PUBLIC

Code Assessment

of the Unslashed-Enzyme Bridge Smart Contract

May 12, 2021

Produced for

by

Contents

1 Executive Summary

Dear Mona and Marh,

First and foremost we would like to thank Avantgarde Finance for giving us the opportunity to assess the current state of their Unslashed-Enzyme Bridge system. This document outlines the findings, limitations, and methodology of our assessment.

The smart contract reviewed implements a bridge between Unslashed and Enzyme. It provides basic bridging functionality without enforcing any additional restrictions or checks. The functionality works as expected if the bridge connects to a properly configured fund and all trusted roles execute their actions correctly. Although these assumptions are documented within the codebase we highly recommend documenting all assumptions/prerequisites on the fund's configuration more thoroughly in the documentation to avoid any potential issue during operation.

We hope that this assessment provides more insight into the current implementation and provides valuable findings. We are happy to receive questions and feedback to improve our service and are highly committed to further support your project.

Sincerely yours,

ChainSecurity

1.1 Overview of the Findings

Below we provide a brief numerical overview of the findings and how they have been addressed.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	5
• Code Corrected	2
• Specification Changed	1
• Risk Accepted	1
• Acknowledged	1

2 Assessment Overview

In this section we briefly describe the overall structure and scope of the engagement including the code commit which is referenced throughout this report.

2.1 Scope

The assessment was performed on the source code file `EnzymeBridge.sol` based on the documentation files. The table below indicates the code versions relevant to this report and when they were received.

V	Date	Commit Hash	Note
1	22 April 2021	1c0dfad2b90d6f3329a1aac6388874a12e9d1b93	Initial Version
2	2 May 2021	98112535d05a4ef0acf4019c938079803d705698	After intermediate report

For the solidity smart contracts, the compiler version `0.6.6` was chosen.

The contract under review connects to an external system called Enzyme. The behavior of the external system may change in the future. For the purpose of the audit the Enzyme implementation as of April 2021 was considered.

2.2 System Overview

This system overview describes the initially received version (Version 1) of the contracts as defined in the Assessment Overview.

The smart contract reviewed implements a bridge between Unslashed and Enzyme. Enzyme is an on-chain asset management system supporting interactions with all major DeFi applications. Unslashed is a decentralized insurance protocol supporting many different markets. Furthermore, it allows for multiple markets to be bundled in a basket. This enables users to provide collateral for the whole basket instead of individual markets. The basket contract is the main entry point for users. In order to avoid having the full collateral being unused in the basket contract, the admin of the basket may invest a part of the available Ether via the Enzyme Protocol. In order to do so, a basket contract can feature functionality to connect to the Enzyme bridge. If this feature is supported only the admin of the basket can deposit/withdraw Ether to and from the Enzyme fund. When depositing into an Enzyme fund, shares are minted to the depositor which represents his share of the fund. These non-transferable shares will be held by the Enzyme Bridge Proxy contract, thus each basket must deploy its own Enzyme Bridge Proxy contract. The Enzyme Fund must be uniquely used by this Basket (via the Enzyme Bridge) only, no other address may invest or hold a share of the fund. The code of the `Basket` contract assumes the full gross asset value of the fund belongs to the basket.

The Enzyme bridge exposes three functionalities:

- `depositEthFromInvestor`: The admin transfers Ether to an Enzyme fund and receives shares.
- `withdrawEthToInvestor`: The admin gives back their shares and receives Ether.
- `getBalanceInEth`: Retrieves the gross asset value of the fund.

2.3 Trust Model & Roles

The Enzyme Bridge simply provides the connection between an Unslashed Basket contract and an Enzyme fund. It does not provide any guarantees or restrictions on what the fund's configuration can be or what is allowed to do with the funds under management. The Enzyme System itself features a set of policies which can be enabled in order to control the risk and exposure. It's a requirement that the `InvestorWhitelist` policy is active with the Enzyme Bridge set as the only whitelisted address allowed to buy shares. The denomination asset of the fund must always be `WETH`. No further information has been given about the planned policy configuration.

The trust model assumes that this fund is fully controlled by an honest administrator of Unslashed who is assumed to do favorable actions (e.g. lendings) only and comply with all requirements of the basket. Notably, this includes that he always exchanges everything back into `WETH` if required so that the Admin of the Unslashed basket can successfully retrieve the funds using `withdrawEthToInvestor()`.

Hence there are three separate admin roles, all controlled by Unslashed and all are fully trusted:

- The admin of the `Basket` contract
- The admin of the Enzyme Fund belonging to this basket
- The admin of the `Proxy`

All state changing functions of the bridge can only be triggered by the `Investor` role which is the `Basket` contract using this bridge. A basket contract in the Unslashed system consists of one or multiple insurance markets. Users interact with Unslashed through these basket contracts.

Our understanding is that the Enzyme Fund is only used to lend Ether in order to build interest.

3 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts associated with the items defined in the engagement letter on whether it is used in accordance with its specifications by the user meeting the criteria predefined in the business specification. We draw attention to the fact that due to inherent limitations in any software development process and software product an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third party infrastructure which adds further inherent risks as we rely on the correct execution of the included third party technology stack itself. Report readers should also take into account the facts that over the life cycle of any software product changes to the product itself or to its environment, in which it is operated, can have an impact leading to operational behaviours other than initially determined in the business specification.

4 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- *Likelihood* represents the likelihood of a finding to be triggered or exploited in practice
- *Impact* specifies the technical and business-related consequences of a finding
- *Severity* is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severities. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

Likelihood	Impact		
	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

5 Findings

In this section, we describe any open findings. Findings that have been resolved, have been moved to the Resolved Findings section. All of the findings are split into these different categories:

- Design: Architectural shortcomings and design inefficiencies
- Correctness: Mismatches between specification and implementation

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	2

- Outdated Compiler Version Risk Accepted
- _weth Could Be a Constant Acknowledged

5.1 Outdated Compiler Version

Design Low Version 1 Risk Accepted

The project uses an outdated version of the Solidity compiler.

```
pragma solidity 0.6.6;
```

Known bugs in version 0.6.6 are: https://github.com/ethereum/solidity/blob/develop/docs/bugs_by_version.json#L1416

More information about these bugs can be found here: https://docs.soliditylang.org/en/latest/bugs.html

At the time of writing the most recent Solidity release is version `0.6.12`. For version `0.6.x` the most recent release is `0.6.12` which contains some bugfixes but no breaking changes.

The compiler was not changed as the client responded:

`We will address this later for an overall code review.`

5.2 _weth Could Be a Constant

Design Low Version 1 Acknowledged

The constant `_weth` which represents the same address across all Enzyme Bridges could be set as constant. This would reduce unnessesary storage operations.

Acknowledged:

Avantgarde Finance is aware of this and commented that it does have a significant impact as the variable will be accessed a few times only.

6 Resolved Findings

Here, we list findings that have been resolved during the course of the engagement. Their categories are explained in the Findings section.

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	3

- Denomination Asset Check on Initialization Code Corrected
- Redeem Shares Return Value Not Used Code Corrected
- withdrawEthToInvestor Specification Discrepancy Specification Changed

6.1 Denomination Asset Check on Initialization

Design Low Version 1 Code Corrected

For the correct operation of the bridge, it is critical that the denomination asset of the vault is correctly set to WETH during initialization. Such a check is not present in the code.

Code Changed:

A check has been introduced in the `initialize` function

```
require(IComptrollerProxy(controllerProxy).getDenominationAsset() == weth, "EnzymeBridge: wrong fund asset");
```

6.2 Redeem Shares Return Value Not Used

Design Low Version 1 Code Corrected

The return values of

```
IComptrollerProxy(_controllerProxy).redeemShares()
```

and

```
IComptrollerProxy(_controllerProxy).redeemSharesDetailed(...)
```

are not used. Both functions return the payout assets and amounts. Instead, the resulting amount of tokens is checked using *balanceOf(...)*, incurring additional gas costs. The benefit of the current implementation is the fact that one can withdraw tokens that were already in the vault before the redemption. However, this choice is inconsistent with the later choice to only send the ether which was recently unwrapped.

```
uint256 result = IWETH9(_weth).balanceOf(address(this));
IWETH9(_weth).withdraw(result); // returns eth to us
IInvestable(_investor).receiveEthFromFund{value: result}();
```

Code Corrected:

The code has been changed so it uses the return value of `redeemShares` and `redeemSharesDetailed`. Moreover, the returned result is used to ensure that only one returned asset is used. Finally, the code now withdraws a consistent amount of WETH and ETH. of

```
(, payoutAmounts) = IComptrollerProxy(_comptrollerProxy).redeemShares();

....

(, payoutAmounts) = IComptrollerProxy(_comptrollerProxy).redeemSharesDetailed(sharesQuantity, empty, empty);
...

require(payoutAmounts.length == 1, "EnzymeBridge: fund not converted");
...

uint256 result = payoutAmounts[0];
IWETH9(_weth).withdraw(result); // returns eth to us
IInvestable(_investor).receiveEthFromFund{value: result}();
```

6.3 `withdrawEthToInvestor` Specification Discrepancy

Correctness **Low** **Version 1** **Specification Changed**

The comment for `withdrawEthToInvestor` reads as follows: `Note that due to possible share value rounding the resulting amount may be slightly greater than requested.`

Another rounding error may happen when the shares' quantity required is calculated:

```
uint256 sharesQuantity = amount.mul(PRECISION_18E).div(shareValue18ePrecision);
```

Consider the case when the share value is not affected by rounding errors. Due to possible rounding errors when the shares' quantity is calculated, the shares' quantity required to receive the `amount` in `WETH` may be underestimated. Hence the Ether amount withdrawn might be slightly smaller and thus the comment does not hold.

Specification Changed:

The documentation correctly now states:

```
/// Note that due to possible share value or division rounding
/// the resulting amount may be slightly greater or smaller than requested.
```

7 Notes

We leverage this section to highlight further findings that are not necessarily issues. The bridge contract in scope for this review connects Unslashed to Enzyme which consists of many interacting contracts. Hence, the mentioned topics serve to clarify or support the report, but do not require a modification inside the project. Instead, they should raise awareness in order to improve the overall understanding for users and developers

7.1 Illiquid Lending Providers

Note Version 1

The basket can only withdraw when all assets of the fund have been exchanged into `WETH`. Illiquid liquidity protocols may be unable to redeem a large amount of tokens at times. Hence a fund manager may be unable to redeem all assets into their underlying.

E.g. a large amount of WETH has been lent into the Aave liquidity pool. When the fund manager wants to redeem this large amount of derivative tokens back into the underlying `WETH`, it could be the case that the liquidity may be insufficient as currently a large amount of `WETH` is lent out.

During this period Ether withdrawal is blocked.

7.2 Migration to New Enzyme Release Is Not Supported

Note Version 1

This Enzyme Bridge in connection with the current implementation of the `Basket` contract does not support the migration of a fund to a new release of Enzyme.

When a fund is upgraded in Enzyme, the `ComptrollerProxy` is replaced by a new one while the old one is selfdestructed. The `VaultProxy` holding the funds remains.

After such a migration all calls from the bridge to the comptroller will fail as the contract no longer exists. This affects all three functions (`deposit` / `withdraw` / `getBalance`).

Note that function `setFund()` of the basket contract cannot be used to recover from this situation: This function requires the fund's balance to be 0. The call to the non-existing comptroller will revert the transaction.

All shares should be redeemed before a migration is initiated. Shares remaining after a migration might be stuck with the current code.

After looking at the migration scripts present, we understand that the EnzymeBridge is to be used via a Proxy. This would allow to upgrade the implementation and recover the shares.

7.3 No Fees

Note Version 1

Funds of Enzyme can configure fees investors have to pay in order to participate in a fund. These fees are paid to the fund manager in the form of shares. Note that the implementation of the Enzyme Bridge relies on the fact that the basket is the only shareholder of the fund and no other address holds such shares. Hence enabled fees for the fund are not compatible with the bridge, funds of the bridge must not have fees configured.