CredShields
Smart Contract Audit

Sept 14th, 2023 • CONFIDENTIAL

Description

This document details the process and result of the ZetaSwap Smart Contracts audit performed by CredShields Technologies PTE. LTD. on behalf of Eddy Finance between Sept 10th, 2023, and Sept 13th, 2023. A retest was performed on Sept 14th, 2023.

Author

Shashank (Co-founder, CredShields)

shashank@CredShields.com

Reviewers

Aditya Dixit (Research Team Lead)

aditya@CredShields.com

Prepared for

Eddy finance

Table of Contents

1. Executive Summary

Eddy Finance engaged CredShields to perform a smart contract audit from Sept 10th, 2023, to Sept 13th, 2023. During this timeframe, Five (5) vulnerabilities were identified. **A retest was performed on Sept 14th, 2023, and all the Critical and High bugs have been addressed while 4 non-critical and high severity bugs remain unfixed.**

During the audit, One (1) vulnerability was found with a severity rating of either High or Critical. These vulnerabilities represent the greatest immediate risk to "Eddy Finance" and should be prioritized for remediation, and fortunately, none were found.

The table below shows the in-scope assets and a breakdown of findings by severity per asset. Section 2.3 contains more information on how severity is calculated.

Assets in Scope	Critical	High	Medium	Low	info	Gas	Σ
ZetaSwap Smart Contracts	1	0	1	3	0	0	**5**
	1	**0**	**1**	**1**	**0**	**0**	**5**

Table: Vulnerabilities Per Asset in Scope

The CredShields team conducted the security audit to focus on identifying vulnerabilities in ZetaSwap Smart Contracts's scope during the testing window while abiding by the policies set forth by ZetaSwap's team.

State of Security

To maintain a robust security posture, it is essential to continuously review and improve upon current security processes. Utilizing CredShields' continuous audit feature allows both Eddy Finance's internal security and development teams to not only identify specific vulnerabilities but also gain a deeper understanding of the current security threat landscape.

To ensure that vulnerabilities are not introduced when new features are added, or code is refactored, we recommend conducting regular security assessments. Additionally, by analyzing the root cause of resolved vulnerabilities, the internal teams at Eddy Finance can implement both manual and automated procedures to eliminate entire classes of vulnerabilities in the future. By taking a proactive approach, Eddy Finance can future-proof its security posture and protect its assets.

2. Methodology

Eddy Finance engaged CredShields to perform the ZetaSwap Smart Contract audit. The following sections cover how the engagement was put together and executed.

2.1 Preparation phase

The CredShields team meticulously reviewed all provided documents and comments in the smart contract code to gain a thorough understanding of the contract's features and functionalities. They meticulously examined all functions and created a mind map to systematically identify potential security vulnerabilities, prioritizing those that were more critical and business-sensitive for the refactored code. To confirm their findings, the team deployed a self-hosted version of the smart contract and performed verifications and validations during the audit phase.

A testing window from Sept 10th, 2023, to Sept 13th, 2023, was agreed upon during the preparation phase.

2.1.1 Scope

During the preparation phase, the following scope for the engagement was agreed-upon:

IN SCOPE ASSETS
https://github.com/EddyFinance/omni-chain-contracts/blob/cb8b9936273572224dceef6945b74471e9df6770/contracts/ZetaSwapV2.sol

Table: List of Files in Scope

2.1.2 Documentation

Documentation was not required as the code was self-sufficient for understanding the project.

2.1.3 Audit Goals

CredShields uses both in-house tools and manual methods for comprehensive smart contract security auditing. The majority of the audit is done by manually reviewing the contract source code, following SWC registry standards, and an extended industry standard self-developed checklist. The team places emphasis on understanding core concepts, preparing test cases, and evaluating business logic for potential vulnerabilities.

2.2 Retesting phase

Eddy Finance is actively partnering with CredShields to validate the remediations implemented towards the discovered vulnerabilities.

2.3 Vulnerability classification and severity

CredShields follows OWASP's Risk Rating Methodology to determine the risk associated with discovered vulnerabilities. This approach considers two factors - Likelihood and Impact - which are evaluated with three possible values - **Low**, **Medium**, and **High**, based on factors such as Threat agents, Vulnerability factors, Technical and Business Impacts. The overall severity of the risk is calculated by combining the likelihood and impact estimates.

Overall Risk Severity				
Impact	HIGH	Medium	High	Critical
	MEDIUM	Low	Medium	High
	LOW	Note	Low	Medium
		LOW	MEDIUM	HIGH
	Likelihood			

Overall, the categories can be defined as described below -

1. Informational

We prioritize technical excellence and pay attention to detail in our coding practices. Our guidelines, standards, and best practices help ensure software stability and reliability. Informational vulnerabilities are opportunities for improvement and do not pose a direct risk to the contract. Code maintainers should use their own judgment on whether to address them.

2. **Low**

   Low-risk vulnerabilities are those that either have a small impact or can't be exploited repeatedly or those the client considers insignificant based on their specific business circumstances.

3. **Medium**

   Medium-severity vulnerabilities are those caused by weak or flawed logic in the code and can lead to exfiltration or modification of private user information. These vulnerabilities can harm the client's reputation under certain conditions and should be fixed within a specified timeframe.

4. **High**

   High-severity vulnerabilities pose a significant risk to the Smart Contract and the organization. They can result in the loss of funds for some users, may or may not require specific conditions, and are more complex to exploit. These vulnerabilities can harm the client's reputation and should be fixed immediately.

5. **Critical**

   Critical issues are directly exploitable bugs or security vulnerabilities that do not require specific conditions. They often result in the loss of funds and Ether from Smart Contracts or users and put sensitive user information at risk of compromise or modification. The client's reputation and financial stability will be severely impacted if these issues are not addressed immediately.

6. **Gas**

   To address the risk and volatility of smart contracts and the use of gas as a method of payment, CredShields has introduced a "Gas" severity category. This category deals with optimizing code and refactoring to conserve gas.

2.4 CredShields staff

The following individual at CredShields managed this engagement and produced this report:

- **Shashank, Co-founder CredShields**
    - shashank@CredShields.com

Please feel free to contact this individual with any questions or concerns you have about the engagement or this document.

3. Findings

This chapter contains the results of the security assessment. Findings are sorted by their severity and grouped by the asset and SWC classification. Each asset section will include a summary. The table in the executive summary contains the total number of identified security vulnerabilities per asset per risk indication.

3.1 Findings Overview

3.1.1 Vulnerability Summary

During the security assessment, Five (5) security vulnerabilities were identified in the asset.

VULNERABILITY TITLE	SEVERITY	SWC | Vulnerability Type
Missing Access Control in onCrossChainCall	Critical	Missing Access Control
Zero Slippage value	Medium	Slippage Risk
Hardcoded Deadline	Low	Business Logic
Missing Validation in Tokens	Low	Missing Input Validation
Outdated Pragma version	Low	Outdated Pragma

Table: Findings in Smart Contracts

3.1.2 Findings Summary

SWC ID	SWC Checklist	Test Result	Notes
SWC-100	Function Default Visibility	Not Vulnerable	Not applicable after v0.5.X (Currently using solidity v >= 0.8.6)
SWC-101	Integer Overflow and Underflow	Not Vulnerable	The issue persists in versions before v0.8.X.
SWC-102	Outdated Compiler Version	Vulnerable	Pragma version is not the latest one
SWC-103	Floating Pragma	Not Vulnerable	Contract uses floating pragma
SWC-104	Unchecked Call Return Value	Not Vulnerable	call() is not used
SWC-105	Unprotected Ether Withdrawal	Not Vulnerable	Appropriate function modifiers and require validations are used on sensitive functions that allow token or ether withdrawal.
SWC-106	Unprotected SELFDESTRUCT Instruction	Not Vulnerable	selfdestruct() is not used anywhere
SWC-107	Reentrancy	Not Vulnerable	No notable functions were vulnerable to it.
SWC-108	State Variable Default Visibility	Not Vulnerable	Not Vulnerable
SWC-109	Uninitialized Storage Pointer	Not Vulnerable	Not vulnerable after compiler version, v0.5.0

SWC-110	Assert Violation	Not Vulnerable	Asserts are not in use.
SWC-111	Use of Deprecated Solidity Functions	Not Vulnerable	None of the deprecated functions like block.blockhash(), msg.gas, throw, sha3(), callcode(), suicide() are in use
SWC-112	Delegatecall to Untrusted Callee	Not Vulnerable	Not Vulnerable.
SWC-113	DoS with Failed Call	Not Vulnerable	No such function was found.
SWC-114	Transaction Order Dependence	Not Vulnerable	Not Vulnerable.
SWC-115	Authorization through tx.origin	Not Vulnerable	tx.origin is not used anywhere in the code
SWC-116	Block values as a proxy for time	Not Vulnerable	Block.timestamp is not used
SWC-117	Signature Malleability	Not Vulnerable	Not used anywhere
SWC-118	Incorrect Constructor Name	Not Vulnerable	All the constructors are created using the constructor keyword rather than functions.
SWC-119	Shadowing State Variables	Not Vulnerable	Not applicable as this won't work during compile time after version 0.6.0
SWC-120	Weak Sources of Randomness from Chain Attributes	Not Vulnerable	Random generators are not used.
SWC-121	Missing Protection against Signature Replay Attacks	Not Vulnerable	No such scenario was found

SWC-122	Lack of Proper Signature Verification	Not Vulnerable	Not used anywhere
SWC-123	Requirement Violation	Not Vulnerable	Not vulnerable
SWC-124	Write to Arbitrary Storage Location	Not Vulnerable	No such scenario was found
SWC-125	Incorrect Inheritance Order	Not Vulnerable	No such scenario was found
SWC-126	Insufficient Gas Griefing	Not Vulnerable	No such scenario was found
SWC-127	Arbitrary Jump with Function Type Variable	Not Vulnerable	Jump is not used.
SWC-128	DoS With Block Gas Limit	Not Vulnerable	Not Vulnerable.
SWC-129	Typographical Error	Not Vulnerable	No such scenario was found
SWC-130	Right-To-Left-Override control character (U+202E)	Not Vulnerable	No such scenario was found
SWC-131	Presence of unused variables	Not Vulnerable	No such scenario was found
SWC-132	Unexpected Ether balance	Not Vulnerable	No such scenario was found
SWC-133	Hash Collisions With Multiple Variable Length Arguments	Not Vulnerable	abi.encodePacked() or other functions are not used.
SWC-134	Message call with hardcoded gas amount	Not Vulnerable	Not used anywhere in the code
SWC-135	Code With No Effects	Not Vulnerable	No such scenario was found
SWC-136	Unencrypted Private Data On-Chain	Not Vulnerable	No such scenario was found

4. Remediation Status

Eddy Finance is actively partnering with CredShields from this engagement to validate the discovered vulnerabilities' remediations. **A retest was performed on Sept 14th, 2023, and all the issues have been addressed.**

Also, the table shows the remediation status of each finding.

VULNERABILITY TITLE	SEVERITY	REMEDIATION STATUS
Missing Access Control in onCrossChainCall	Critical	**Fixed [14/09/2023]**
Zero Slippage value	Medium	**Pending Fix**
Hardcoded Deadline	Low	**Pending Fix**
Missing Validation in Tokens	Low	**Pending Fix**
Outdated Pragma version	Low	**Pending Fix**

Table: Summary of findings and status of remediation

5. Bug Reports

Bug ID #1 [Fixed]

Missing Access Control in onCrossChainCall

Vulnerability Type
Missing Access Control

Severity
Critical

Description
The contract ZetaSwapV2 has an external function onCrossChainCall which is used to start a swap by calling other functions. This function is supposed to be called from inside the SystemContract.depositAndCall() but there's no such access control validation on the function, allowing anyone to call the function and start the swap without depositing the tokens into the contract.

Affected Code

- https://github.com/EddyFinance/omni-chain-contracts/blob/cb8b99362735722224dceef6945b74471e9df6770/contracts/ZetaSwapV2.sol#L45-L59

Impacts
Allowing users to start a swap without depositing any tokens could drain the gas fee deposited in the contract and lead to a loss of funds.

Remediation
It is recommended to add an access control validation on the function such as only the SystemContract is able to make a call to onCrossChainCall.

Reference:
https://www.zetachain.com/docs/developers/omnichain/tutorials/withdraw/#creating-the-contract

Retest
This has been remediated by adding the access control validation in the onCrossChainCall function.

Bug ID #2

Zero Slippage value

Vulnerability Type
Slippage Risk

Severity
Medium

Description
onCrossChainCall function that is part of a contract handling cross-chain token swaps. In this function, a token swap operation is performed with a hardcoded minAmt (slippage) value set to zero. Slippage refers to the maximum acceptable difference between the expected price of an asset and the actual executed price during a swap. A slippage of zero means that the code expects the swap to return a value near 0.

Affected Code

- https://github.com/EddyFinance/omni-chain-contracts/blob/cb8b9936273572224dceef6945b74471e9df6770/contracts/ZetaSwapV2.sol#L52C7-L52C7

Impacts
The impact of setting the `minAmt` (slippage) to zero is that the code assumes that the token swap will always occur at an exact price without any price fluctuations. In a decentralized environment, token prices can vary rapidly due to market conditions, resulting in the possibility of the swap failing or being executed at a significantly different rate than expected. This can lead to undesirable outcomes, including failed transactions or losses for users.

Remediation
To make the token swap function more robust and adaptable to market conditions, it is recommended to set a non-zero slippage tolerance (e.g., a small percentage) rather than a hardcoded zero value. This will allow the code to accommodate minor price fluctuations and ensure that the swap is more likely to succeed. Or take input from user to set `minAmt`.

Retest

Bug ID #3

Hardcoded Deadline

Vulnerability Type
Business Logic

Severity
Low

Description
When performing swap operations or adding/removing liquidity on Uniswap, specifying a valid deadline is paramount. Failing to do so can result in delayed executions, causing trades at unfavorable prices and potentially leading to financial losses.

Affected Code

- https://github.com/EddyFinance/omni-chain-contracts/blob/cb8b99362735722224dceef6945b74471e9df6770/contracts/ZetaSwapV2.sol#L45-L59

Impacts.
Having a hardcoded deadline could potentially expose the users to sandwich attacks.

Remediation
It is recommended to take the deadline as input from the user to allow proper execution of transactions in a timely manner.

Retest

Bug ID #4

Missing Validation in Tokens

Vulnerability Type
Missing Input Validation

Severity
Low

Description
In the `onCrossChainCall` function, there is a missing validation check that ensures `zrc20` (the source token) should not be the same as `targetZRC20` (the target token). This means the code allows for a situation where a token can be swapped for itself, which is unnecessary and can lead to unexpected behavior.

Affected Code

- https://github.com/EddyFinance/omni-chain-contracts/blob/cb8b99362735722224dceef6945b74471e9df6770/contracts/ZetaSwapV2.sol#L45-L59

Impacts.
Allowing tokens to be swapped for themselves doesn't make logical sense and can result in inefficiencies and unexpected outcomes. It could lead to unnecessary gas costs, as the code would attempt a swap that doesn't change the token at all. It might also confuse users and waste resources on the blockchain.

Remediation
To prevent this issue, you should add a validation check at the beginning of the function to ensure that zrc20 and targetZRC20 are not the same token address. If they are the same, you should handle this case appropriately, which could involve skipping the swap or taking other actions depending on the intended behavior of your contract. This validation will help make the code more efficient and avoid unnecessary token swaps.

Retest

Bug ID #5

Outdated Pragma version

Vulnerability Type
Outdated Pragma

Severity
Low

Description
Using an outdated compiler version can be problematic, especially if there are publicly disclosed bugs and issues that affect the current compiler version.
The contracts found in the repository were allowing an old compiler version to be used, i.e., 0.8.7.

Affected Code

- https://github.com/EddyFinance/omni-chain-contracts/blob/cb8b9936273572224dceef6945b74471e9df6770/contracts/ZetaSwapV2.sol#L2

Impacts
If the smart contract gets compiled and deployed with an older or too recent version of the solidity compiler, there's a chance that it may get compromised due to the bugs present in the older versions or unidentified exploits in the new versions.
Incompatibility issues may also arise if the contract code does not support features in other compiler versions, therefore, breaking the logic.
The likelihood of exploitation is really low therefore this is only Low severity.

Remediation
Keep the compiler versions updated in all the smart contract files. Do not allow floating pragmas anywhere. It is suggested to use the 0.8.9 pragma version which is stable and not too recent.
Reference: https://swcregistry.io/docs/SWC-103

Retest

6. Disclosure

The Reports provided by CredShields is not an endorsement or condemnation of any specific project or team and do not guarantee the security of any specific project. The contents of this report are not intended to be used to make decisions about buying or selling tokens, products, services, or any other assets and should not be interpreted as such.

Emerging technologies such as Smart Contracts and Solidity carry a high level of technical risk and uncertainty. CredShields does not provide any warranty or representation about the quality of code, the business model or the proprietors of any such business model, or the legal compliance of any business. The report is not intended to be used as investment advice and should not be relied upon as such.

CredShields Audit team is not responsible for any decisions or actions taken by any third party based on the report.