Three Sigma

Code Audit

keyring

Keyring Zero-knowledge compliance solution

Disclaimer

Code Audit

Keyring Zero-knowledge compliance solution

Disclaimer

The ensuing audit offers no assertions or assurances about the code's security. It cannot be deemed an adequate judgment of the contract's correctness on its own. The authors of this audit present it solely as an informational exercise, reporting the thorough research involved in the secure development of the intended contracts, and make no material claims or guarantees regarding the contract's post-deployment operation. The authors of this report disclaim all liability for all kinds of potential consequences of the contract's deployment or use. Due to the possibility of human error occurring during the code's manual review process, we advise the client team to commission several independent audits in addition to a public bug bounty program.

Table of Contents

Code Audit

Keyring Zero-knowledge compliance solution

Table of Contents

Summary

Code Audit

Keyring Zero-knowledge compliance solution

Summary

Three Sigma Labs audited keyring in a 1 person week engagement. The audit was conducted from 15-12-2023 to 29-12-2023..

Protocol Description

Keyring is a Zero-Knowledge compliance solution for maximum liquidity and composability.

Scope

Code Audit

Keyring Zero-knowledge compliance solution

Scope

The only code in scope was the update of the KeyringZkCredentialUpdater.sol.

Assumptions

The external calls performed from the **`KeyringZkCredentialUpdater`** to other smart contracts are assumed to be correct and return the expected results.

Methodology

Code Audit

Keyring Zero-knowledge compliance solution

Methodology

To begin, we reasoned meticulously about the contract's business logic, checking security-critical features to ensure that there were no gaps in the business logic and/or inconsistencies between the aforementioned logic and the implementation. Second, we thoroughly examined the code for known security flaws and attack vectors. Finally, we discussed the most catastrophic situations with the team and reasoned backwards to ensure they are not reachable in any unintentional form.

Taxonomy

In this audit we report our findings using as a guideline Immunefi's vulnerability taxonomy, which can be found at immunefi.com/severity-updated/. The final classification takes into account the severity, according to the previous link, and likelihood of the exploit. The following table summarizes the general expected classification according to severity and likelihood; however, each issue will be evaluated on a case-by-case basis and may not strictly follow it.

Severity / Likelihood	LOW	MEDIUM	HIGH
NONE	None		
LOW	Low		
MEDIUM	Low	Medium	Medium
HIGH	Medium	High	High
CRITICAL	High	Critical	Critical

Project Dashboard

Code Audit

Keyring Zero-knowledge compliance solution

Project Dashboard

Application Summary

Name	Keyring
Commit	2ae2376de2c63745613866760ccc9a1bc9ff08b1
Language	Solidity
Platform	Ethereum

Engagement Summary

Timeline	22-12-2023 to 29-12-2023
№ of Auditors	1
Review Time	1 person weeks

Vulnerability Summary

Issue Classification	Found	Addressed	Acknowledged
Critical	0	0	0
High	0	0	0
Medium	0	0	0
Low	1	0	1
None	7	4	3

Category Breakdown

Suggestion	3
Documentation	0
Bug	1
Optimization	2
Good Code Practices	2

Code Maturity Evaluation

Code Audit

Keyring Zero-knowledge compliance solution

Code Maturity Evaluation

Code Maturity Evaluation Guidelines

Category	Evaluation
Access Controls	The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system.
Arithmetic	The proper use of mathematical operations and semantics.
Centralization	The presence of a decentralized governance structure for mitigating insider threats and managing risks posed by contract upgrades
Code Stability	The extent to which the code was altered during the audit.
Upgradeability	The presence of parameterizations of the system that allow modifications after deployment.
Function Composition	The functions are generally small and have clear purposes.
Front-Running	The system's resistance to front-running attacks.
Monitoring	All operations that change the state of the system emit events, making it simple to monitor the state of the system. These events need to be correctly emitted.
Specification	The presence of comprehensive and readable codebase documentation.
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage.

Code Maturity Evaluation Results

Category	Evaluation
Access Controls	Satisfactory. The codebase has a strong access control mechanism.
Arithmetic	Satisfactory. The codebase uses Solidity version >0.8.0 implementing safe arithmetic
Centralization	Weak. Although the access control is very well implemented, there are several privileged roles.
Code Stability	Satisfactory. The code was stable during the audit.
Upgradeability	Satisfactory. Transparent proxies are used for most contracts.
Function Composition	Satisfactory. Functionalities are well split into different contracts and helpers.
Front-Running	Satisfactory. There are no known front-running opportunities.
Monitoring	Satisfactory. Events are generally emitted when there are state changing occurrences.
Specification	Satisfactory. Functions are generally commented and the protocol has public documentation.
Testing and Verification	Satisfactory. Extensive test code coverage as well as usage of tools and different test methods.

Findings

Code Audit

Keyring Zero-knowledge compliance solution

Findings

3S-KEYRING-L01

AccessControlDefaultAdminRules is preferred when using the default admin

Id	3S-KEYRING-L01
Classification	Low
Severity	Low
Likelihood	Low
Category	Suggestion
Status	Acknowledged

Description

The **`DEFAULT_ADMIN_ROLE`** is the default admin of all roles in Openzeppelin's AccessControl. The codebase uses this mechanism extensively, which means that it would be preferred to use AccessControlDefaultAdminRules instead. It implements a 2 step role transfer procedure, a scheduling delay, cancellation, being much more secure overall.

3S-KEYRING-N01

Wrong error message when configuring new policies

Id	3S-KEYRING-N02
Classification	None
Category	Bug
Status	Addressed in eb3cb19

Description

In **`Degradable:setPolicyParameters()`**, the error message indicates that policies 0 and 1 can not be configured. However, policy 1 can be configured, as **`FIRST_CONFIGURABLE_POLICY = 1`**.

3S-KEYRING-N02

`++i` is cheaper than `i++` and increment could be placed in an unchecked block

Id	3S-KEYRING-N02
Classification	None
Category	Optimization
Status	Addressed in 5e03215

Description

In **`KeyringMerkleAuthZkCredentialUpdater:updateCredentials()`**, the for loop can be modified to save additional gas:

```
for(uint256 i = 0; i < 24;) {
    ...
    unchecked {
        ++i;
    }
}
```

3S-KEYRING-N03

Specific imports should be preferred over file imports

Id	3S-KEYRING-N03
Classification	None
Category	Good Code Practices
Status	To be addressed in future release

Description

`KeyringMerkleAuthZkCredentialUpdater` imports files directly, but it is recommended to import the exact smart contract, lib or struct directly.

3S-KEYRING-N04

Solidity version can be upgraded

Id	3S-KEYRING-N04
Classification	None
Category	Suggestion
Status	Acknowledged

Description

`KeyringMerkleAuthZkCredentialUpdater` is using solidity 0.8.14, but could use 0.8.19 or more if deployed to a blockchain that supports **`PUSH0`**.

3S-KEYRING-N05

Sender and trader are the same in **`AcceptCredentialUpdate`**

Id	3S-KEYRING-N05
Classification	None
Category	Suggestion
Status	Acknowledged

Description

Sender and trader in **`KeyringMerkleAuthZkCredentialUpdater:updateCredentials()`** are the same, it might not be necessary to emit both in the AcceptCredentialUpdate event.

3S-KEYRING-N06

Nested ifs can be refactored to simplify code

Id	3S-KEYRING-N06
Classification	None
Category	Good Code Practices
Status	Addressed in 313e076

Description

The if (policyBackdoorCount == 1) condition in **KeyringMerkleAuthZkCredentialUpdater** can be simplified to:

```
if (policyBackdoorCount == 1) {
    bytes32 nextRequireBackdoor =
IPolicyManager(POLICY_MANAGER).policyBackdoorAtIndex(policyId, 0);
    if (requireBackdoor != NULL_BYTES32 && requireBackdoor !=
nextRequireBackdoor)
        revert Unacceptable({
            reason: "all policies in the proof must rely on the same
backdoor or no backdoor"
        });
    requireBackdoor = nextBackdoor;
}
```

3S-KEYRING-N07

Pack and unpack in **`Pack12x20.sol`** could use a loop to reduce deployment costs

Id	3S-KEYRING-N07
Classification	None
Category	Optimization
Status	Acknowledged

Description

pack() and unpack() in **`Pack12x20.sol`** make the same operation 12 times, which increases deployment costs significantly. Consider replacing it by a loop.