ENTERSOFT

Perpetual Staking AvaXlauncher

Smart Contract Audit (Final Report)

23rd September 2021

For: AvaXLauncher
Prepared By: Entersoft Pte Ltd of 1B Trengganu Street, Singapore 058455

Contents

1.0 Disclaimer 4

2.0 Overview 5

2.2 Scope 5

2.3 Project Summary 5

2.4 Audit Summary 5

2.5 Security Level references 5

2.6 Vulnerability Summary 6

2.7 Audit Results Overview 6

3.0 Executive Summary 7

3.1 Files in Scope 7

3.2 Findings 7

3.3 Comments 7

4.0 Vulnerabilities 8

4.1 Use external function modifier instead of public 8

4.2After unStake rewards got zero, update in global var 8

4.3 Public Function that could be declared external 9

4.4 Reentrancy in unStake 10

5.0 AvaXLauncher Functional Tests 11

6.0 Unit Tests 12

7.0 Automated Testing 17

7.1 Slither 17

7.2 Surya 19

8.0 Auditing Approach and Methodologies applied 23

8.1 Structural Analysis 23

8.2 Static Analysis 23

8.3 Code Review / Manual Analysis 23

8.4 Gas Consumption 23

8.5 Tools and Platforms used for Audit 23

8.6 Checked Vulnerabilities 24

9.0 Limitations on Disclosure and Use of this Report 24

Revision History and Version Control

Version	Date	Author(s)	Description
1.0	September 23rd ,2021	ES Auditors	Initial Draft of Final Report
1.0	September 23rd ,2021	Jake Lemke	Reviewed
1.0	September 23rd ,2021	Paul Kang	Released Final Report

Entersoft was commissioned by AvaXlauncher to perform source code review on their solidity smart contract. The review was conducted between September 20th 2021 to September 23rd 2021. The report is organized into the following sections.

- Executive Summary: A high-level overview of the security audit findings.
- Technical analysis: Our detailed analysis of the Smart Contract code

The information in this report should be used to understand overall code quality, security, correctness, and meaning that code will work as AvaxLauncher described in the smart contract.

1.0 Disclaimer

This is a limited audit report on our findings based on our analysis, in accordance with good industry practice as at the date of this report, in relation to: (i) smart contract best coding practices and issues in the framework and algorithms based on white paper, code, the details of which are set out in this report, (Smart Contract audit). To get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us based on what it says or does not say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the below disclaimer below – please make sure to read it in full. DISCLAIMER: By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and Entersoft Australia and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers, and other representatives) (Entersoft) owe no duty of care towards you or any other person, nor does Entersoft make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties or other terms of any kind except as set out in this disclaimer, and Entersoft hereby excludes all representations, warranties, conditions and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, Entersoft hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against Entersoft, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of use of this report, and any reliance on this report. The analysis of the Smart contract is purely based on the smart contract code shared with us alone.

2.0 Overview

2.1 Project Overview

During the period of **September 20, 2021 to September 23, 2021**– Entersoft performed security audits for **AvaXLauncher (AVXL)** smart contracts.

2.2 Scope

The scope of this audit was to analyze and document the AvaXLauncher Perpetual Staking smart contract codebase for quality, security, and correctness.

OUT-OF-SCOPE: External contracts, External Oracles, other smart contracts in the repository or imported smart contracts.

2.3 Project Summary

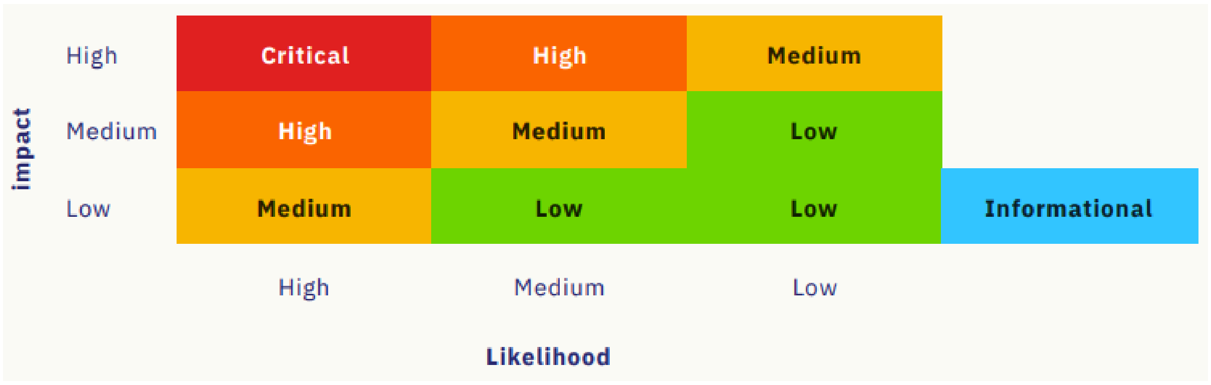
Project Name	AvaXlauncher
Platform	Avalanche-Avax
Codebase	https://bscscan.com/token/0xbd29490383edfd560426c3b63d01534408bc2da6
Token Name	AXVL
Contract Name(s)	AvaXlauncher
Contract Address	https://bscscan.com/address/0xcc5bd209c5202254d826061f4e13e2c9f57fa911
Verified	Yes
Audited	Yes
Vulnerabilities / Issues	Below

2.4 Audit Summary

Delivery Date	23rd September 2021
Method of Audit	
Consultants Engaged	1

2.5 Security Level references

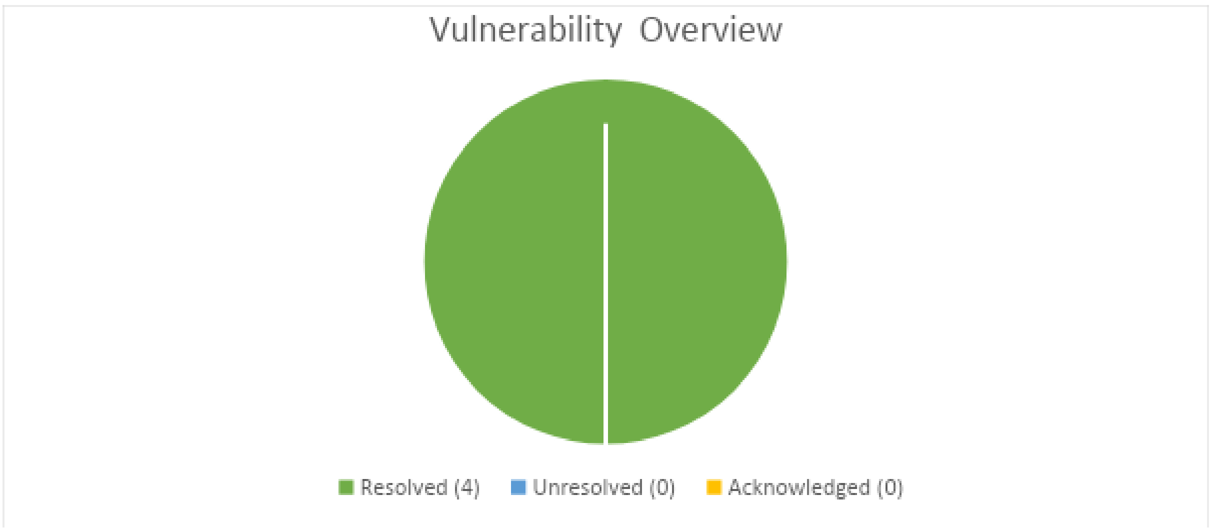
Every issue in this report was assigned a severity level from the following classification table:



impact				
High	Critical	High	Medium	
Medium	High	Medium	Low	
Low	Medium	Low	Low	Informational
	High	Medium	Low	
	Likelihood			

2.6 Vulnerability Summary

Total Critical	0
Total High	0
Total Medium	0
Total Low	2
Total Informational	2



2.7 Audit Results Overview

Audit Item	Audit Subclass	Audit Result
Overflow	-	Passed
Race Conditions	-	Passed
Permissions	Permission Vulnerability Audit	Passed
	Excessive Auditing Authority	
Safety Design	Zeppelin Safe Math	Passed
DDOS Attack	Call Function Security	Passed
Gas Optimization	-	Passed

Design Logic	-	Passed
Know Attacks	-	Passed
Overall Audit Result	-	Passed

3.0 Executive Summary

3.1 Files in Scope

3.2 Findings

ID	Title	Severity	Resolved
AXVL-001	GetRewards not updated after unstake	Low	**RESOLVED**
AXVL-002	After unStake rewards got zero, update in global var	Low	**RESOLVED**
AXVL-003	Public Function that could be declared external	Informational	**RESOLVED**
AXVL-004	Re-entrancy in unStake	Informational	**RESOLVED**

3.3 Comments

Overall, smart contracts are very well written and adhere to guidelines.

No instances of Integer Overflow and Underflow vulnerabilities or Back-Door Entry were found in the contract, but relying on other contracts might cause Reentrancy Vulnerability.

Some low severity issues were detected; it is recommended to fix them.

4.0 Vulnerabilities

4.1 Use external function modifier instead of public

Severity	Confidence	Status
Low	High	Resolved

Description:

Get rewards struct variables are not getting updated when unstake.

Remediation

Update the variable inside the scope of function

4.2After unStake rewards got zero, update in global var

Severity	Confidence	Status
Low	High	Resolved

Description:

After Unstake, total rewards pending become zero, use global variable to update rewards of a staker after unstaked by user

Remediation

Update global variable for every change in rewards stats of a user.

4.3 Public Function that could be declared external

Severity	Confidence	Status
Informational	High	Resolved

Description:

The following public functions that are never called by the contract should be declared external to save gas:

- ERC20.totalSupply (StakingContract.sol#335-337) should be declared external
- IERC20.totalSupply (StakingContract.sol#184) should be declared external
- IERC20.balanceOf (StakingContract.sol#189) should be declared external
- ERC20.balanceOf (StakingContract.sol#342-344) should be declared external
- IERC20.transfer (StakingContract.sol#198-200) should be declared external
- ERC20.transfer (StakingContract.sol#354-357) should be declared external
- ERC20.allowance (StakingContract.sol#362-368) should be declared external
- IERC20.allowance (StakingContract.sol#209-212) should be declared external
- ERC20.approve (StakingContract.sol#377-380) should be declared external
- IERC20.approve (StakingContract.sol#228) should be declared external
- ERC20.transferFrom (StakingContract.sol#394-409) should be declared external
- IERC20.transferFrom (StakingContract.sol#239-243) should be declared external
- ERC20.increaseAllowance (StakingContract.sol#423-433) should be declared external
- ERC20.decreaseAllowance (StakingContract.sol#449-462) should be declared external
- Staking.stake (StakingContract.sol#703-733) should be declared external
- Staking.unstake (StakingContract.sol#738-787) should be declared external
- Staking.totalStakers (StakingContract.sol#867-869) should be declared external
- Staking.afterUnstakeStats (StakingContract.sol#985-999) should be declared external
- **Staking.getUserStats (StakingContract.sol#1010-1043) should be declared external**

Remediation

Use the external attribute for functions that are never called from the contract.

4.4 Reentrancy in unStake

Severity	Confidence	Status
Informational	High	Resolved

Description

Reentrancy possibilities in unStake function

Remediation

Reentrancy in Staking.unstake (StakingContract.sol#738-787):
External calls:

- claimReward() (StakingContract.sol#756)
- State variables written after the call(s):
- rewards (StakingContract.sol#776)
- user (StakingContract.sol#779)
- user (StakingContract.sol#780)
- user (StakingContract.sol#781)
- user (StakingContract.sol#782)

5.0 AvaXLauncher Functional Tests

The following is the list of functions tested and checked for vulnerabilities during audit:

Function Name()	**Technical Result**	**Logical Result**	**Overall Result**
Read Functions()			
getUserStats	Pass	Pass	Pass
totalStaked	Pass	Pass	Pass
totalStakers	Pass	Pass	Pass
afterUnstakeStats	Pass	Pass	Pass
getRewardStats	Pass	Pass	Pass
isStakeHolders	Pass	Pass	Pass
user	Pass	Pass	Pass
Write Functions()			
stake	Pass	Pass	Pass
unStake	Pass	Pass	Pass
claim	Pass	Pass	Pass

6.0 Unit Tests

✓ Should correctly initialize constructor values of AvaxLancher token contract (75ms)

✓ Should correctly initialize constructor values of Staking contract (57ms)

✓ Should check avxl Contract address in staking contract

✓ Should check APY of Staking

✓ Should check stake contract user stats

✓ Should check stake contract user stats

✓ Should check total staked AVXL tokens using function

✓ Should check total avxl tokens staked

✓ Should check if stake holder or not , when not

✓ Should check getUserStats of accounts[1] before staking

✓ Should not be able to stake when doesnt have avxl token (70ms)

✓ Should not be able to claim tokens before stake (50ms)

✓ Should not be able to unstake tokens before stake (55ms)

✓ Should check a AVXL balance of a Contract address Staking

✓ Should be able to transfer AVXL to staking contract that will be rewarded (52ms)

✓ Should be able to transfer AVXL to accounts[2] (103ms)

✓ Should check a AVXL balance of a Contract address Staking after (47ms)

✓ Should check a AVXL balance of a accounts[1]

✓ Should be able to transfer AVXL to staking contract that will be rewarded (43ms)

✓ Should check a AVXL balance of a account[1]

✓ should check approval by accounts 1 to Staking contract to spend tokens on the behalf of staking

✓ should Approve staking to spend specific tokens of accounts[1] (72ms)

✓ should check approval by accounts 0 to Staking contract to spend tokens on the behalf of staking after

✓ Should be able to stake when have avxl token (63ms)

✓ Should check total staked AVXL tokens using function after staked by accounts[1]

✓ Should check total avxl tokens staked, after staked by accounts[1]

✓ Should check if stake holder or not , after staked by accounts[1]

✓ Should check stake contract user stats after staking by accounts[1]

✓ Should check a AVXL balance of a Contract address Staking after staked by accounts[1]

✓ Should check a AVXL balance of a accounts[1]

✓ Should check getUserStats of accounts[1] after staking

✓ Should not be able to claim tokens after stake, before vesting of claimable tokens (45ms)

✓ Should be able to unstake tokens before stake (67ms)

✓ Should check unStake UserData

✓ Should not be able to stake when already staked (49ms)

✓ Should check total stakers AVXL tokens using function after staked by accounts[1]

✓ Should check total avxl tokens staked, after staked by accounts[1]

✓ Should check if stake holder or not , after staked by accounts[1]

✓ Should check stake contract user stats after staking by accounts[1]

✓ Should check a AVXL balance of a Contract address Staking after staked by accounts[1]

✓ Should check a AVXL balance of a accounts[1]

✓ Should check getUserStats of accounts[1] after staking (44ms)

✓ Should not be able to claim tokens after unstake and no claimable tokens (48ms)

✓ Should check a AVXL balance of a account[2]

✓ should check approval by accounts 2 to Staking contract to spend tokens on the behalf of staking

✓ should Approve staking to spend specific tokens of accounts[1] (55ms)

✓ should check approval by accounts 0 to Staking contract to spend tokens on the behalf of staking after

✓ Should be able to stake when have avxl token (63ms)

✓ Should check total staked AVXL tokens using function after staked by accounts[2]

✓ Should check total avxl tokens staked, after staked by accounts[2]

✓ Should check if stake holder or not , after staked by accounts[2]

✓ Should check stake contract user stats

✓ Should check stake contract user stats after staking by accounts[2]

✓ Should check a AVXL balance of a Contract address Staking after staked by accounts[2]

✓ Should check a AVXL balance of a accounts[1]

✓ Should check getUserStats of accounts[2] after staking

✓ Should not be able to claim tokens after stake, before vesting of claimable tokens (66ms)

✓ Should check getUserStats of accounts[4] to know status, when not staked

✓ Should be able to increase time to get 37 days

✓ Should check Rewards stats of a accounts[2] after 38 days of staking

✓ Should be able to claim tokens after stake of 37 days (51ms)

✓ Should check a AVXL balance of a Contract address Staking after 1st week, claimed by accounts[2]

✓ Should check a AVXL balance of a accounts[2] after claiming 1st week tokens

✓ Should check Rewards stats of a accounts[2] after 38 days of staking and after claiming (106ms)

✓ Should check stake contract user stats after staking by accounts[2] and after claiming

✓ Should not be able to claim tokens after stake, before vesting of claimable tokens (68ms)

✓ Should be able to increase time to get 7 days

✓ Should check Rewards stats of a accounts[2] after 44 days of staking (53ms)

✓ Should be able to increase time to get 7 days

✓ Should check Rewards stats of a accounts[2] after 51 days of staking (41ms)

✓ Should be able to claim tokens after stake of 51 days (94ms)

✓ Should check Rewards stats of a accounts[2] after 51 days of staking (48ms)

✓ Should check a AVXL balance of a Contract address Staking after 1st week, claimed by accounts[2]

✓ Should check a AVXL balance of a accounts[2]

✓ Should not be able to claim tokens after stake, before vesting of claimable tokens (63ms)

✓ Should be able to unstake tokens after claiming tokens of 3 weeks and total 7 weeks staking complete (112ms)

✓ Should check Rewards stats of a accounts[2] after 51 days of staking and then unstake (38ms)

✓ Should be able to increase time to get 100 days

✓ Should check Rewards stats of a accounts[2] after 51 days of staking and then unstake

✓ Should be able to claim tokens after stake of 51 days (43ms)

✓ Should check a AVXL balance of a Contract address Staking after unstake

✓ Should check a AVXL balance of a accounts[2]

✓ Should check Rewards stats of a accounts[2] after 51 days of staking and then unstake and claimed all

✓ Should not be able to claim tokens after stake of 51 days

✓ Should check unStake UserData

✓ Should check a AVXL balance of a account[3]

✓ should check approval by accounts 3 to Staking contract to spend tokens on the behalf of staking

✓ should Approve staking to spend specific tokens of accounts[3] (39ms)

✓ Should be able to transfer AVXL to accounts[3] (45ms)

✓ Should check a AVXL balance of a account[3]

✓ Should check stake contract user stats of accounts[3] before stake

✓ Should check total staked AVXL tokens using function

✓ Should check total avxl tokens staked

✓ Should check stake contract user stats

✓ Should check unStake UserData before staking of accounts[2]

✓ Should not be able to claim tokens before stake by accounts[3] (39ms)

✓ Should not be able to unstake tokens before stake (41ms)

✓ Should not be able to stake less then 1 avxl token

✓ Should be able to stake when have avxl token from accounts 3 only 1 (51ms)

✓ Should check total staked AVXL tokens using function after staked by accounts[3]

✓ Should check total avxl tokens staked, after staked by accounts[3]

✓ Should check if stake holder or not , after staked by accounts[3]

✓ Should check stake contract user stats after staking by accounts[3]

✓ Should check a AVXL balance of a account[4]

✓ should check approval by accounts 4 to Staking contract to spend tokens on the behalf of staking

✓ should Approve staking to spend specific tokens of accounts[4] (43ms)

✓ Should be able to transfer AVXL to accounts[3] (56ms)

✓ Should check a AVXL balance of a account[4]

✓ Should check stake contract user stats of accounts[4] before stake

✓ Should check total staked AVXL tokens using function

✓ Should check total avxl tokens staked

✓ Should check stake contract user stats

✓ Should check unStake UserData before staking of accounts[4]

✓ Should not be able to claim tokens before stake by accounts[4] (75ms)

✓ Should not be able to unstake tokens before stake (85ms)

✓ Should not be able to stake less then 1 avxl token (44ms)

✓ Should be able to stake when have avxl token from accounts 3 only 1 (63ms)

✓ Should check total staked AVXL tokens using function after staked by accounts[3]

✓ Should check total avxl tokens staked, after staked by accounts[4]

✓ Should check if stake holder or not , after staked by accounts[4]

✓ Should check stake contract user stats after staking by accounts[4]

✓ Should be able to increase time to get 400 days

✓ Should check stake contract user stats

125 passing (3s)

0 Failed

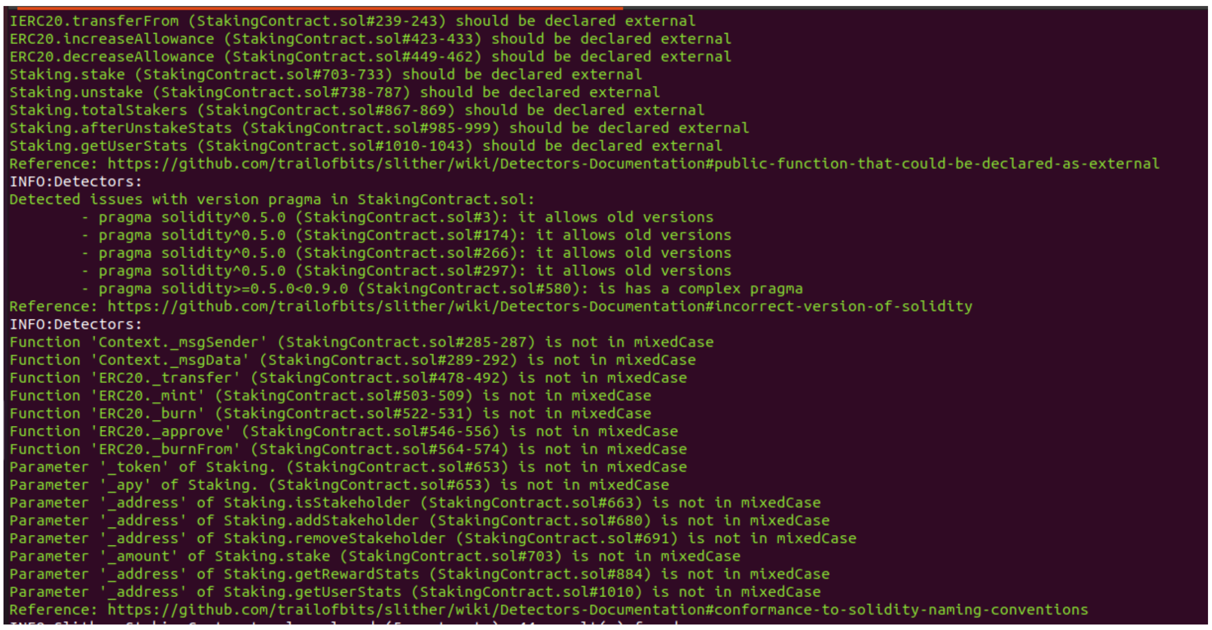
7.0 Automated Testing

Automated testing is carried out with the following tools:

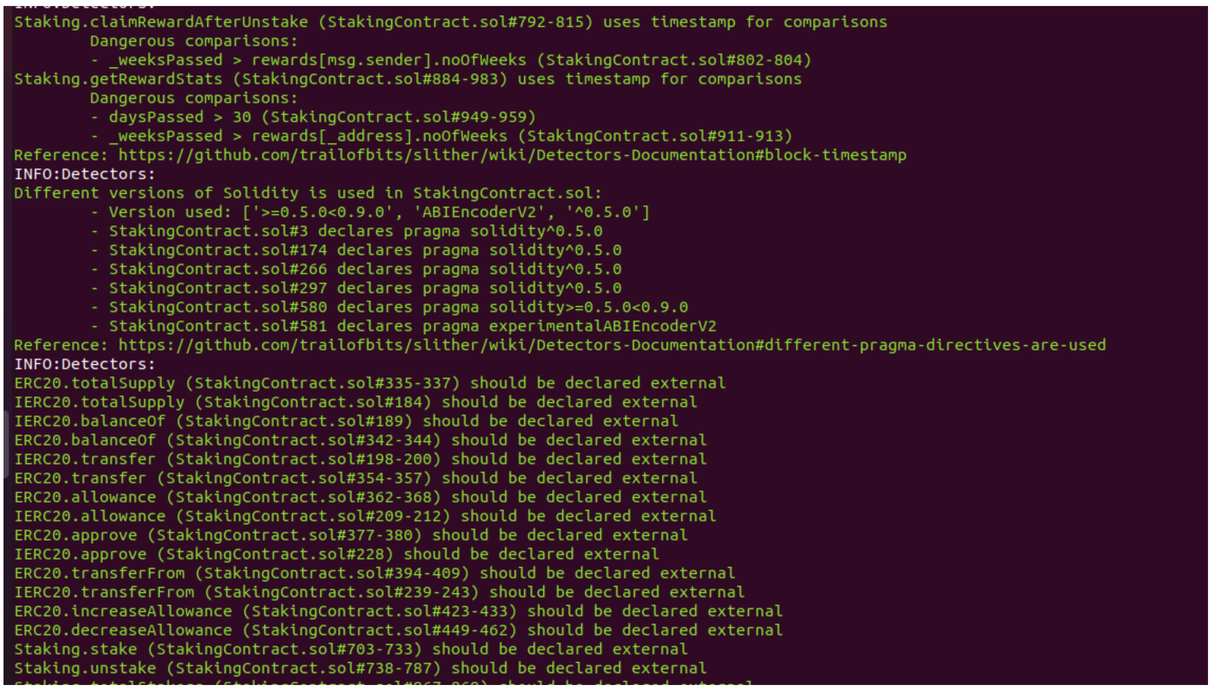
- Slither
- Mythril
- Echidna
- Manticore

7.1 Slither

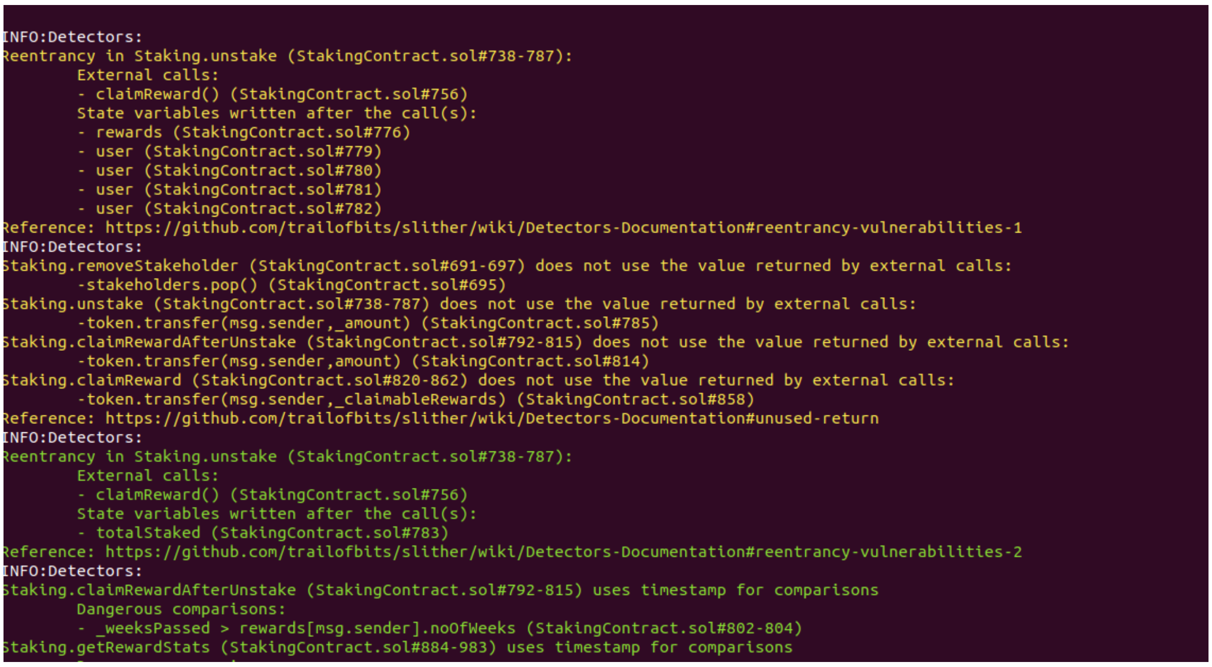
Slither is an open-source Solidity static analysis framework. This tool provides rich information about Ethereum smart contracts and has the critical properties. It runs a suite of vulnerability detectors, prints visual information about contract details, and provides an API to easily write custom analyses.



```
IERC20.transferFrom (StakingContract.sol#239-243) should be declared external
ERC20.increaseAllowance (StakingContract.sol#423-433) should be declared external
ERC20.decreaseAllowance (StakingContract.sol#449-462) should be declared external
Staking.stake (StakingContract.sol#703-733) should be declared external
Staking.unstake (StakingContract.sol#738-787) should be declared external
Staking.totalStakers (StakingContract.sol#867-869) should be declared external
Staking.afterUnstakeStats (StakingContract.sol#985-999) should be declared external
Staking.getUserStats (StakingContract.sol#1010-1043) 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 StakingContract.sol:
        - pragma solidity^0.5.0 (StakingContract.sol#3): it allows old versions
        - pragma solidity^0.5.0 (StakingContract.sol#174): it allows old versions
        - pragma solidity^0.5.0 (StakingContract.sol#266): it allows old versions
        - pragma solidity^0.5.0 (StakingContract.sol#297): it allows old versions
        - pragma solidity>=0.5.0<0.9.0 (StakingContract.sol#580): is has a complex pragma
Reference: https://github.com/trailofbits/slither/wiki/Detectors-Documentation#incorrect-version-of-solidity
INFO:Detectors:
Function 'Context._msgSender' (StakingContract.sol#285-287) is not in mixedCase
Function 'Context._msgData' (StakingContract.sol#289-292) is not in mixedCase
Function 'ERC20._transfer' (StakingContract.sol#478-492) is not in mixedCase
Function 'ERC20._mint' (StakingContract.sol#503-509) is not in mixedCase
Function 'ERC20._burn' (StakingContract.sol#522-531) is not in mixedCase
Function 'ERC20._approve' (StakingContract.sol#546-556) is not in mixedCase
Function 'ERC20._burnFrom' (StakingContract.sol#564-574) is not in mixedCase
Parameter '_token' of Staking. (StakingContract.sol#653) is not in mixedCase
Parameter '_apy' of Staking. (StakingContract.sol#653) is not in mixedCase
Parameter '_address' of Staking.isStakeholder (StakingContract.sol#663) is not in mixedCase
Parameter '_address' of Staking.addStakeholder (StakingContract.sol#680) is not in mixedCase
Parameter '_address' of Staking.removeStakeholder (StakingContract.sol#691) is not in mixedCase
Parameter '_amount' of Staking.stake (StakingContract.sol#703) is not in mixedCase
Parameter '_address' of Staking.getRewardStats (StakingContract.sol#884) is not in mixedCase
Parameter '_address' of Staking.getUserStats (StakingContract.sol#1010) is not in mixedCase
Reference: https://github.com/trailofbits/slither/wiki/Detectors-Documentation#conformance-to-solidity-naming-conventions
```



```
Staking.claimRewardAfterUnstake (StakingContract.sol#792-815) uses timestamp for comparisons
        Dangerous comparisons:
        - _weeksPassed > rewards[msg.sender].noOfWeeks (StakingContract.sol#802-804)
Staking.getRewardStats (StakingContract.sol#884-983) uses timestamp for comparisons
        Dangerous comparisons:
        - daysPassed > 30 (StakingContract.sol#949-959)
        - _weeksPassed > rewards[_address].noOfWeeks (StakingContract.sol#911-913)
Reference: https://github.com/trailofbits/slither/wiki/Detectors-Documentation#block-timestamp
INFO:Detectors:
Different versions of Solidity is used in StakingContract.sol:
        - Version used: ['>=0.5.0<0.9.0', 'ABIEncoderV2', '^0.5.0']
        - StakingContract.sol#3 declares pragma solidity^0.5.0
        - StakingContract.sol#174 declares pragma solidity^0.5.0
        - StakingContract.sol#266 declares pragma solidity^0.5.0
        - StakingContract.sol#297 declares pragma solidity^0.5.0
        - StakingContract.sol#580 declares pragma solidity>=0.5.0<0.9.0
        - StakingContract.sol#581 declares pragma experimentalABIEncoderV2
Reference: https://github.com/trailofbits/slither/wiki/Detectors-Documentation#different-pragma-directives-are-used
INFO:Detectors:
ERC20.totalSupply (StakingContract.sol#335-337) should be declared external
IERC20.totalSupply (StakingContract.sol#184) should be declared external
IERC20.balanceOf (StakingContract.sol#189) should be declared external
ERC20.balanceOf (StakingContract.sol#342-344) should be declared external
IERC20.transfer (StakingContract.sol#198-200) should be declared external
ERC20.transfer (StakingContract.sol#354-357) should be declared external
ERC20.allowance (StakingContract.sol#362-368) should be declared external
IERC20.allowance (StakingContract.sol#209-212) should be declared external
ERC20.approve (StakingContract.sol#377-380) should be declared external
IERC20.approve (StakingContract.sol#228) should be declared external
ERC20.transferFrom (StakingContract.sol#394-409) should be declared external
IERC20.transferFrom (StakingContract.sol#239-243) should be declared external
ERC20.increaseAllowance (StakingContract.sol#423-433) should be declared external
ERC20.decreaseAllowance (StakingContract.sol#449-462) should be declared external
Staking.stake (StakingContract.sol#703-733) should be declared external
Staking.unstake (StakingContract.sol#738-787) should be declared external
```

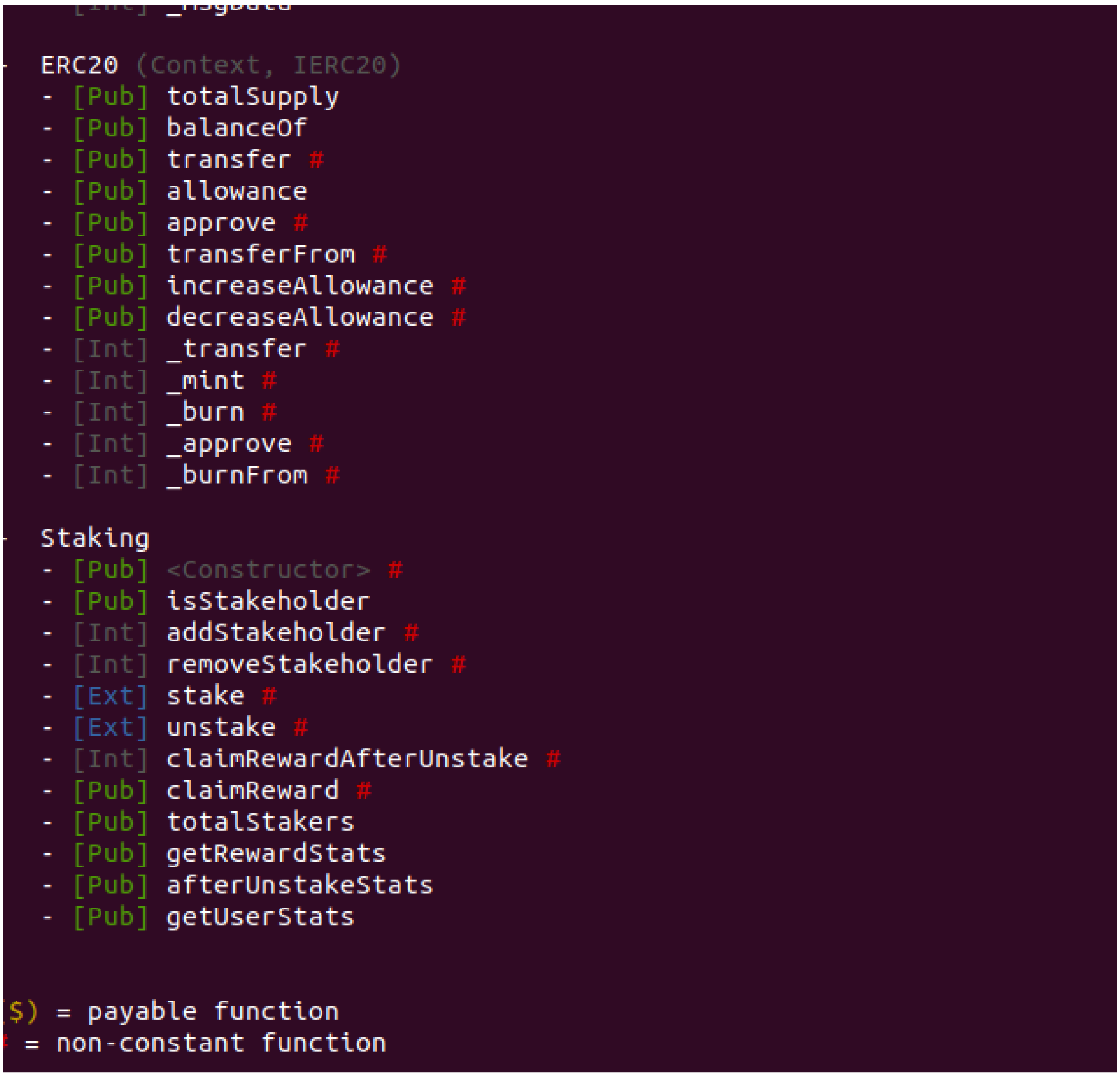


```
INFO:Detectors:
Reentrancy in Staking.unstake (StakingContract.sol#738-787):
        External calls:
        - claimReward() (StakingContract.sol#756)
        State variables written after the call(s):
        - rewards (StakingContract.sol#776)
        - user (StakingContract.sol#779)
        - user (StakingContract.sol#780)
        - user (StakingContract.sol#781)
        - user (StakingContract.sol#782)
Reference: https://github.com/trailofbits/slither/wiki/Detectors-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
Staking.removeStakeholder (StakingContract.sol#691-697) does not use the value returned by external calls:
        -stakeholders.pop() (StakingContract.sol#695)
Staking.unstake (StakingContract.sol#738-787) does not use the value returned by external calls:
        -token.transfer(msg.sender,_amount) (StakingContract.sol#785)
Staking.claimRewardAfterUnstake (StakingContract.sol#792-815) does not use the value returned by external calls:
        -token.transfer(msg.sender,amount) (StakingContract.sol#814)
Staking.claimReward (StakingContract.sol#820-862) does not use the value returned by external calls:
        -token.transfer(msg.sender,_claimableRewards) (StakingContract.sol#858)
Reference: https://github.com/trailofbits/slither/wiki/Detectors-Documentation#unused-return
INFO:Detectors:
Reentrancy in Staking.unstake (StakingContract.sol#738-787):
        External calls:
        - claimReward() (StakingContract.sol#756)
        State variables written after the call(s):
        - totalStaked (StakingContract.sol#783)
Reference: https://github.com/trailofbits/slither/wiki/Detectors-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
Staking.claimRewardAfterUnstake (StakingContract.sol#792-815) uses timestamp for comparisons
        Dangerous comparisons:
        - _weeksPassed > rewards[msg.sender].noOfWeeks (StakingContract.sol#802-804)
Staking.getRewardStats (StakingContract.sol#884-983) uses timestamp for comparisons
```

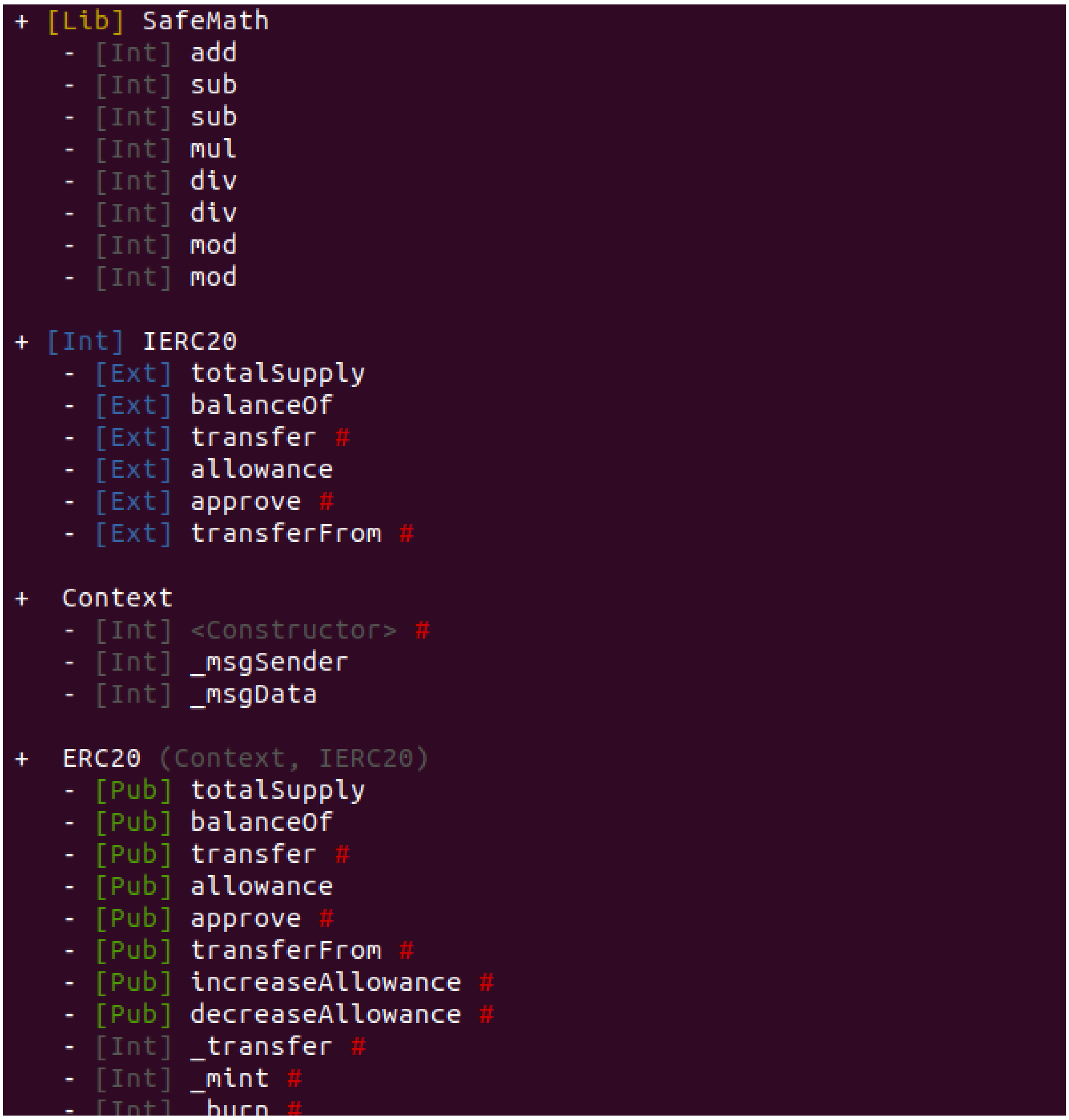
Results:

No major issues were found. Some false positive errors were reported by the tool. All the other issues have been categorized above according to their level of severity.

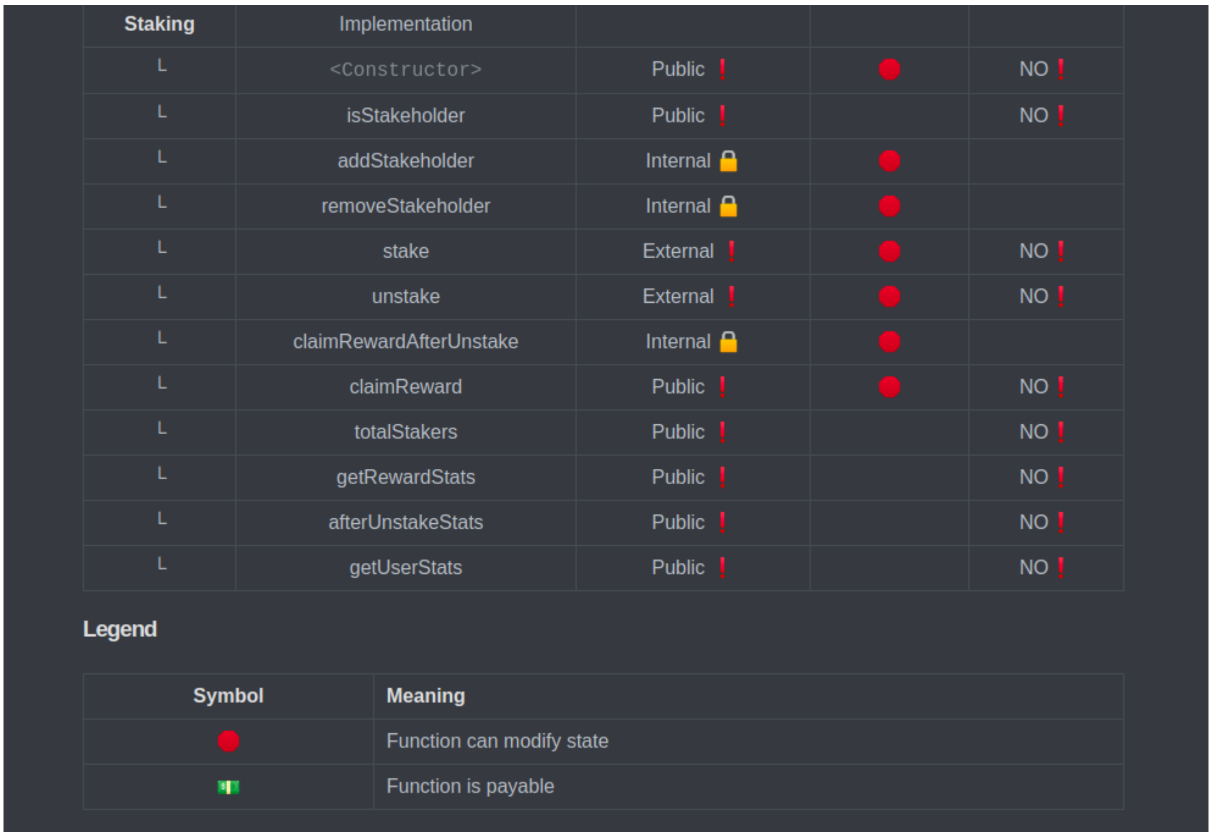
7.2 Surya



```
    [Int] _msgData

 - ERC20 (Context, IERC20)
    - [Pub] totalSupply
    - [Pub] balanceOf
    - [Pub] transfer #
    - [Pub] allowance
    - [Pub] approve #
    - [Pub] transferFrom #
    - [Pub] increaseAllowance #
    - [Pub] decreaseAllowance #
    - [Int] _transfer #
    - [Int] _mint #
    - [Int] _burn #
    - [Int] _approve #
    - [Int] _burnFrom #

 - Staking
    - [Pub] <Constructor> #
    - [Pub] isStakeholder
    - [Int] addStakeholder #
    - [Int] removeStakeholder #
    - [Ext] stake #
    - [Ext] unstake #
    - [Int] claimRewardAfterUnstake #
    - [Pub] claimReward #
    - [Pub] totalStakers
    - [Pub] getRewardStats
    - [Pub] afterUnstakeStats
    - [Pub] getUserStats


$) = payable function
 = non-constant function
```



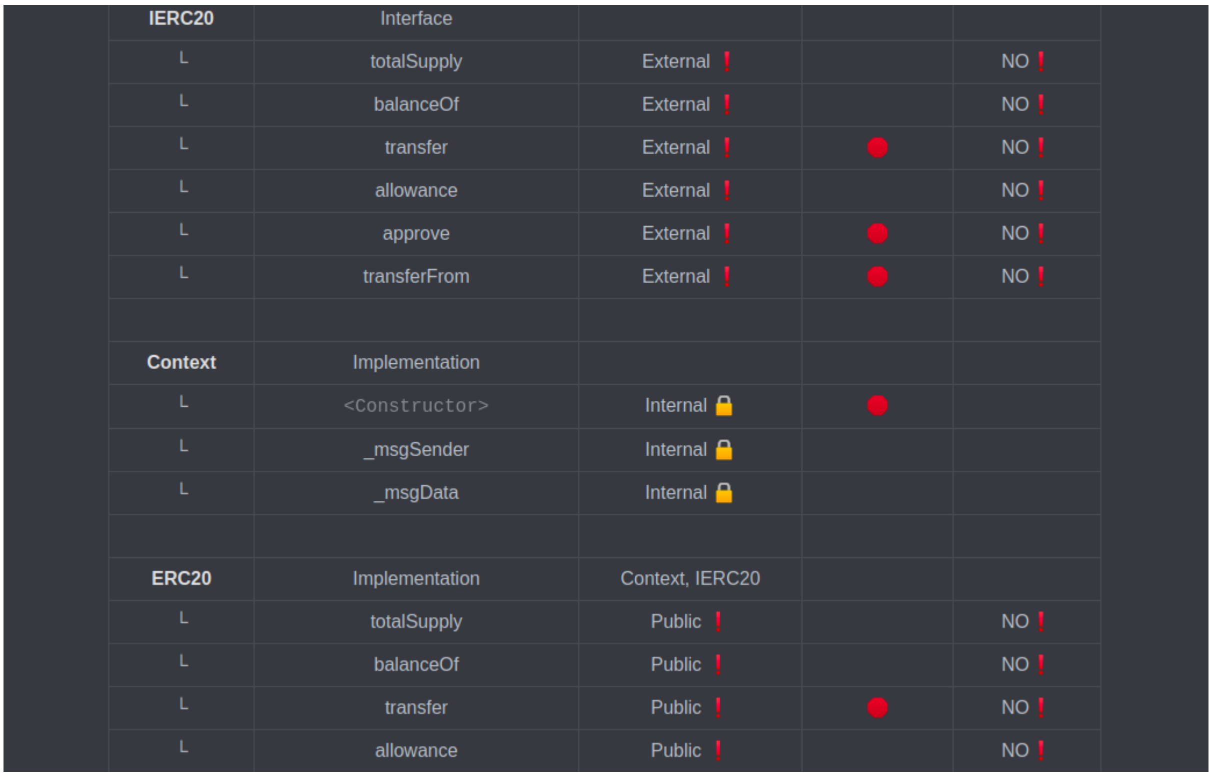
```
+ [Lib] SafeMath
   - [Int] add
   - [Int] sub
   - [Int] sub
   - [Int] mul
   - [Int] div
   - [Int] div
   - [Int] mod
   - [Int] mod

+ [Int] IERC20
   - [Ext] totalSupply
   - [Ext] balanceOf
   - [Ext] transfer #
   - [Ext] allowance
   - [Ext] approve #
   - [Ext] transferFrom #

+  Context
   - [Int] <Constructor> #
   - [Int] _msgSender
   - [Int] _msgData

+  ERC20 (Context, IERC20)
   - [Pub] totalSupply
   - [Pub] balanceOf
   - [Pub] transfer #
   - [Pub] allowance
   - [Pub] approve #
   - [Pub] transferFrom #
   - [Pub] increaseAllowance #
   - [Pub] decreaseAllowance #
   - [Int] _transfer #
   - [Int] _mint #
   - [Int] _burn #
```



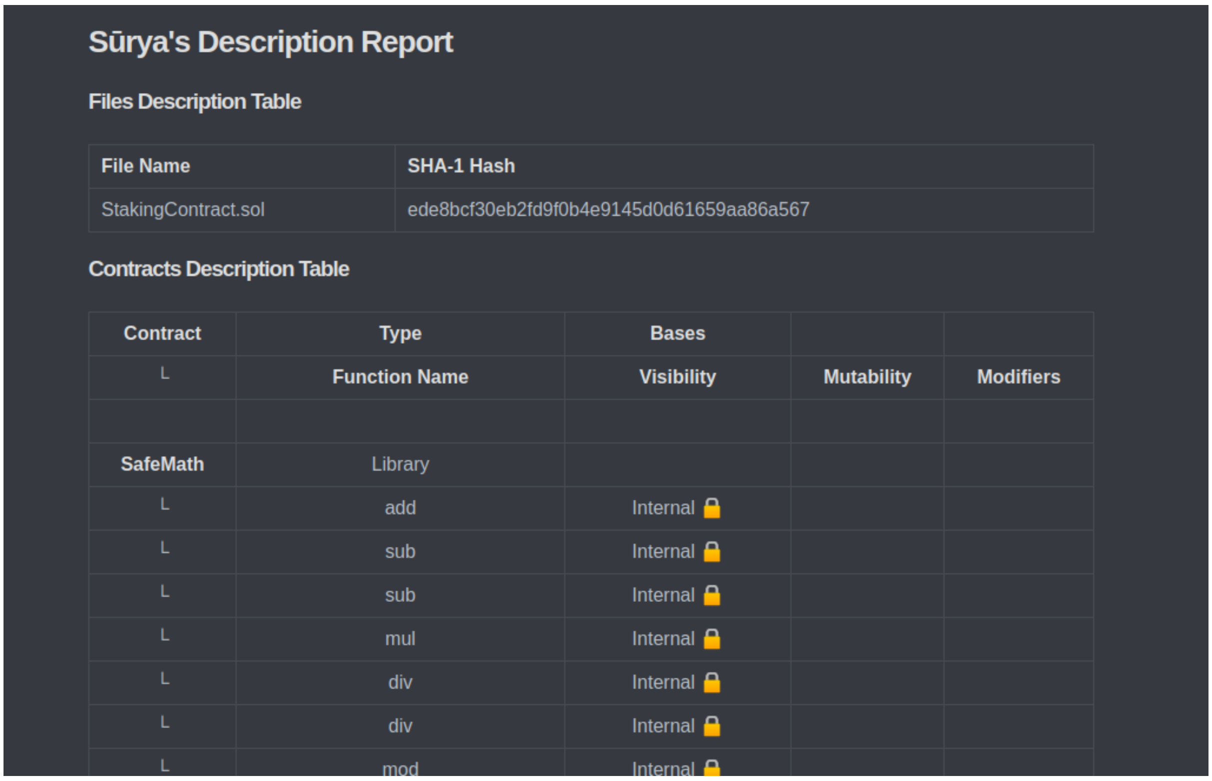
Staking	Implementation			
└	<Constructor>	Public ❗	●	NO ❗
└	isStakeholder	Public ❗		NO ❗
└	addStakeholder	Internal 🔒	●	
└	removeStakeholder	Internal 🔒	●	
└	stake	External ❗	●	NO ❗
└	unstake	External ❗	●	NO ❗
└	claimRewardAfterUnstake	Internal 🔒	●	
└	claimReward	Public ❗	●	NO ❗
└	totalStakers	Public ❗		NO ❗
└	getRewardStats	Public ❗		NO ❗
└	afterUnstakeStats	Public ❗		NO ❗
└	getUserStats	Public ❗		NO ❗

Legend

Symbol	Meaning
●	Function can modify state
💵	Function is payable



IERC20	Interface			
└	totalSupply	External ❗		NO ❗
└	balanceOf	External ❗		NO ❗
└	transfer	External ❗	●	NO ❗
└	allowance	External ❗		NO ❗
└	approve	External ❗	●	NO ❗
└	transferFrom	External ❗	●	NO ❗
Context	Implementation			
└	<Constructor>	Internal 🔒	●	
└	_msgSender	Internal 🔒		
└	_msgData	Internal 🔒		
ERC20	Implementation	Context, IERC20		
└	totalSupply	Public ❗		NO ❗
└	balanceOf	Public ❗		NO ❗
└	transfer	Public ❗	●	NO ❗
└	allowance	Public ❗		NO ❗


Sūrya's Description Report

Files Description Table

File Name	SHA-1 Hash
StakingContract.sol	ede8bcf30eb2fd9f0b4e9145d0d61659aa86a567

Contracts Description Table

Contract	Type	Bases		
└	**Function Name**	**Visibility**	**Mutability**	**Modifiers**
SafeMath	Library			
└	add	Internal 🔒		
└	sub	Internal 🔒		
└	sub	Internal 🔒		
└	mul	Internal 🔒		
└	div	Internal 🔒		
└	div	Internal 🔒		
└	mod	Internal 🔒		

8.0 Auditing Approach and Methodologies applied

Throughout the audit of **AvaXLauncher** smart contract care was taken to ensure:

- Overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per intended behaviour mentioned in whitepaper.
- Implementation of token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

A combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy regarding the scope of the smart contract audit. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of smart contracts and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the audit:

8.1 Structural Analysis

In this step we have analysed the design patterns and structure of smart contracts. A thorough check was done to ensure Smart contract is structured in a way that will not result in future problems.

8.2 Static Analysis

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

8.3 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 analysed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of automated analysis were manually verified.

8.4 Gas Consumption

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

8.5 Tools and Platforms used for Audit

VSCode, Remix IDE, Truffle, Truffle Team, Ganache, Solhint, Mythril, Manticore, Slither.

8.6 Checked Vulnerabilities

We have scanned The People Reserves smart contracts for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered:

- 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
- ERC-20 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

9.0 Limitations on Disclosure and Use of this Report

This report contains information concerning potential details of AvaXLauncher and methods for exploiting them. Entersoft recommends that special precautions be taken to protect the confidentiality of both this document and the information contained herein. Security Assessment is an uncertain process, based on past experiences, currently available information, and known threats. All information security systems, which by their nature are dependent on human beings, are vulnerable to some degree. Therefore, while Entersoft considers the major security vulnerabilities of the analyzed systems to have been identified, there can be no assurance that any exercise of this nature will identify all possible vulnerabilities or propose exhaustive and operationally viable recommendations to mitigate those exposures. In addition, the analysis set forth herein is based on the technologies and known threats as of the date of this report. As technologies and risks change over time, the vulnerabilities associated with the operation of the AvaXLauncher Smart Contract described in this report, as well as the actions necessary to reduce the exposure to such vulnerabilities will also change. Entersoft makes no undertaking to supplement or update this report based on changed circumstances or facts of which Entersoft becomes aware after the date hereof, absent a specific written agreement to perform the supplemental or updated analysis. This report may recommend that Entersoft use certain software or hardware products manufactured or maintained by other vendors. Entersoft bases these recommendations upon its prior experience with the capabilities of those products. Nonetheless, Entersoft does not and cannot warrant that a particular product will work as advertised by the vendor, nor that it will operate in the manner intended. This report was prepared by Entersoft for the exclusive benefit of AvaXLauncher and is proprietary information. The Non-Disclosure Agreement (NDA) in effect between Entersoft and AvaXLauncher govern the disclosure of this report to all other parties including product vendors and suppliers.