QuillAudits

Audit Report
June, 2021

FORMATION.FI

Contents

Introduction

This Audit Report highlights the overall security of the Formation Finance Smart Contract. With this report, we have tried to ensure the reliability of their smart contract by a complete assessment of their system's architecture and the smart contract codebase.

Auditing Approach and Methodologies applied

The Quillhash team has performed thorough testing of the project starting with analysing the code design patterns in which we reviewed the smart contract architecture to ensure it is structured and safe use of third party smart contracts and libraries.

Our team then performed a formal line by line inspection of the Smart Contract to find any potential issue like race conditions, transaction-ordering dependence, timestamp dependence, and denial of service attacks.

In the Unit testing Phase, we coded/conducted Custom unit tests written for each function in the contract to verify that each function works as expected. In Automated Testing, We tested the Smart Contract with our in-house developed tools to identify vulnerabilities and security flaws.

The code was tested in collaboration with our multiple team members and this included -

- Testing the functionality of the Smart Contract to determine proper logic has been followed throughout the process.
- Analysing the complexity of the code by thorough, manual review of the code, line-by-line.
- Deploying the code on testnet using multiple clients to run live tests
- Analysing failure preparations to check how the Smart Contract performs in case of bugs and vulnerabilities.
- Checking whether all the libraries used in the code are on the latest version.
- Analysing the security of the on-chain data.

Audit Details

Project Name: Formation Finance
Github Commit: 734db769c98f5ebb9698fa09b8fbe2c0ed0477f0
Languages: Solidity (Smart contract), Javascript (Unit Testing)
Platforms and Tools: Remix IDE, Truffle, Truffle Team, Ganache, Slither, Surya

Summary of Smart Contract

QuillAudits conducted a security audit of a smart contract of Formation Finance. Formation Finance contracts are used to create a token contract and vesting contract to vest formation finance native tokens.

- Native Tokens Contract
- Vesting contract with vesting period of tokens and withdraw based on eligibility

Audit Goals

The focus of the audit was to verify that the smart contract system is secure, resilient and working according to its specifications. The audit activities can be grouped into the following three categories:

Security
Identifying security related issues within each contract and the system of contract.

Sound Architecture
Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.

Code Correctness and Quality

A full review of the contract source code. The primary areas of focus include:

- Accuracy
- Readability
- Sections of code with high complexity
- Quantity and quality of test coverage

Security Level references

Every issue in this report was assigned a severity level from the following:

High level severity issues

Issues on this level are critical to the smart contract's performance/ functionality and should be fixed before moving to a live environment.

Medium level severity issues

Issues on this level could potentially bring problems and should eventually be fixed.

Low level severity issues

Issues on this level are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

Number of issues per severity

	Low	Medium	High	Recommendations
Open	0	0	0	0
Closed	4	1	0	0

Manual Audit

High level severity issues

No high severity issues

Medium level severity issues

1. **No assurity that contract has sufficient tokens at the time of withdraw**

   At the time of adding the recipient, you should use the transferFrom() function to transfer tokens to a smart contract with the same amount that could be withdrawn later when vesting is complete.

   There is no surety that tokens are withdrawable after the vesting period and that are available in vesting smart contract.

   Status: Fixed by Developer

Low level severity issues

1. **unnecessary require has been used in the constructor of the Vesting contract**

```
require(_totalAmount > 0);
require(_withdrawInterval > 0);
```

   Require conditions use more gas if used unnecessarily.

   As the constructor values are in control of the owner or deployer, they will take care of the condition no need to add require in constructor values.

   Status: Fixed by Developer

2. **Variable value set is not required**

```
isStartTimeSet = false;
isPaused = false;
```

   Variable is set to false in the constructor. Although its value automatically is ‘false’, you don’t need to put it again unless you need to emit an event to notify through deployment.

Variable is set to false in the constructor. Although its value automatically is ‘false’, you don’t need to put it again unless you need to emit an event to notify through deployment.

You can simply remove the lines and save the gas cost.

Or if you want to add it, please use an event to notify on the status of the variable unless it’s not in favour of gas usage.

Status: Fixed by Developer

3. **Should not be able to add recipient again one added**

   Function addRecipient () doesn’t check if the recipient is already added or not and overwrites the recipient again.

   It shouldn't be like that, or that’s intentionally there?

   Please add a required check for that to save gas so that by mistake, no overwrite takes place.

   Status: Fixed by Developer

4. **Use Safemath for all operations**

```
uint256 vestedAmount =
    unlockRate.mul(block.timestamp - startTime).add(
      initialUnlockAmount
    );
```

   Use safe math for all the operations to avoid overflow and underflow condition.

```
uint256 vestedAmount =
    unlockRate.mul((block.timestamp).sub(startTime)).add(
      initialUnlockAmount
    );
```

Status: Fixed by Developer

Functional test

We did functional tests for different contracts as well manually. Below is the report.

FormToken.sol

- symbol provide symbol of a token in string
  -- > PASS

- name provide name of a token in string
  --> PASS

- decimal provide decimal of a token in string
  --> PASS

- totalSupply provide total supply of a token in string
  --> PASS

- balanceOf provide balance of an address
  --> PASS

- transfer transfer token from one address to another address provided the value to transfer
  --> PASS

- transferFrom transfer tokens on behalf of an address (token holder)
  --> PASS

- Approve function allow msg.sender to spend tokens by address within the value provided
  --> PASS

tokenVesting.sol

- constructor use to initialize values to token vesting contract
  -- > PASS

- setStartTime set time to start vesting
  --> PASS

- addRecipient is use to add recipient for vesting tokens
  --> PASS

- addRecipients is use to add multiple recipient for vesting token in one transaction
  --> PASS

Automated Testing

Slither Tool Result

FormToken.sol

```
Detected issues with version pragma in FormToken.sol:
        - pragma solidity0.5.16 (Context.sol#3): it allows old versions
        - pragma solidity0.5.16 (FormToken.sol#1): it allows old versions
        - pragma solidity0.5.16 (IERC20.sol#3): it allows old versions
        - pragma solidity0.5.16 (SafeMath.sol#3): it allows old versions
Reference: https://github.com/trailofbits/slither/wiki/Detectors-Documentation#incorrect-version-of-solidity
INFO:Detectors:
Function 'Context._msgSender' (Context.sol#16-18) is not in mixedCase
Function 'Context._msgData' (Context.sol#20-23) is not in mixedCase
Variable 'FormToken._symbol' (FormToken.sol#10) is not in mixedCase
Variable 'FormToken._name' (FormToken.sol#11) is not in mixedCase
Variable 'FormToken._decimals' (FormToken.sol#12) is not in mixedCase
Variable 'FormToken._totalSupply' (FormToken.sol#13) is not in mixedCase
Reference: https://github.com/trailofbits/slither/wiki/Detectors-Documentation#conformance-to-solidity-naming-conventions
```

tokenVesting.sol

```
INFO:Detectors:
Function 'Context._msgSender' (Context.sol#16-18) is not in mixedCase
Function 'Context._msgData' (Context.sol#20-23) is not in mixedCase
Function 'Pausable._pause' (Pausable.sol#74-77) is not in mixedCase
Function 'Pausable._unpause' (Pausable.sol#86-89) is not in mixedCase
Parameter '_formToken' of TokenVesting. (TokenVesting.sol#42) is not in mixedCase
Parameter '_totalAmount' of TokenVesting. (TokenVesting.sol#43) is not in mixedCase
Parameter '_initialUnlock' of TokenVesting. (TokenVesting.sol#44) is not in mixedCase
Parameter '_wlthdrawInterval' of TokenVesting. (TokenVesting.sol#45) is not in mixedCase
Parameter '_releaseRate' of TokenVesting. (TokenVesting.sol#46) is not in mixedCase
Parameter '_newRecipient' of TokenVesting.addRecipient (TokenVesting.sol#63) is not in mixedCase
Parameter '_totalAmount' of TokenVesting.addRecipient (TokenVesting.sol#63) is not in mixedCase
Parameter '_newRecipients' of TokenVesting.addRecipients (TokenVesting.sol#87) is not in mixedCase
Parameter '_totalAmounts' of TokenVesting.addRecipients (TokenVesting.sol#88) is not in mixedCase
Parameter '_newStartTime' of TokenVesting.setStartTime (TokenVesting.sol#114) is not in mixedCase
Reference: https://github.com/trailofbits/slither/wiki/Detectors-Documentation#conformance-to-solidity-naming-conventions
```

```
TokenVesting.addRecipient (TokenVesting.sol#63-84) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool)(! isStartTimeSet || startTime > block.timestamp) (TokenVesting.sol#68)
TokenVesting.addRecipients (TokenVesting.sol#86-112) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool)(! isStartTimeSet || startTime > block.timestamp) (TokenVesting.sol#91)
TokenVesting.setStartTime (TokenVesting.sol#114-123) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool)(! isStartTimeSet || startTime > block.timestamp) (TokenVesting.sol#116)
        - require(bool)(_newStartTime > block.timestamp) (TokenVesting.sol#117)
TokenVesting.vested (TokenVesting.sol#126-162) uses timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp < startTime (TokenVesting.sol#141-143)
        - ! isStartTimeSet || (_vestingSchedule.totalAmount == 0) || (block.timestamp < startTime) (TokenVesting.sol#133-139)
        - vestedAmount > _vestingSchedule.totalAmount (TokenVesting.sol#158-160)
Reference: https://github.com/trailofbits/slither/wiki/Detectors-Documentation#block-timestamp
INFO:Detectors:
Ownable.renounceOwnership (Ownable.sol#54-57) should be declared external
Ownable.transferOwnership (Ownable.sol#63-67) should be declared external
Reference: https://github.com/trailofbits/slither/wiki/Detectors-Documentation#public-function-that-could-be-declared-as-external
INFO:Detectors:
Detected issues with version pragma in TokenVesting.sol:
        - pragma solidity0.5.16 (Context.sol#3): it allows old versions
        - pragma solidity0.5.16 (IERC20.sol#3): it allows old versions
        - pragma solidity0.5.16 (Ownable.sol#3): it allows old versions
        - pragma solidity0.5.16 (Pausable.sol#3): it allows old versions
        - pragma solidity0.5.16 (SafeMath.sol#3): it allows old versions
        - pragma solidity0.5.16 (TokenVesting.sol#1): it allows old versions
```

Implementation Recommendations

Function 'Context._msgSender' (Context.sol#16-18) is not in mixedCase

Function 'Context._msgData' (Context.sol#20-23) is not in mixedCase

Variable 'FormToken._symbol' (FormToken.sol#10) is not in mixedCase

Variable 'FormToken._name' (FormToken.sol#11) is not in mixedCase

Variable 'FormToken._decimals' (FormToken.sol#12) is not in mixedCase

Variable 'FormToken._totalSupply' (FormToken.sol#13) is not in mixedCase

Function 'Context._msgSender' (Context.sol#16-18) is not in mixedCase

Function 'Context._msgData' (Context.sol#20-23) is not in mixedCase

Function 'Pausable._pause' (Pausable.sol#74-77) is not in mixedCase

Function 'Pausable._unpause' (Pausable.sol#86-89) is not in mixedCase

Parameter '_formToken' of TokenVesting. (TokenVesting.sol#42) is not in mixedCase

Parameter '_totalAmount' of TokenVesting. (TokenVesting.sol#43) is not in mixedCase

Parameter '_initialUnlock' of TokenVesting. (TokenVesting.sol#44) is not in mixedCase

Parameter '_withdrawInterval' of TokenVesting. (TokenVesting.sol#45) is not in mixedCase

Parameter '_releaseRate' of TokenVesting. (TokenVesting.sol#46) is not in mixedCase

Parameter '_newRecipient' of TokenVesting.addRecipient (TokenVesting.sol#63) is not in mixedCase

Parameter '_totalAmount' of TokenVesting.addRecipient (TokenVesting.sol#63) is not in mixedCase

Parameter '_newRecipients' of TokenVesting.addRecipients (TokenVesting.sol#87) is not in mixedCase

Parameter '_totalAmounts' of TokenVesting.addRecipients (TokenVesting.sol#88) is not in mixedCase

Parameter '_newStartTime' of TokenVesting.setStartTime (TokenVesting.sol#114) is not in mixedCase

Disclaimer

Quillhash audit is not a security warranty, investment advice, or an endorsement of the Formation Finance contract. Securing smart contracts is a multistep process; therefore, running a bug bounty program as a complement to this audit is strongly recommended.

Summary

The use case of the smart contract is very well designed and Implemented. Overall, the code is written and demonstrates effective use of abstraction, separation of concerns, and modularity. **The Formation Finance** development team demonstrated high technical capabilities, both in the design of the architecture and in the implementation.

A medium severity and some low-severity issues were reported during the initial audit. However, the Formation Finance team has now fixed them.

FORMATION.FI

QuillAudits

Canada, India, Singapore and United Kingdom

audits.quillhash.com

audits@quillhash.com