Cyber Unit

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Elrond
Date: December 27, 2019
Platform: Ethereum
Language: Solidity

This document may contain confidential information about IT systems and intellectual property of the customer as well as information about potential vulnerabilities and methods of their exploitation.
The report containing confidential information can be used internally by the customer or it can be disclosed publicly after all vulnerabilities fixed – upon decision of customer.

Name	Smart Contract Code Review and Security Analysis Report for Elrond
Platform	Ethereum / Solidity
Link	https://github.com/ElrondNetwork/pre-staking/
Commit version	35304c6418fe1abd72cfbca84d9f4a5b7f2e41da
Date	27.12.2019

Table of contents

Introduction

This report presents the findings of the security assessment of Customer`s smart contract and its code review conducted between December 21, 2019 – December 27, 2019.

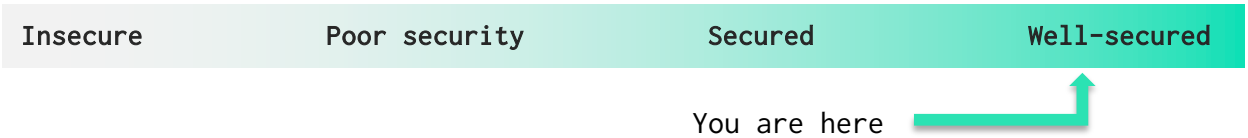
Scope

The scope of the project is Elrond smart contracts, which can be found by link below:
https://github.com/ElrondNetwork/pre-staking/
We have scanned this smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that are considered (the full list includes them but is not limited to them):

- Reentrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with (Unexpected) Throw
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Style guide violation
- Transfer forwards all gas
- ERC20 API violation
- Compiler version not fixed
- Unchecked external call - Unchecked math
- Unsafe type inference
- Implicit visibility level

Executive Summary

According to the assessment, Customer`s smart contracts are well-secured.



Our team performed analysis of code functionality, manual audit and automated checks with Mythril, Slither and remix IDE (see Appendix B pic 1-8). All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in Audit overview section. General overview is presented in AS-IS section and all found issues can be found in Audit overview section.

We found 2 low issues in smart contract. We also outline 1 best practice recommendation.

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to tokens lose etc.
High	High-level vulnerabilities are difficult to exploit; however, they also have significant impact on smart contract execution, e.g. public access to crucial functions
Medium	Medium-level vulnerabilities are important to fix; however, they can’t lead to tokens loss
Low	Low-level vulnerabilities are mostly related to outdated, unused etc. code snippets, that can’t have significant impact on execution

Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations and info statements can't affect smart contract execution and can be ignored.

AS-IS overview

The scope of the audit was mainly StakingContract.

StakingContract contract uses following libraries and smart contracts:

- SafeMath, Math, Address, Arrays, ReentrancyGuard, Pausable, IERC20 – all of them are standard libraries and smart contracts from OpenZeppelin.

StakingContract constructor - has onlyContract modifier. It sets token, rewardsAddress, launchTimestamp and currentStatus with provided values.

StakingContract has 5 modifiers:

- guardMaxStakingLimit - prevents exceeding the maximum staking limit.
- guardForPrematureWithdrawal - does not allow to withdraw prematurely.
- onlyContract - verifies that the given address is a contract address.
- onlyDuringSetup - checks if the setup hasn't been done yet.
- onlyAfterSetup - checks if the setup is already done.

StakingContract has 17 functions:

- deposit – is a public function, that has nonReentrant, onlyAfterSetup, whenNotPaused, guardMaxStakingLimit modifiers. It makes a stake deposit if the stakeDeposit.exists value is equal to false.
- initiateWithdrawal – is an external function, that has whenNotPaused, onlyAfterSetup, guardForPrematureWithdrawal modifiers. The function checks if there is a stake deposit for the address and verifies that withdrawal is not already initiated.
- executeWithdrawal – is an external function, that has nonReentrant, whenNotPaused, onlyAfterSetup modifiers. It produces previously initiated withdrawal.
- toggleRewards – is an external function, that has onlyOwner and onlyAfterSetup modifiers. Turns on the rewarding process.
- currentStakingLimit – is a public view function, that has onlyAfterSetup modifier. It returns the current staking limit.
- currentReward – is an external view function, that has onlyAfterSetup modifier. It returns the current amount of deposit and reward for the specified account.

- `getStakeDeposit` – is an external view function, that has `onlyAfterSetup` modifier. It returns all information about the message sender's stake deposit.
- `baseRewardsLength` – is an external view function, that has `onlyAfterSetup` modifier. The function returns a length of `baseRewards` array.
- `baseReward` – is an external view function, that has `onlyAfterSetup` modifier. It returns `anualRewardRate`, `lowerBound`, `upperBound` for the specified reward.
- `baseRewardHistoryLength` – is an external view function, that returns an amount of previous rewards.
- `baseRewardCheckpoint` – is an external view function, that has `onlyAfterSetup` modifier. It returns `baseRewardIndex`, `startTimestamp`, `endTimestamp` and `fromBlock` for the given index.
- `setupStakingLimit` – is an external function, that has `onlyOwner`, `whenPaused`, `onlyDuringSetup` modifiers. It configures the staking limit.
- `setupRewards` – is an external function, that has `onlyOwner`, `whenPaused`, `onlyDuringSetup` modifiers. It setups the reward rates.
- `_updateSetupState` – is a private function. It sets state to `Running` if rewards and staking are already setup.

- `_computeCurrentStakingLimit` – is a public view function, that returns a current staking limit.
- `_getIntervalsPassed` – is a public view function, that returns an amount of passed intervals.
- `_computeReward` – is a public view function, that calculates current amount of reward.

Audit overview

Critical

No critical severity vulnerabilities were found.

High

No high severity vulnerabilities were found.

Medium

No medium severity vulnerabilities were found.

Low [FIXED]

1. Compiler version is not locked. Consider locking compiler version with the latest one (see Appendix A pic 1 for evidence).
2. `currentReward` function returns "`SafeMath: multiplication overflow`" for an address that has just called a `deposit` function. It happens because `stakeDeposit.endDate` is `0` and

```
uint256 stakingPeriod = (stakeDeposit.endDate - stakeDeposit.startDate) / 1 days;
```

   expression inside `_computeRewardRatesWeightedSum` doesn't evaluate as expected. Consider adding a require statement inside `currentReward` function, that checks `endDate` value (see Appendix A pic 2 for evidence).

Lowest / Code style / Best Practice

Best Practice

3. Require statement inside **setupStakingLimit** function provides the misleading error message. It says "Some parameters are 0", when one of the parameters could be passed as zero. (see Appendix A pic 3 for evidence).

Conclusion

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. For the contract high level description of functionality was presented in As-is overview section of the report.

Audit report contains all found security vulnerabilities and other issues in the reviewed code.

Overall quality of reviewed contracts is good. Security engineers found 2 low vulnerabilities, which couldn’t have any significant security impact.

Disclaimers

Disclaimer

The smart contracts given for audit have been analyzed in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

The audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only – we recommend proceeding with several independent audits and a public bug bounty program to ensure security of smart contracts.

Technical Disclaimer

Smart contracts are deployed and executed on blockchain platform. The platform, its programming language, and other software related to the smart contract can have own vulnerabilities that can lead to hacks. Thus, the audit can’t guarantee explicit security of the audited smart contracts.

Appendix A. Evidences

Pic 1. Pragma not locked:

```
1  pragma solidity ^0.5.14;
```

Pic 2. _computeRewardRatesWeightedSum function:

```
397     function _computeRewardRatesWeightedSum(StakeDeposit memory stakeDeposit)
398     private
399     view
400     returns (uint256, uint256)
401     {
402         uint256 stakingPeriod = (stakeDeposit.endDate - stakeDeposit.startDate) / 1 days;
```

Appendix B. Automated tools reports

Pic 1. Slither automated report:

```
auditor@linux:~/Desktop/pre-staking-master/contracts$ slither StakingContract.sol
INFO:Detectors:
StakingContract._computeCurrentStakingLimit() (StakingContract.sol#356-366) uses a dangerous strict equality:
        - intervalsPassed == 0 (StakingContract.sol#362)
StakingContract._computeRewardRatesWeightedSum(StakingContract.StakeDeposit) (StakingContract.sol#397-434) uses a dangerous strict equality:
        - stakeDeposit.startCheckpointIndex == stakeDeposit.endCheckpointIndex (StakingContract.sol#407)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
StakingContract.toggleRewards(bool).index (StakingContract.sol#204) is a local variable never initialiazed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
INFO:Detectors:
StakingContract.guardMaxStakingLimit(uint256).currentStakingLimit (StakingContract.sol#86) shadows:
        - StakingContract.currentStakingLimit() (StakingContract.sol#220-227) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
Reentrancy in StakingContract.deposit(uint256) (StakingContract.sol#129-151):
        External calls:
        - require(bool,string)(token.transferFrom(msg.sender,address(this),amount),[Deposit] Something went wrong during the token transfer) (StakingContract.so
l#149)
        Event emitted after the call(s):
        - StakeDeposited(msg.sender,amount) (StakingContract.sol#150)
Reentrancy in StakingContract.executeWithdrawal() (StakingContract.sol#168-194):
        External calls:
        - require(bool,string)(token.transfer(msg.sender,amount),[Withdraw] Something went wrong while transferring your initial deposit) (StakingContract.sol#1
90)
        - require(bool,string)(token.transferFrom(rewardsAddress,msg.sender,reward),[Withdraw] Something went wrong while transferring your reward) (StakingCont
ract.sol#191)
        Event emitted after the call(s):
        - WithdrawExecuted(msg.sender,amount,reward) (StakingContract.sol#193)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
Address.isContract(address) (libs/utils/Address.sol#18-31) uses assembly
        - INLINE ASM None (libs/utils/Address.sol#29)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Different versions of Solidity is used in :
        - Version used: ['^0.5.0', '^0.5.14']
        - ^0.5.14 (StakingContract.sol#1)
        - ^0.5.14 (libs/lifecycle/Pausable.sol#1)
        - ^0.5.14 (libs/math/Math.sol#1)
        - ^0.5.14 (libs/math/SafeMath.sol#1)
        - ^0.5.14 (libs/ownership/Ownable.sol#1)
        - ^0.5.14 (libs/utils/Address.sol#1)
        - ^0.5.14 (libs/utils/Arrays.sol#1)
```

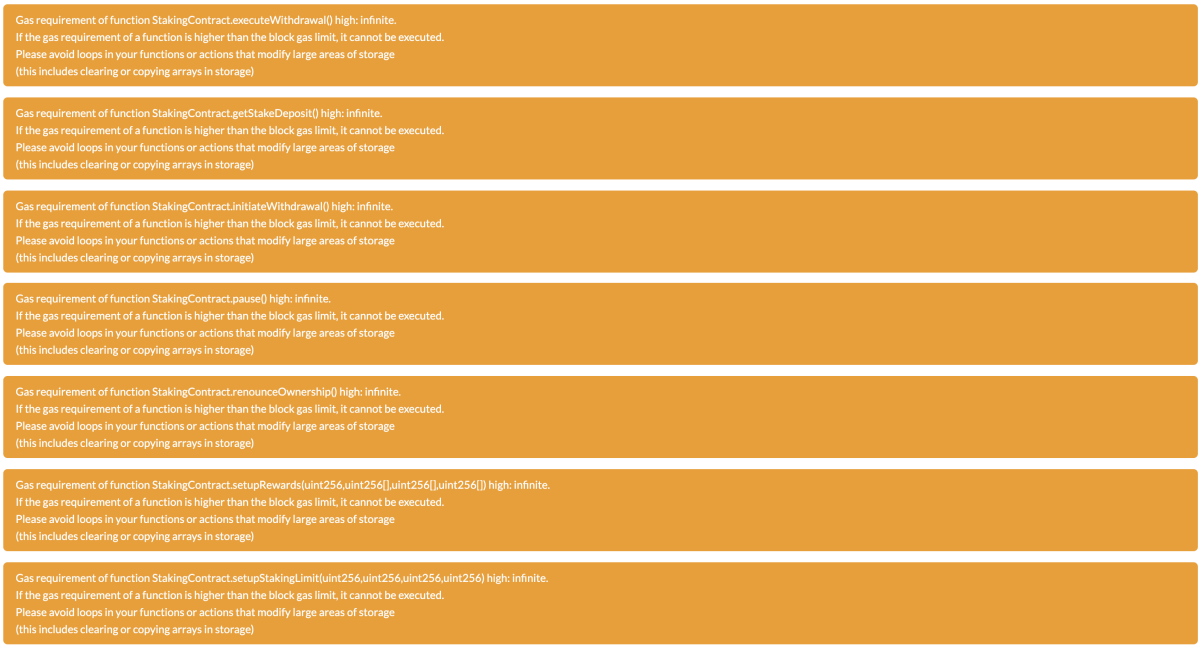
Pic 2. Slither automated report:

```
        - ^0.5.14 (libs/ownership/Ownable.sol#1)
        - ^0.5.14 (libs/utils/Address.sol#1)
        - ^0.5.14 (libs/utils/Arrays.sol#1)
        - ^0.5.14 (libs/utils/ReentrancyGuard.sol#1)
        - ^0.5.0 (token/ERC20/IERC20.sol#1)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used
INFO:Detectors:
Pragma version^0.5.14 (StakingContract.sol#1) is known to contain severe issue (https://solidity.readthedocs.io/en/v0.5.8/bugs.html)
Pragma version^0.5.14 (libs/lifecycle/Pausable.sol#1) is known to contain severe issue (https://solidity.readthedocs.io/en/v0.5.8/bugs.html)
Pragma version^0.5.14 (libs/math/Math.sol#1) is known to contain severe issue (https://solidity.readthedocs.io/en/v0.5.8/bugs.html)
Pragma version^0.5.14 (libs/math/SafeMath.sol#1) is known to contain severe issue (https://solidity.readthedocs.io/en/v0.5.8/bugs.html)
Pragma version^0.5.14 (libs/ownership/Ownable.sol#1) is known to contain severe issue (https://solidity.readthedocs.io/en/v0.5.8/bugs.html)
Pragma version^0.5.14 (libs/utils/Address.sol#1) is known to contain severe issue (https://solidity.readthedocs.io/en/v0.5.8/bugs.html)
Pragma version^0.5.14 (libs/utils/Arrays.sol#1) is known to contain severe issue (https://solidity.readthedocs.io/en/v0.5.8/bugs.html)
Pragma version^0.5.14 (libs/utils/ReentrancyGuard.sol#1) is known to contain severe issue (https://solidity.readthedocs.io/en/v0.5.8/bugs.html)
Pragma version^0.5.0 (token/ERC20/IERC20.sol#1) allows old versions
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in Address.sendValue(address,uint256) (libs/utils/Address.sol#61-67):
        - (success) = recipient.call.value(amount)() (libs/utils/Address.sol#65)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
SafeMath.TEN18 (libs/math/SafeMath.sol#17) is never used in SafeMath (libs/math/SafeMath.sol#16-158)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variables
INFO:Detectors:
deposit(uint256) should be declared external:
        - StakingContract.deposit(uint256) (StakingContract.sol#129-151)
currentStakingLimit() should be declared external:
        - StakingContract.currentStakingLimit() (StakingContract.sol#220-227)
paused() should be declared external:
        - Pausable.paused() (libs/lifecycle/Pausable.sol#38-40)
pause() should be declared external:
        - Pausable.pause() (libs/lifecycle/Pausable.sol#61-64)
unpause() should be declared external:
        - Pausable.unpause() (libs/lifecycle/Pausable.sol#69-72)
owner() should be declared external:
        - Ownable.owner() (libs/ownership/Ownable.sol#28-30)
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (libs/ownership/Ownable.sol#54-57)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (libs/ownership/Ownable.sol#63-65)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-as-external
INFO:Slither:StakingContract.sol analyzed (9 contracts with 41 detectors), 27 result(s) found
```

Pic 3. RemixIDE automated report:

Pic 4. RemixIDE automated report:

Pic 5. RemixIDE automated report:



Pic 6. RemixIDE automated report:

Pic 7. RemixIDE automated report:

Pic 8. Mythril automated report: