Smart Contract Audit

UsersNetworkFactory

2022.4.28

1. Background

The purpose of the audit was to achieve the following:

- Ensure that the smart contract functions as intended.
- Identify potential security issues with the smart contract.

The information in this report should be used to understand the risk exposure of the smart contract, and as a guide to improve the security posture of the smart contract by remediating the issues that were identified.

2. Project Information

- Platform: Oasis (Emerald) network
- Contract Address: 0x71f76966A0aB786276FA7015f93Bb6CF4759c38F
- Code:

https://explorer.emerald.oasis.dev/address/0x71f76966A0aB786276FA7015f93Bb6CF4759c38F/contracts

3. Executive Summary

According to our assessment, the customer`s solidity smart contract is **Secure**.

Automated checks are with remix IDE. All issues were performed by me, which included the analysis of code functionality, manual audit found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the audit overview section. The general overview is presented in the Project Information section and all issues found are located in the audit overview section.

I have found some very-low level issues in all solidity files of the contract

The files:

File in Scope:

- Contract: Factory Contract
- Inherit: IUserNetworkFactory, IUsersNetworkPair, IUserNetworkERC20
- Observation: All passed including security check
- Test Report: passed
- Score: passed
- Unit Testing: 4 passed, 1 Failed
- Conclusion: passed

4. Issue Checking Status

No.	Issue Description	Checking Status
1	Compiler Warning	✓
2	Race conditions and Reentrancy. Cross-function race conditions.	✓
3	Possible delays in data delivery.	✓
4	Oracle calls.	✓
5	Design Logic.	✓

6	Timestamp dependence.	✓
7	Integer Overflow and Underflow.	✓
8	DoS with Revert.	✓
9	DoS with block gas limit.	✓
10	Methods execution permissions.	✓
11	The impact of the exchange rate on the logic.	✓
12	Private user data leaks.	✓
13	Malicious Event log.	✓
14	Scoping and Declarations.	✓
15	Uninitialized storage pointers.	✓
16	Arithmetic accuracy.	✓

5. Contract Audit Findings

Critical:

No Critical severity vulnerabilities were found.

High:

No High severity vulnerabilities were found.

Medium: No Medium severity vulnerabilities were found

Low:

✓ Solidity Static Analysis

- Security

No.	Issue Position(row)	Issue Description
1	334:4	Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.
2	355:4	Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.
3	320:39	"block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.

4	290:44	Use of "call": should be avoided whenever possible. It can lead to unexpected behavior if return value is not handled properly. Please use Direct Calls via specifying the called contract's interface.

- Gas & Economy

No.	Issue Position(row)	Issue Description
1	355:4	If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
2	379:4	//
3	404:4	//
4	435:4	//
5	443:4	//
6	449:4	//
7	467:4	//

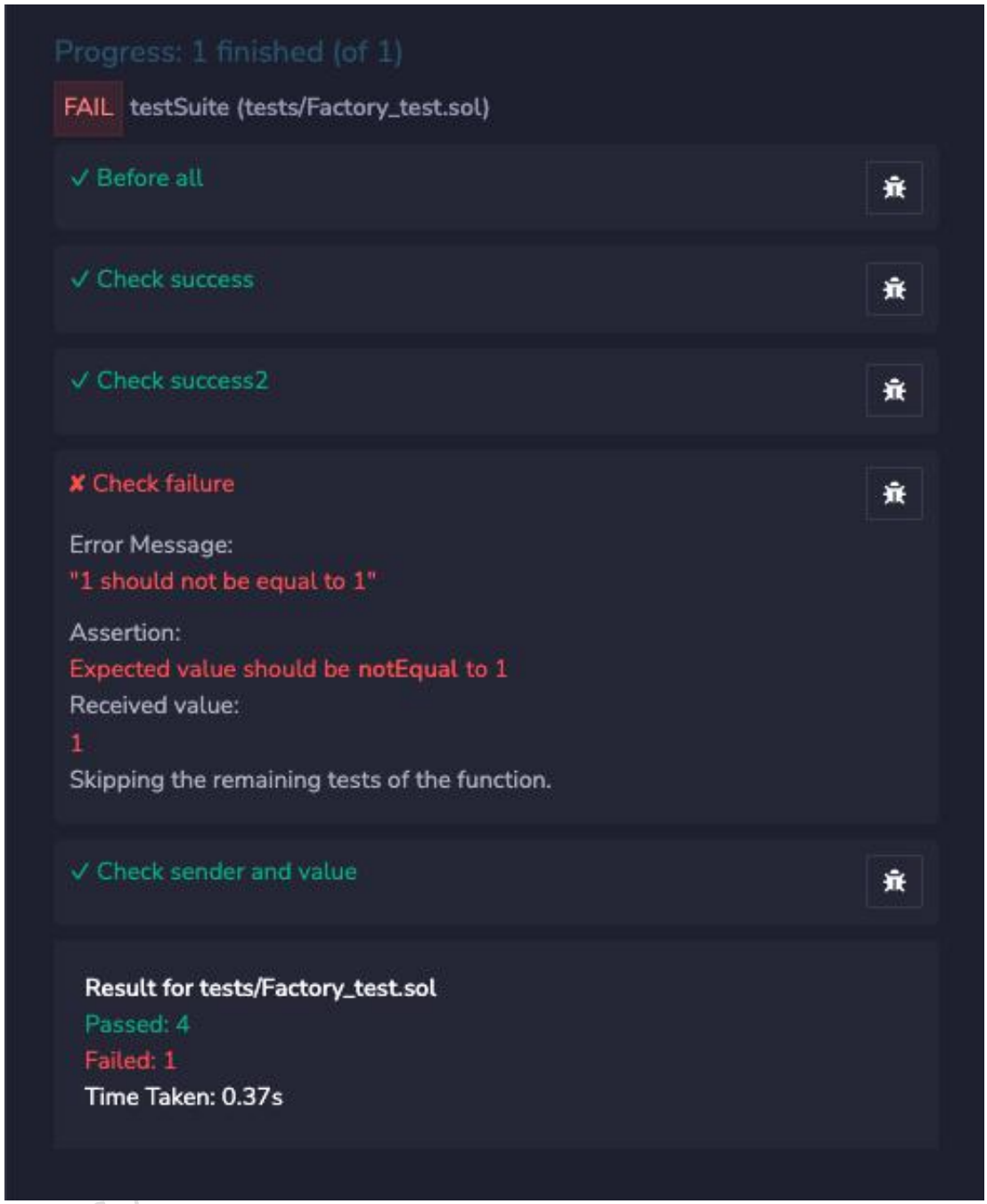
- Miscellaneous

* Found bunch of similar variable names in Factory contract
* Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component. (490:8)
* "bytes" and "string" lengths are not the same since strings are assumed to be UTF-8 encoded (according to the ABI defintion) therefore one character is not nessesarily encoded in one byte of data.
* Division of integer values yields an integer value again. That means e.g. 10 / 100 = 0 instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants. (368:33, 368:72, 390:18)

- Multiple Pragma Statement

There are multiple pragma statements in the code. Only the compiler version 0.5.16 will work with the code,
but keeping only one pragma statement helps in maintaining readability of the code.

✓ Solidity Unit Testing


Progress: 1 finished (of 1)
FAIL testSuite (tests/Factory_test.sol)
✓ Before all
✓ Check success
✓ Check success2
✘ Check failure
Error Message:
"1 should not be equal to 1"
Assertion:
Expected value should be notEqual to 1
Received value:
1
Skipping the remaining tests of the function.
✓ Check sender and value
Result for tests/Factory_test.sol
Passed: 4
Failed: 1
Time Taken: 0.37s

6. **Conclusion**

The contracts are written systematically. I have found no critical issues. As such, it is clear for production.

Since possible test cases can be unlimited, for such an extensive smart contract protocol, we provide no such guarantee of future outcomes. I have used all the latest static tools and manual observations to cover maximum possible test cases to scan everything.

Security state of the reviewed contract is “Secure”.

✓ No volatile code.

✓No high severity issues were found.