IRMSynth` contract, `irmCache` is defined locally and then passed to the `_computeRate` function, unlike what is done in the `computeInterestRateView` function. Consider avoiding the definition of the `irmCache` local variable.
- In the `PegStabilityModule` contract, the check for `amountIn` and `amountOut` not being zero is repeated in all swap functions. Consider defining a single location in the code for this check to avoid code repetition. The same issue occurs with the check for the length of the controllers set in the `EulerSavingsRate` contract.
- The `for` loop in the `ESynth` contract could be more efficient if `++i` is used instead of `i++`. This optimization skips storing the value before the increment operation as the return value of the expression is ignored.

Consider addressing the above listed instances.

Update: *Partially resolved in pull request #9. The Euler team stated:*

> *Partial fix, we kept the code repetition for legibility.*

N-09 Incorrect Docstrings

In line 129 of the `EulerSavingsRate` contract, the docstring says "spend" but it should be "spent" instead. Moreover, the docstrings of the `withdraw` function are the same as those of the `deposit` function.

In order to improve the correctness of the docstrings, consider addressing the mentioned incorrect instances.

Update: *Resolved in pull request #10 and in pull request #1.*

N-10 Removal of Addresses from `totalSupply` Calculation Has Unclear Purpose

The documentation regarding the removal of synthetics vaults from the accounting of `totalSupply` specifies that this is done because deposits into synthetics vaults are not

backed by any collateral. However, this is true only if the owner of the `ESynth` contract is allocating (i.e., depositing) tokens that have been minted directly.

Any donor can mint tokens through the `PegStabilityModule` contract, transfer those tokens into the `ESynth` contract, and once they are allocated, the assumption that such a deposit is not backed by any collateral becomes incorrect. Therefore, while there are alternative ways to mint synthetic assets, it is not always the case that they lack backing once minted.

Consider fixing the documentation and/or the feature itself to avoid having collateral-backed tokens which are not accounted for in the `totalSupply`.

Update: *Acknowledged, will resolve. The Euler team stated:*

> *We will fix it in docs post-contest.*

N-11 Functions Are Updating the State Without Event Emissions

Throughout the codebase, instances of functions that are updating the state without an event emission were found:

- The `gulp` function in `EulerSavingsRate.sol`
- The `updateInterestAndReturnESRSlotCache` function in `EulerSavingsRate.sol`
- The `computeInterestRate` function in `IRMSynth.sol`

Consider emitting events whenever there are state changes to make the codebase less error-prone and improve its readability.

Update: *Resolved in pull request #11.*

Client Reported

CR-01 Assets Can be Lost if `gulp` is Called With Empty Deposit

When the `gulp` function is called on `EulerSavingsRate` and there's no deposits it essentially gets lost. This is because the `totalAssets` function will be positive even if no deposits have been made, making the virtual shares to accrue those matured interests.

The Euler team consider that this is not really something that would happen in practice but they will fix it by returning early if the `totalSupply` is below some threshold.

Update: *Resolved in pull request #12. The Euler team stated:*

> *Fixed by requiring the total amount of shares to be at least 10x the `VIRTUAL_AMOUNT`*

CR-02 Redeem and Withdraw Might Be Prevented by Enabled Controllers

The `maxRedeem` and `maxWithdraw` in the `EulerSavingsRate` contract returns zero whenever a controller is set. However in the `withdraw` and `redeem` functions of original the openzeppelin ERC-4626 implementation those are used. This would result in a user not being able to withdraw or redeem whenever an user has a controller enabled on the EVC no matter if and how much liabilities are actually present.

The Euler team consider this not a major vulnerability but an inconvenient behaviour for an user. The proposed fix would be to overwrite withdraw and redeem to not call these functions and just process the withdraw.

Update: *Resolved in pull request #13. The Euler team stated:*

> *Fixed by overriding `redeem` and `withdraw`.*

Conclusion

The audited codebase comprises logic pertaining to creating synthetic assets and an interest-accruing savings vault. The audit was conducted over the course of one week and did not reveal any significant issues. Various recommendations have been made to enhance the quality of the codebase. The well-implemented test suite, along with the fuzzing implementation, showed that the codebase is very mature. The Euler team was very supportive in answering questions in a timely manner.