Smart Contract Audit

Wrapped ROSE (wROSE)

2022.4.30

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: 0x21C718C22D52d0F3a789b752D4c2fD5908a8A733
- Code: https://explorer.emerald.oasis.dev/address/0x21C718C22D52d0F3a789b752D4c2fD5908a8A733/contracts

3. Executive Summary

According to our assessment, the customer`s solidity smart contract is **Very 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: Wrapped ROSE (wROSE)
- Inherit:
- 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	44:4	Potential violation of Checks-Effects-Interaction pattern in WROSE.withdraw(uint256): Could potentially lead to re-entrancy vulnerability.

- Gas & Economy

No.	Issue Position(row)	Issue Description
1	19:4	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	24:4	//
3	36:4	//
4	43:4	//
5	25:4	//
6	26:4	//
7	37:4	//
8	44:4	//

- Version Checking

No.	Function Name	Issue Description
1	42:9	Invoking events without “emit” prefix is deprecated. Deposit(msg.sender, msg.value);
2	48:9	Invoking events without “emit” prefix is deprecated. Withdrawal(msg.sender, wad);
3	52:16	Using contract member “balance” inherited from the address type is deprecated. Convert the contract to “address” type to access the member, for example use “address(contract).balance” instead. Return this.balance;
4	57:9	Invoking events without “emit” prefix is deprecated. Approval(msg.sender, guy, wad);
5	79:9	Invoking events without “emit” prefix is deprecated. Transfer(src, dst, wad);

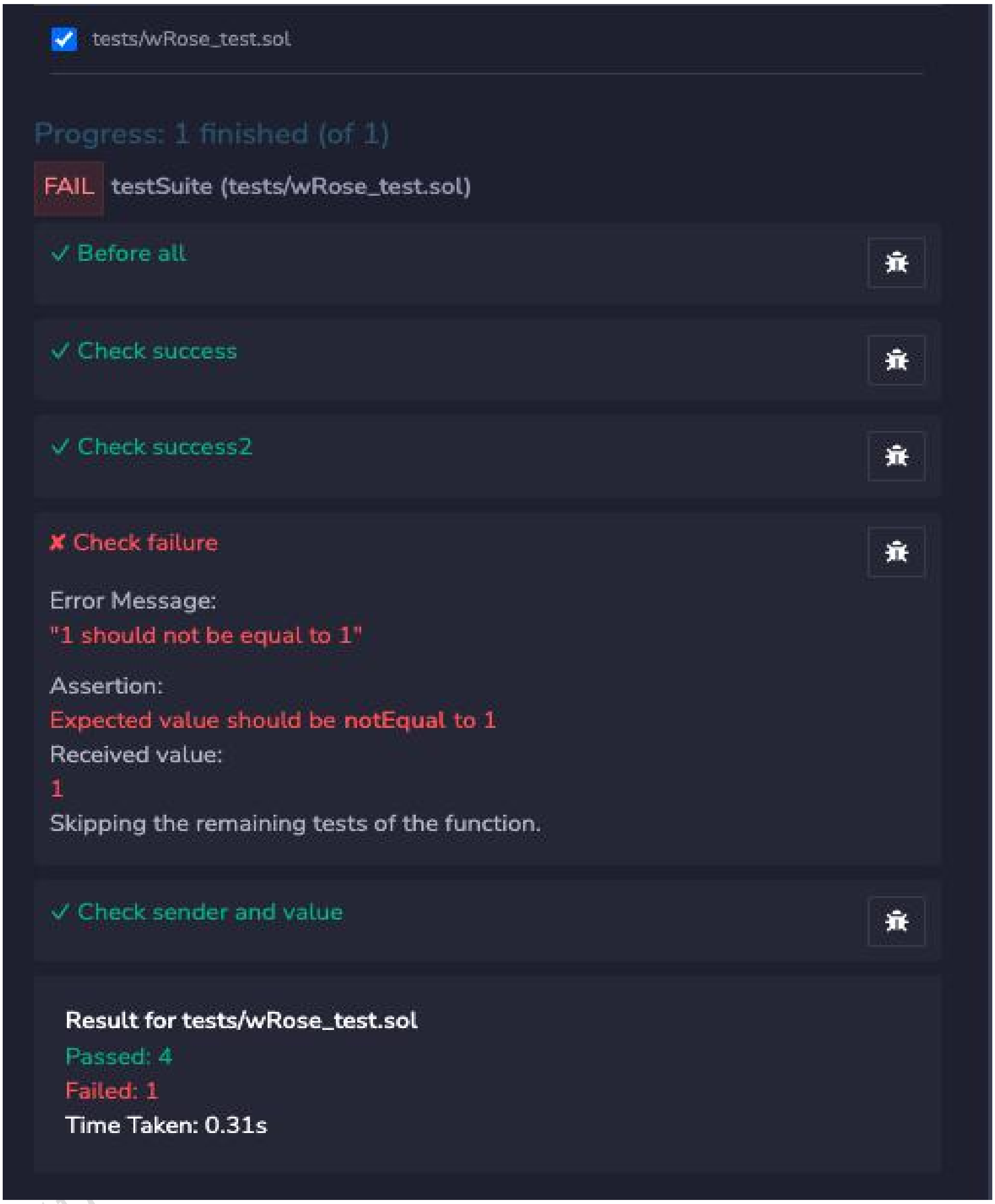
- Miscellaneous

* 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. (45:8,69:8,72:12)

- Pragma Statement

This contract is very old style, so that this contract can compile with solidity version 0.4.18. and some functions and variables are already deprecated.

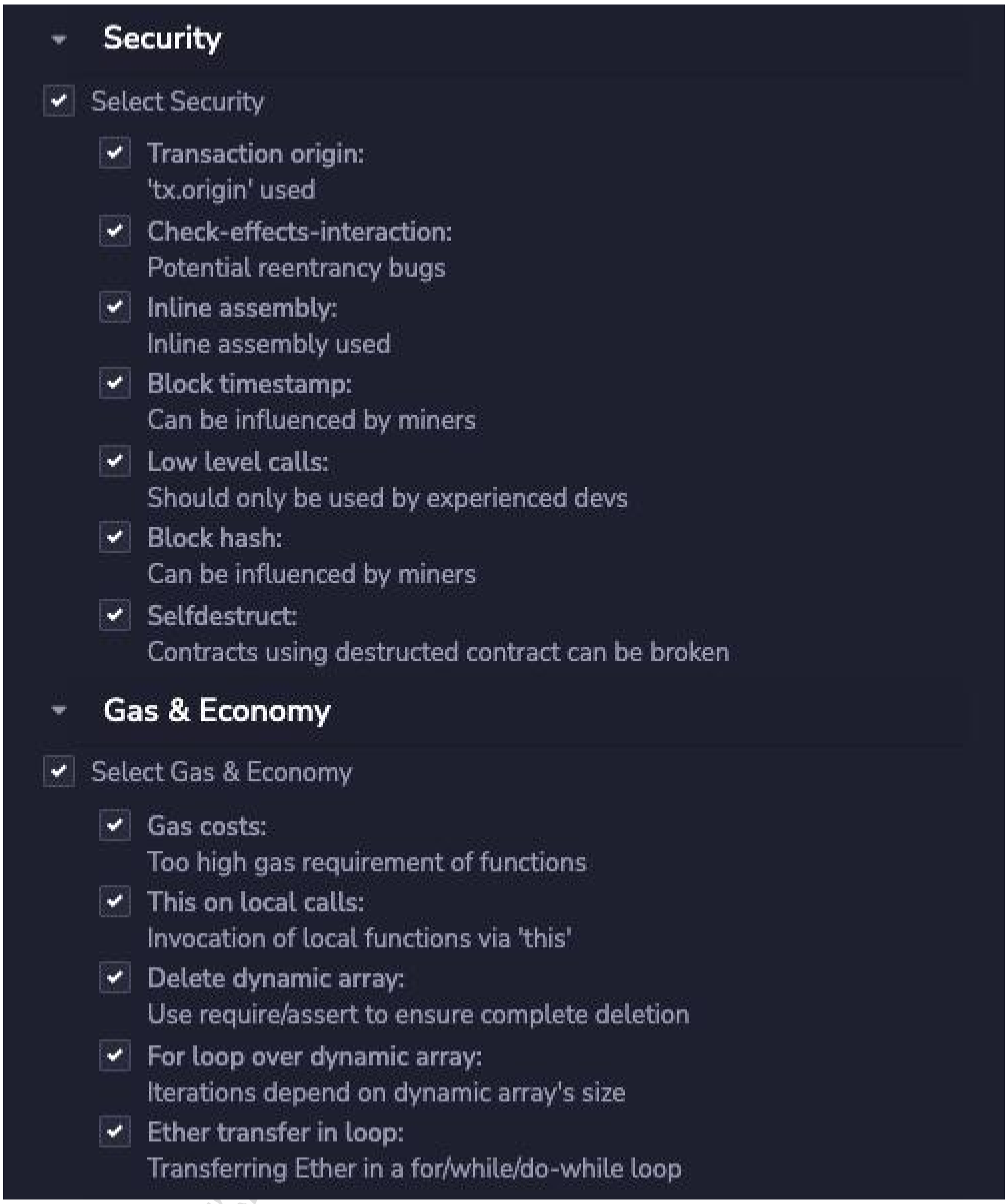
✓ Solidity Unit Testing



```
☑ tests/wRose_test.sol

Progress: 1 finished (of 1)
FAIL testSuite (tests/wRose_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/wRose_test.sol
Passed: 4
Failed: 1
Time Taken: 0.31s
```



Security

☑ Select Security

- ☑ Transaction origin:
  'tx.origin' used
- ☑ Check-effects-interaction:
  Potential reentrancy bugs
- ☑ Inline assembly:
  Inline assembly used
- ☑ Block timestamp:
  Can be influenced by miners
- ☑ Low level calls:
  Should only be used by experienced devs
- ☑ Block hash:
  Can be influenced by miners
- ☑ Selfdestruct:
  Contracts using destructed contract can be broken

Gas & Economy

☑ Select Gas & Economy

- ☑ Gas costs:
  Too high gas requirement of functions
- ☑ This on local calls:
  Invocation of local functions via 'this'
- ☑ Delete dynamic array:
  Use require/assert to ensure complete deletion
- ☑ For loop over dynamic array:
  Iterations depend on dynamic array's size
- ☑ Ether transfer in loop:
  Transferring Ether in a for/while/do-while loop

6. Conclusion

The contracts are written systematically. I have found no critical issues. As such, it is clear for production. But this contract is made by very old version.
Since possible test cases can be unlimited and developer level documentation (code flow diagram with function level description) not provided, 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 "Very Secure".

✓ No volatile code.

✓No high severity issues were found.