QuillAudits

Audit Report
March, 2021

Savix

Contents

Scope of Audit

The scope of this audit was to analyze and document Savix smart contract codebase for quality, security, and correctness.

Check Vulnerabilities

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Byte array
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Redundant fallback function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- Unsafe type inference
- 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-20 token 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.
SmartCheck.

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

Code Review / Manual Analysis
Manual Analysis or review of code was done to identify new vulnerability 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 automated analysis were manually verified.

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

Issue Categories

Every issue in this report has been assigned with 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 to be fixed before moving to a live environment.

Medium level 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 level severity issues

Low level severity issues can cause minor impact and or 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 severity four issues which indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Number of issues per severity

	High	Medium	Low	Informational
Open	0	0	0	0
Closed	0	5	3	0

Introduction

During the period of **March 2nd, 2021 to March 3rd, 2021** - Quillhash Team performed a security audit for Savix smart contracts.
The code for the audit was taken from following the official GitHub link:
https://github.com/SavixOrg/Savix/tree/main/contracts

Commit Hash-

- **Savix.sol:**
  5b900419d83135d33902245ab158a8e0333c5de9

- **SavixSupply.sol:**
  6b5923e6d289b5fba4e394734768265f952b64bb

Issues Found – Code Review / Manual Testing

A. Savix.sol

About the Contract:
Savix.sol implements the basic ERC20 token standard along with an embedded mechanism of automatically inflating the total supply of the token based on a mathematical function over time defined in the SavixSupply library.

Moreover, the balances of the token are stored as fragments of the total supply. This ensures that the inflation is equally distributed among all token holders
and yields an interest on the token balances.

Medium severity issues

1. **State Variable Initialized but never Never Used**
   Line no - 15, 23
   Status: CLOSED

   Description:
   While the previous unused State Variable(MAX_SUPPLY) has now been removed, 2 more unused state variables were found during the final audit.

The following State Variables have been initialized but never used throughout the contract.

- CONSTINTEREST
- MAX_UINT128

Recommendation:
In order to optimize Gas usage in the contract, the variables that aren't used in the contract should either be removed or used in an effective manner.

2. **Excessive Gas Consumption in Loopsn**
Line no - 311-315, 306-307, 288-292
Status: CLOSED

Description:
The loops implemented in the above-mentioned lines use the **.length** state variable of the respective array to dynamically decide the upper bound of the iteration.
However, for every iteration of for loop, state variables like .length of the non-memory array will consume comparatively more gas.
Therefore, it would be more effective to use a local variable instead of state variable like **.length** in a loop

Recommended way :
In order to reduce the extra gas consumption, it's advisable to use a local variable.

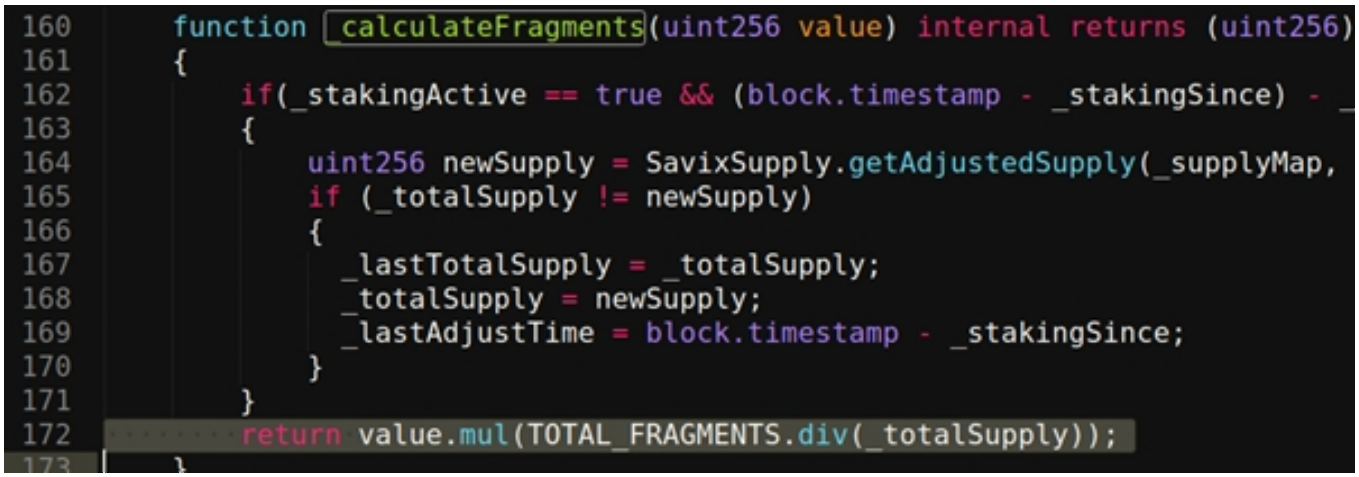
For Instance:

```
uint256 localVariable =values.length;
 for (uint i = 0; i < localVariable; i++){
            valuesum += values[i];
        _balances[_owner] =
_balances[_owner].sub( _calculateFragments(valuesum),"ERC20:
distribution total amount exceeds balance");
}
```

3. **Division precedes Multiplication**
   Line no - 172
   Status: CLOSED

   Description:
   The return value in the **_calculateFragments** function performs multiplication on the result of a division.

   This might lead to a loss of expected precision to some extent.



   Recommendation:
   Multiplication before the division is considered a better practice.

Low level severity issues

4. **External visibility should be preferred**
   Status: CLOSED

   Description:
   Functions that are never called throughout the contract should be marked as **external** visibility instead of **public** visibility.

   This will effectively result in Gas Optimization as well.

   Therefore, the following function must be marked as **external** within the contract:

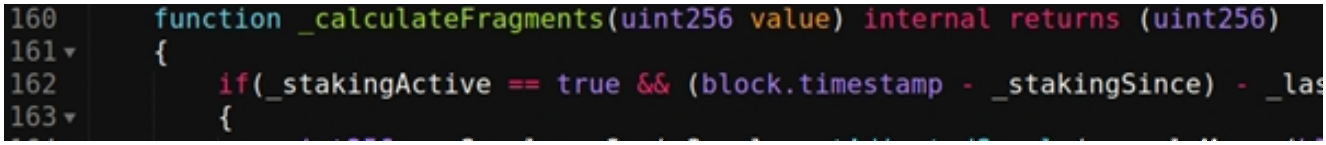
   - **getBurnAmount**

Recommendation:
The above-mentioned functions should be assigned external visibility.

5. **Comparison to boolean Constant**
   Line no - 162
   Status: CLOSED

   Description:
   Boolean constants can directly be used in conditional statements.

   Therefore, it's not considered a better practise to explicitly use **TRUE or FALSE** for comparisons.



   Recommendation:
   The equality to boolean constants must be eradicated from the above-mentioned line.

6. **Absence of Error messages in Require Statements**
   Status: CLOSED

   Description:
   Some of the **require statements** in the contract don't include an error message. While this makes it troublesome to detect the reason behind a particular function revert, it also reduces the readability of the code.

   The following lines still have **require** statements without error messages:
   - Line 62
   - Line 68
   - Line 303

   Recommendation:
   Error messages should be included in every require statement

B. SavixSupply.sol

About the Contract:
SavixSupply.sol is a library that includes the crucial mathematical function and is mainly used by the Savix.sol contract to perform imperative calculations.

Medium severity issues

1. **Division precedes Multiplication**
   Line no - 66, 74-75
   Status: CLOSED

   Description:
   The return value in the **getAdjustedSupply** & **getDailyInterest** function performs multiplication on the result of a division.

   This might lead to a loss of expected precision to some extent.

   Recommendation:
   Multiplication before division is considered as a better practice.

2. **SafeMath library Not used for Subtractions**
   Status: CLOSED

   Description:
   The SavixSupply.sol includes some crucial mathematical operations. However, the effective use of SafeMath library for the arithmetic operations performed was not found.

   While the SafeMath library was used for division operations(SafeMath.div), the same was not used to perform Subtractions.
   Since the functions like **getAdjustedSupply & getDailyInterest** include the subtraction operations on some larger and significant arithmetic variables, not using the SafeMath could lead to an integer overflow or underflow exploit scenario

Recommendation:
Using SafeMath library ensures protection from OverFlow and Underflow attacks in Solidity. Hence it's recommended to use SafeMath library for crucial subtraction operations.

Closing Summary

Overall, smart contracts are very well written and adhere to guidelines. All the issues found during the initial audit in the Savix.sol and Savix Supply.sol have now been fixed and the ERC20 compatibility of the Savix.sol Contract has now been confirmed.

Disclaimer

Quillhash audit is not a security warranty, investment advice, or an endorsement of the Savix platform. This audit does not provide a security or correctness guarantee of the audited smart contracts. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the Savix Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

Savix

QuillAudits

Canada, India, Singapore and United Kingdom

audits.quillhash.com

hello@quillhash.com