BEOSIN
Web3 Security & Compliance

slcVault3

Smart Contract Security Audit

No. 202411211051

Nov 21st, 2024

SECURING BLOCKCHAIN ECOSYSTEM

WWW.BEOSIN.COM

Contents

Summary of Audit Results

After auditing, 2 Medium, 1 Low risk and 2 Info items were identified in the slcVault3 project. Specific audit details will be presented in the Findings section. Users should pay attention to the following aspects when interacting with this project:

Medium	Fixed : 2 Acknowledged: 0
Low	Fixed : 1 Acknowledged: 0
Info	Fixed : 2 Acknowledged: 0

Business overview:

The slcVault3 contract provides computational support for the algorithmic stablecoin SLC.

The stablecoin pools in xlending (USD1 and USD2) are used.

The user pledges one of the two tokens to obtain SLC tokens via the `mintSLC` function, which takes the tokens pledged by the user (e.g., USD1), and divides them equally into two parts, one of which is exchanged in xlending for USD2 to add liquidity, and the obtained LP is held by the slcVault3 contract. After that the user gets SLC (twice the amount of LP). Finally different price regression operations are performed according to the current SLC price.

If the SLC price is higher than the setting and the quantity of SLC in the contract meets the conditions, the SLC will be sold in xlending for USD1 and USD2. If there is not enough SLC in the contract, the corresponding quantity will be minted to the specified address to adjust the price of SLC downwards.

If the SLC price is lower than the setting, the contract will be USD1 and USD2 in xlending to buy SLC. if the contract USD1 or USD2 is insufficient, the contract will be destroyed to hold the LP, remove the corresponding amount of liquidity, replenish the amount and then purchase. In this way, the price of SLC is adjusted upwards.

Finally, the remaining USD1 and USD2 in the contract are checked to see if they are available to add liquidity.

Users can destroy the SLC through the `burnSLC` function, the contract will remove the corresponding amount of LP, so as to withdraw USD1 or USD2, the price return operation is the same as `mintSLC`.

1 Overview

1.1 Project Overview

Project Name	slcVault3
Project language	Solidity
Platform	Merlin
Code Base	https://github.com/artixv/xslcV3/blob/main/contracts/slcVault3.sol
Commit Hash	17289694b14fcade729083c52d3809132876faff 9f40ab04a2cfc776c26c8c443185749d27b941d9 129f4bb3aa6f6bf2a75f1cf3a301c56d0062ffa2

1.2 Audit Overview

Audit work duration: Nov 12, 2024 – Nov 21, 2024

Audit team: Beosin Security Team

1.3 Audit Method

The audit methods are as follows:

1. Formal Verification

Formal verification is a technique that uses property-based approaches for testing and verification. Property specifications define a set of rules using Beosin's library of security expert rules. These rules call into the contracts under analysis and make various assertions about their behavior. The rules of the specification play a crucial role in the analysis. If the rule is violated, a concrete test case is provided to demonstrate the violation.

2. Manual Review

Using manual auditing methods, the code is read line by line to identify potential security issues. This ensures that the contract's execution logic aligns with the client's specifications and intentions, thereby safeguarding the accuracy of the contract's business logic.

The manual audit is divided into three groups to cover the entire auditing process:

The Basic Testing Group is primarily responsible for interpreting the project's code and conducting comprehensive functional testing.

The Simulated Attack Group is responsible for analyzing the audited project based on the collected historical audit vulnerability database and security incident attack models. They identify potential attack vectors and collaborate with the Basic Testing Group to conduct simulated attack tests.

The Expert Analysis Group is responsible for analyzing the overall project design, interactions with third parties, and security risks in the on-chain operational environment. They also conduct a review of the entire audit findings.

3. Static Analysis

Static analysis is a method of examining code during compilation or static analysis to detect issues. Beosin-VaaS can detect more than 100 common smart contract vulnerabilities through static analysis, such as reentrancy and block parameter dependency. It allows early and efficient discovery of problems to improve code quality and security.

2 Findings

Index	Risk description	Severity level	Status
slcVault3-01	Wrong parameter correspondences in burnSLC functions and missing updates	Medium	Fixed
slcVault3-02	Missing logic and parameter updates in slcValueRenew function	Medium	Fixed
slcVault3-03	Missing judgement for exchange in valueRegression function	Low	Fixed
slcVault3-04	Missing events	Info	Partially Fixed
slcVault3-05	Redundant Code	Info	Fixed

Finding Details:

[slcVault3-01] Wrong parameter correspondences in burnSLC functions and missing updates

Severity Level	Medium
Type	Business Security
Lines	slcVault3.sol#L295-311
Description	The `burnSLC` function adjusts the order of tokens[ ] when performing an exchange based on the token entered by the user, but does not correspondingly adjust the order of _amount[ ] after removing mobility, which may result in USD2 corresponding to USD1's amount. And the function does not update latestBlockNumber and latestBlockUser as it in the `mintSLC` function.
Recommendation	It is recommended to adjust _amount[ ] synchronously when determining token type and adjusting tokens[ ]. And add updates to latestBlockNumber and latestBlockUser
Status	**Fixed.** Changed the order of amounts[ ] to match the corresponding tokens[ ].

[slcVault3-02] Missing logic and parameter updates in slcValueRenew function

Severity Level	**Medium**
Type	Business Security
Lines	slcVault3.sol#L167-179
Description	The `mintSLC` in the `slcValueRenew` function does not update the slcnum, which in the calculation of `slcValueRenew` will result in a larger-than-actual result. And there is no handling of the `value`<1 case. slc is balanced by minting when the price goes up, but not destroyed when the price goes down. The price regression by `valueRegression` alone when the price is lowered is not as efficient as the operation when the price is raised.
Recommendation	It is recommended to add slcnum update after `mintSLC`. And add corresponding logic. For example, when value < (999/1000), destroy the function of SLC for the corresponding number of accolades address (the algorithm is opposite to accolades) and update slcnum.
Status	**Fixed.** Added update of slcnum in `slcValueRenew` function. According to the algorithm and the actual situation, the `value` will not be less than 1, so no more processing is needed.

[slcVault3-03] Missing judgement for exchange in valueRegression function

Severity Level	Low
Type	Business Security
Lines	slcVault3.sol#L210-228
Description	Inconsistent logic when balancing prices in `valueRegression`. After the redemption has been triggered, there is no compare outAmount and tokensLimitsAmount before the exchange.
Recommendation	It is recommended to move the outAmount and tokensLimitsAmount judgment to the front of the logic, first determine whether the amount meets the requirements, then determine whether redemption is required, and finally perform the exchange.
Status	**Fixed.** Modified the position of outAmount and tokensLimitsAmount judgment.

[slcVault3-04] Missing events

Severity Level	Info
Type	Coding Conventions
Lines	slcVault3.sol#L79-121
Description	The contract's setter does not trigger an event when the contract's key parameters are modified. `function setup(address _superLibraCoin, address _xInterface, address _accoladesAddress, address _oracleAddr) public onlySetter{ superLibraCoin = _superLibraCoin; xInterface = _xInterface; oracleAddr = _oracleAddr; accoladesAddress = _accoladesAddress; } function setUSDAddress(address _USD1, address _USD2, address _USD1_USD2_Lp) public onlySetter{ USD1 = _USD1; USD2 = _USD2; USD1_USD2_Lp = _USD1_USD2_Lp; }`
Recommendation	It is recommended to emit events when modifying critical variables is a recommended practice as it provides a standardized way to capture and communicate important changes within the contract. Events enable transparency and allow external systems and users to easily track and react to these modifications.
Status	**Partially Fixed.** Only the setter transfer event is added, and other events are missing and not fixed.

[slcVault3-05] Redundant Code

Severity Level	Info
Type	Coding Conventions
Lines	slcVault3.sol#L30
Description	timestampInterval is not used in the contract and is redundant code. `uint public timestampInterval;`
Recommendation	It is recommended to delete.
Status	**Fixed.** Redundant code has been removed.

3 Appendix

3.1 Vulnerability Assessment Metrics and Status in Smart Contracts

3.1.1 Metrics

In order to objectively assess the severity level of vulnerabilities in blockchain systems, this report provides detailed assessment metrics for security vulnerabilities in smart contracts with reference to CVSS 3.1(Common Vulnerability Scoring System Ver 3.1).

According to the severity level of vulnerability, the vulnerabilities are classified into four levels: "critical", "high", "medium" and "low". It mainly relies on the degree of impact and likelihood of exploitation of the vulnerability, supplemented by other comprehensive factors to determine of the severity level.

Likelihood \ Impact	Severe	High	Medium	Low
Probable	Critical	High	Medium	Low
Possible	High	Medium	Medium	Low
Unlikely	Medium	Medium	Low	Info
Rare	Low	Low	Info	Info

3.1.2 Degree of impact

- **Critical**

Critical impact generally refers to the vulnerability can have a serious impact on the confidentiality, integrity, availability of smart contracts or their economic model, which can cause substantial economic losses to the contract business system, large-scale data disruption, loss of authority management, failure of key functions, loss of credibility, or indirectly affect the operation of other smart contracts associated with it and cause substantial losses, as well as other Critical and mostly irreversible harm.

- **High**

High impact generally refers to the vulnerability can have a relatively serious impact on the confidentiality, integrity, availability of the smart contract or its economic model, which can cause a greater economic loss, local functional unavailability, loss of credibility and other impact to the contract business system.

- **Medium**

Medium impact generally refers to the vulnerability can have a relatively minor impact on the confidentiality, integrity, availability of the smart contract or its economic model, which can cause a small amount of economic loss to the contract business system, individual business unavailability and other impact.

- **Low**

Low impact generally refers to the vulnerability can have a minor impact on the smart contract, which can pose certain security threat to the contract business system and needs to be improved.

3.1.3 Likelihood of Exploitation

- **Probable**

Probable likelihood generally means that the cost required to exploit the vulnerability is low, with no special exploitation threshold, and the vulnerability can be triggered consistently.

- **Possible**

Possible likelihood generally means that exploiting such vulnerability requires a certain cost, or there are certain conditions for exploitation, and the vulnerability is not easily and consistently triggered.

- **Unlikely**

Unlikely likelihood generally means that the vulnerability requires a high cost, or the exploitation conditions are very demanding and the vulnerability is highly difficult to trigger.

- **Rare**

Rare likelihood generally means that the vulnerability requires an extremely high cost or the conditions for exploitation are extremely difficult to achieve.

3.1.4 Fix Results Status

Status	Description
Fixed	The project party fully fixes a vulnerability.
Partially Fixed	The project party did not fully fix the issue, but only mitigated the issue.
Acknowledged	The project party confirms and chooses to ignore the issue.

3.2 Audit Categories

No.	Categories	Subitems
1	Coding Conventions	Deprecated Items
		Redundant Code
		require/assert Usage
		Default Values
2	General Vulnerability	Insufficient Address Validation
		Lack Of Address Normalization
		Variable Override
		DoS (Denial Of Service)
		Function Call Permissions
		Call/Delegatecall Security
		Tx.origin Usage
		Returned Value Security
		Mathematical Risk
		Overriding Variables
3	Business Security	Business Logics
		Business Implementations
		Manipulable Token Price
		Centralized Asset Control
		Arbitrage Attack
		Access Control

Beosin classified the security issues of smart contracts into three categories: Coding Conventions, General Vulnerability, Business Security. Their specific definitions are as follows:

- **Coding Conventions**

Audit whether smart contracts follow recommended language security coding practices. For example, smart contracts developed in Rust language should fix the compiler version and do not use deprecated keywords.

- **General Vulnerability**

General Vulnerability include some common vulnerabilities that may appear in smart contract projects. These vulnerabilities are mainly related to the characteristics of the smart contract itself, such as integer overflow/underflow and denial of service attacks.

- **Business Security**

Business security is mainly related to some issues related to the business realized by each project, and has a relatively strong pertinence. For example, whether the lock-up plan in the code match the white paper, or the flash loan attack caused by the incorrect setting of the price acquisition oracle.

3.3 Disclaimer

The Audit Report issued by Beosin is related to the services agreed in the relevant service agreement. The Project Party or the Served Party (hereinafter referred to as the "Served Party") can only be used within the conditions and scope agreed in the service agreement. Other third parties shall not transmit, disclose, quote, rely on or tamper with the Audit Report issued for any purpose.

The Audit Report issued by Beosin is made solely for the code, and any description, expression or wording contained therein shall not be interpreted as affirmation or confirmation of the project, nor shall any warranty or guarantee be given as to the absolute flawlessness of the code analyzed, the code team, the business model or legal compliance.

The Audit Report issued by Beosin is only based on the code provided by the Served Party and the technology currently available to Beosin. However, due to the technical limitations of any organization, and in the event that the code provided by the Served Party is missing information, tampered with, deleted, hidden or subsequently altered, the audit report may still fail to fully enumerate all the risks.

The Audit Report issued by Beosin in no way provides investment advice on any project, nor should it be utilized as investment suggestions of any type. This report represents an extensive evaluation process designed to help our customers improve code quality while mitigating the high risks in blockchain.

3.4 About Beosin

Beosin is the first institution in the world specializing in the construction of blockchain security ecosystem. The core team members are all professors, postdocs, PhDs, and Internet elites from world-renowned academic institutions. Beosin has more than 20 years of research in formal verification technology, trusted computing, mobile security and kernel security, with overseas experience in studying and collaborating in project research at well-known universities. Through the security audit and defense deployment of more than 2,000 smart contracts, over 50 public blockchains and wallets, and nearly 100 exchanges worldwide, Beosin has accumulated rich experience in security attack and defense of the blockchain field, and has developed several security products specifically for blockchain.

BEOSIN
Web3 Security & Compliance

Official Website
https://www.beosin.com

Telegram
https://t.me/beosin

X
https://x.com/Beosin_com

Email
service@beosin.com

LinkedIn
https://www.linkedin.com/company/beosin/