QuillAudits

Audit Report
October, 2023

For

SELFKEY

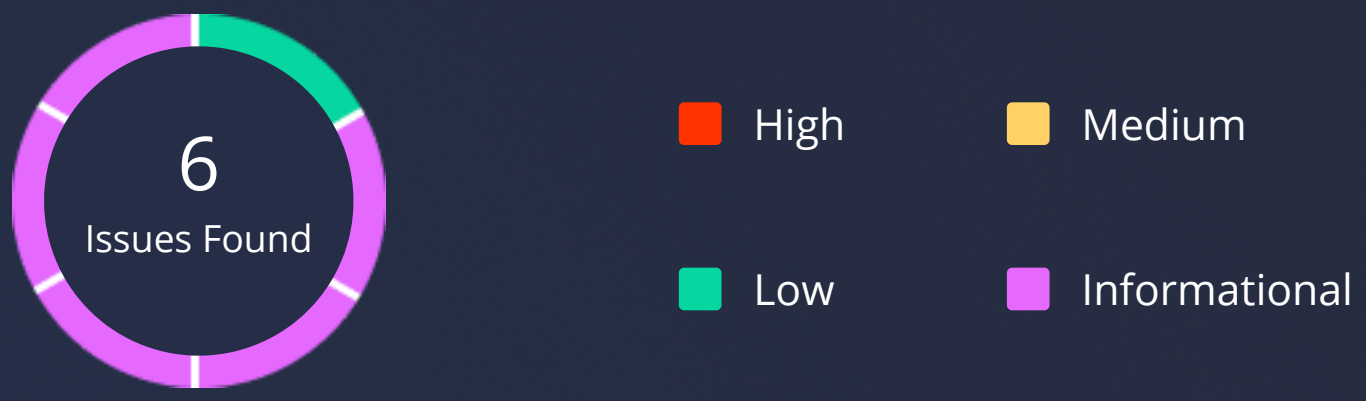
Table of Content

Table of Content

Executive Summary

Project Name	Selfkey PoI Locking
Project URL	*https://selfkey.org/*
Overview	The contracts allows user to Lock Key tokens in Exchange for the Self tokens. The user submits Locking parameters along with Signature to Lock the tokens. The Mintable amount are calculated per second.
Audit Scope	*https://github.com/SelfKeyDAO/selfkey-poi-lock*
Contracts in Scope	ISelfkeyPoiLock.sol SelfkeyPoiLock.sol
Commit Hash	e07657581ee6b47f4040d2ffc3ce8511ac87b719
Language	Solidity
Blockchain	Polygon
Method	Manual Analysis, Functional Testing and Automated Tests
Review 1	17th October 2023 - 25th October 2023
Updated Code Received	27th October 2023
Review 2	28th October 2023 - 2nd November 2023
Fixed In	*https://github.com/SelfKeyDAO/selfkey-poi-lock* Commit: 2ce160fa37c79da307d1b41d5e3d49b427304103

Number of Security Issues per Severity



	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	1	2
Partially Resolved Issues	0	0	0	0
Resolved Issues	0	0	0	3

Checked Vulnerabilities

- Access Management
- Arbitrary write to storage
- Centralization of control
- Ether theft
- Improper or missing events
- Logical issues and flaws
- Arithmetic Correctness
- Race conditions/front running
- SWC Registry
- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- Exception Disorder
- Gasless Send
- Use of tx.origin
- Malicious libraries
- Compiler version not fixed
- Address hardcoded
- Divide before multiply
- Integer overflow/underflow
- ERC's conformance
- Dangerous strict equalities
- Tautology or contradiction
- Return values of low-level calls
- Missing Zero Address Validation
- Private modifier
- Revert/require functions
- Multiple Sends
- Using suicide
- Using delegatecall
- Upgradeable safety
- Using throw

Checked Vulnerabilities

- Using inline assembly
- Unsafe type inference
- Style guide violation
- Implicit visibility level

Techniques and Methods

Throughout the audit of smart contracts, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments, match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour mentioned in the whitepaper.
- Implementation of ERC standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods, and tools were used to review all the smart contracts.

Structural Analysis

In this step, we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

A static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual Analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behavior of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit

Remix IDE, Truffle, Solhint, Mythril, Slither, Solidity Statistic Analysis.

Types of Severity

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below.

High Severity Issues

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium Severity Issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low Severity Issues

Low-level severity issues can cause minor impact and are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are four severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

A. Contract - SelfkeyPoILock

High Severity Issues

No issues were found.

Medium Severity Issues

No issues were found.

Low Severity Issues

A1. TimeLock becomes insignificant after a few Locks and Unlocks

Description

The **availableOf()** function calculates how much amount is allowed to be withdrawn according to the lock time after Locking, but when a significant number of Locks have been made then the sum becomes so large that the user can withdraw any amount which is less than equal to his balance immediately after Locking.

The below test shows that in the first iteration, after **3800 seconds (timeLockDuration)**, the user withdraws the Locked amount, which is intended.
But in further iterations after Locks, users can immediately withdraw the locked amount despite the timelock.

Test

https://gist.github.com/aga7hokakological/6694a7bce19fe2069a3000bfa2982d9f

Remediation

To resolve the issue please update the code with appropriate logic.

Status

Acknowledged

SelfKey Team's Comment

After talking with the Selfkey team about the issue, they said that the functionality related to timelock will be removed. If needed they will add the functionality in the future that's why the status is kept as **acknowledged**.

Informational Issues

A2. Missing require statement while setting Mintable Token rate

Description

In the contract SelfkeyPoILock Mintable Token rate is getting set using function **setMintableTokenRate()** function but it is not checking if the Mintable Token Rates are active or not. This is for the first time when the contract is deployed and the variables are supposed to be initiated.

Remediation

Please make sure to add require statement checking the active variable status in **setMintableTokenRate()** function.

Status

Acknowledged

A3. Wrong event emission

Description

In the contract SelfkeyPOILock in **updateLockingMintableTokenStatus()** function when status is changed the events are emitted but the emitted events suggest otherwise. **LockingResumed()** and **LockingPaused()** suggest that the contract has paused locking or has resumed instead of mintable tokens rate being on.

Remediation

Please make sure to change the event's name to something like RewardsActive(), RewardsPaused().

Status

Resolved

A4. Redundant timestamp assignment

Description

Storage variable **updatedAt** is being reassigned **block.timestamp** in the functions which already are using **checkpoint()** modifier which updates that variable.

Remediation

Remove the line **updateAt = block.timestamp** from **setMintableTokenRate()** and **updateLockingMintableTokenStatus()** functions to save gas.

Status

Resolved

A5. rewardsTokenAddress cannot call notifyMintableTokenWithdraw()

Description

There is a check in the **notifyMintableTokenWithdraw()** function to confirm that the caller of the function is the **rewardsTokenAddress**. This address is an ERC20 contract with no capability to make external calls thereby making the calls function always revert.

Remediation

Ensure proper passing checks happen in functions.

Status

Resolved

B. Contract - SelfkeyIdAuthorization

High Severity Issues

No issues were found.

Medium Severity Issues

No issues were found.

Low Severity Issues

No issues were found.

Informational Issues

No issues were found.

C. Contract - SelfToken

High Severity Issues

No issues were found.

Medium Severity Issues

No issues were found.

Low Severity Issues

No issues were found.

Informational Issues

C1. Use of whenNotPaused() modifier

Description

whenNotPaused() modifier is only used with **mint()** and **selfMint()** functions, but burning and transfer is unrestricted.

Remediation

Use **whenNotPaused()** modifier with **burn()**, **burnFrom()**, **transfer()** and **transferFrom()** functions.

Status

Acknowledged

D. Contract - KeyToken

High Severity Issues

No issues were found.

Medium Severity Issues

No issues were found.

Low Severity Issues

No issues were found.

Informational Issues

No issues were found.

Functional Tests

Some of the tests performed are mentioned below:

- User should not be able to Lock twice with same signature
- Attacker should not be able to Lock tokens for other user
- Mintable Token rate change should not affect existing accumulated amounts
- Global userPerTokenStored should not affect the already accumulated amounts for the user2 when he locks KEY tokens
- User is able to unlock before timelock

Automated Tests

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

Closing Summary

In this report, we have considered the security of the SelfkeyPOILock. We performed our audit according to the procedure described above.

Some issues of Low and informational severity were found, Some suggestions and best practices are also provided in order to improve the code quality and security posture.

Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in SelfkeyPOILock smart contracts. However, it is important to understand that no security audit can guarantee complete protection against all possible security threats. QuillAudits audit reports are based on the information provided to us at the time of the audit, and we cannot guarantee the accuracy or completeness of this information. Additionally, the security landscape is constantly evolving, and new security threats may emerge after the audit has been completed.

Therefore, it is recommended that multiple audits and bug bounty programs be conducted to ensure the ongoing security of SelfkeyPOILock smart contracts. One audit is not enough to guarantee complete protection against all possible security threats. It is important to implement proper risk management strategies and stay vigilant in monitoring your smart contracts for potential security risks.

QuillAudits cannot be held liable for any security breaches or losses that may occur subsequent to and despite using our audit services.. It is the responsibility of the SelfkeyPOILock to implement the recommendations provided in our audit reports and to take appropriate steps to mitigate potential security risks.

About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem.

850+
Audits Completed

$30B
Secured

$30B
Lines of Code Audited

Follow Our Journey

Audit Report
October, 2023

For

SELFKEY

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com