PUBLIC

Code Assessment

of the Deposit Wrapper Fix Smart Contracts

October 12, 2023

Produced for

enzyme

by

CHAINSECURITY

Contents

1 Executive Summary

Dear Enzyme team,

On October 7th, 2023 an issue was reported through the Immunify platform regarding Enzyme's Deposit Wrapper peripheral contract.

The ChainSecurity team responded immediately and supported the Enzyme team in understanding and mitigating the issue. Furthermore, the following day a team in ChainSecurity further investigated the issue and possible relevant issues and reviewed the proposed fix.

From our investigation, no further issues were uncovered.

It is important to note that security audits are time-boxed and cannot uncover all vulnerabilities. They complement but don't replace other vital measures to secure a project.

The following sections will give an overview of the system, our methodology, the issues uncovered, and how they have been addressed. We are happy to receive questions and feedback to improve our service.

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	1
• Code Corrected	1
Medium-Severity Findings	0
Low-Severity Findings	0

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 files inside the Deposit Wrapper Fix repository 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	11 Oct 2023	3f40c0b1e652a57ced282b297a9e4c02334fa3b3	Deposit Wrapper fix

For the solidity smart contracts, the compiler version `0.8.19` was chosen.

The smart contracts in scope are:

```
contracts/release/peripheral/DepositWrapper.sol
```

2.1.1 Excluded from Scope

All the contracts that are not explicitly mentioned in the Scope section are excluded from the scope. The external systems, with which the Sulu interacts, are assumed to work correctly. Moreover, all the libraries used such as the OpenZeppelin library are assumed to work correctly and not in the scope of this review. Finally, attack vectors, such as unfavorable for the fund trades, employed by the managers of the funds have not been considered as the managers are considered trusted by the system.

2.2 System Overview

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

Furthermore, in the findings section, we have added a version icon to each of the findings to increase the readability of the report.

This assessment is only concerned with specific code parts of the system. Here, we only provide a brief overview of the newly introduced modules of the system. Please refer to Enzyme's documentation or our previous reports for a more detailed overview of the system.

Enzyme is an investment management system that allows one to buy shares of certain vaults. Users can invest in the fund by either directly interacting with the Comptroller of the fund or through a deposit wrapper. The Deposit Wrapper exposes currently two functionalities:

- `exchangeErc20AndBuyShares`: transfers an arbitrary asset from the user to the wrapper and then, using a decentralized exchange, converts it to the denomination asset of a fund. It returns the remaining funds to the user.
- `exchangeEthAndBuyShares`: wraps ETH sent from a user to WETH, converts it to the denomination of a fund, and uses this amount to buy shares of the fund. It unwraps and returns the remaining assets to the user.

The caller can specify the exchange they want to use and the exact parameters of that call to the exchange. The exchange should be whitelisted. The DepositWrapper is assumed to never hold assets at the end of a transaction.

2.2.1 Roles and Trust Model

We assume the token and exchanged used in the wrapper are not the same address and that the wrapper will never own funds after a call is finished. Generally, the roles and the trust model remain the same as described in ChainSecurity's security report of the system. It is assumed that the whitelisted exchanges are not ERC20 tokens themselves.

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 the 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 defined in the engagement letter. We assessed whether the project follows the provided specifications. These assessments are based on the provided threat model and trust assumptions. 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 that over the life cycle of any software, changes to the product itself or to the environment in which it is operated can have an impact leading to operational behaviors other than those 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 severity. 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. The findings are split into these different categories:

- Security: Related to vulnerabilities that could be exploited by malicious actors

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	0

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	1

- Unsanitized _exchange Address Code Corrected

Medium-Severity Findings	0

Low-Severity Findings	0

6.1 Unsanitized _exchange Address

Security High Version 1 Code Corrected

CS-SULDW-001

The `_exchange` address is not sanitized in `exchangeErc20AndBuyShares`. This makes the following attack possible.

- A user with address `victim` who wants to use `exchangeErc20AndBuyShares()` must first grant approval on the DepositWrapper for the token they want to convert to the denomination asset of they want to buy shares. Let's call this token TargetToken.
- An attacker with address `attacker`, who sees this approval, front-runs the call of `exchangeErc20AndBuyShares()` by calling `exchangeErc20AndBuyShares()`. As the `_exchange` parameter is not sanitized the attacker can set `_exchange` address to be TargetToken and `_exchangeData` to be `safeTransferFrom(victim, attacker)`.
- Since the DepositWrapper has been approved by the `victim`, it is able to transfer the assets successfully.

Code Corrected:

The `_exchange` parameter is checked to belong to a list of whitelisted addresses specified by Enzyme Council. Note that the exchange should not be an ERC20 token. In such a case the attack is still possible, should a user approve the `DepositWrapper` for that token. Also, note that the `_exchangeApproveTarget` is still unsanitized. However, no potential issue was uncovered regarding having the `DepositWrapper` approve arbitrary addresses. For a better security practice, Avantgarde Finance may want to consider whitelisting this address.