Retreeb

Smart Contract Audit
Final Report

November 14, 2021

Introduction 3
About Retreeb 3
About ImmuneBytes 3
Documentation Details 3
Audit Process & Methodology 4
Audit Details 4
Audit Goals 5
Security Level References 5
Found During Final Audit 6
Medium severity issues 6
Initial audit 8
High Severity Issues 8
Medium severity issues 10
Low severity issues 11
Recommendations/Informational 12
Automated Audit Result 13
Unit Test 14
Fuzz Testing 18
Vulnerability Checks 20
Concluding Remarks 21
Disclaimer 21

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Introduction

1. About Retreeb

Within a rapidly changing sector, Retreeb presents a new means of payment, simple, practical, economical, which allows it to comply with the universal values such as ethics, sharing and solidarity. It targets all persons who are part of a solidarity and sustainable approach. In consideration of their adoption of the service, Retreeb commits to its users to **pay 33% of the transaction fees collected by Retreeb** to the funding of social and environmental projects. With this business model, the technical infrastructure, the redistribution of transaction fees, and the monitoring of projects, they opt for an unprecedented level of transparency in a particularly opaque sector. Concerned about environmental issues, their technological choices are determined by a desire to reduce our carbon footprint to its strict minimum. Finally, they take a new approach to payment by placing corporate social and environmental responsibility (CSR) at the heart of their ambitions.

Visit https://retreeb.io/ to know more about.

2. About ImmuneBytes

ImmuneBytes is a security start-up to provide professional services in the blockchain space. The team has hands-on experience in conducting smart contract audits, penetration testing, and security consulting. ImmuneBytes's security auditors have worked on various A-league projects and have a great understanding of DeFi projects like AAVE, Compound, 0x Protocol, Uniswap, dydx.

The team has been able to secure 105+ blockchain projects by providing security services on different frameworks. ImmuneBytes team helps start-up with a detailed analysis of the system ensuring security and managing the overall project.

Visit http://immunebytes.com/ to know more about the services.

Documentation Details

The Retreeb team has provided the following doc for the purpose of audit:

1. https://retreeb.io/assets/retreeb-white-paper.pdf
2. Short description of the intended behaviour

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Audit Process & Methodology

ImmuneBytes team has performed thorough testing of the project starting with analyzing 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 in order to find any potential issues like Signature Replay Attacks, Unchecked External Calls, External Contract Referencing, Variable Shadowing, Race conditions, Transaction-ordering dependence, timestamp dependence, DoS attacks, and others.

In the Unit testing phase, we run unit tests written by the developer in order to verify the functions work as intended. In Automated Testing, we tested the Smart Contract with our in-house developed tools to identify vulnerabilities and security flaws.

The code was audited by a team of independent auditors which includes -

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

Audit Details

- Project Name: Retreeb
- Contracts Names: StakingPlatform, TesterStakingPlatform, Token.sol
- Languages: Solidity(Smart contract), Typescript (Unit Testing)
- Github commit hash for audit: 30471f1fe81580d56cbc2f3189e64d583cd78a85
- Github commit hash for final audit: d482164125e65a36e652d7bb5df7475bd4bcb50b
- Platforms and Tools: Remix IDE, Truffle, Truffle Team, Ganache, Solhint, VScode, Contract Library, Slither, SmartCheck, SFuzz

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

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:

1. Security: Identifying security-related issues within each contract and within the system of contracts.
2. Sound Architecture: Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.
3. Code Correctness and Quality: A full review of the contract source code. The primary areas of focus include:
    a. Correctness
    b. Readability
    c. Sections of code with high complexity
    d. Quantity and quality of test coverage

Security Level References

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

Admin/Owner Privileges can be misused either intentionally or unintentionally.

High severity issues will bring problems and should be fixed.

Medium severity issues could potentially bring problems and should eventually be fixed.

Low severity issues are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

Issues	High	Medium	Low
Open	-	2	-
Closed	2	1	1

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Found During Final Audit

Medium severity issues

1. **withdraw() function updates a User's start-time state variable twice**
   Line no - 118, 123
   Explanation:
   The withdraw() function calls the **_updateRewards()** function within itself which updates the user's claimable rewards as well as the start time for the specific user.

```
293     function _updateRewards() private {
294         _rewardsToClaim[_msgSender()] = _calculateRewards(_msgSender());
295         _userStartTime[_msgSender()] = (block.timestamp >= endPeriod)
296             ? endPeriod
297             : block.timestamp;
298     }
```

   However, **the withdraw()** function updates the user's start time once again at Line 123.

   Is this Intentional?

```
107     function withdraw(uint amount) external override {
108         require(
109             block.timestamp >= lockupPeriod,
110             "No withdraw until lockup ends"
111         );
112         require(amount > 0, "Amount must be greater than 0");
113         require(
114             amount <= staked[_msgSender()],
115             "Amount higher than stakedAmount"
116         );
117
118         _updateRewards();
119         if (_rewardsToClaim[_msgSender()] > 0) {
120             _claimRewards();
121         }
122
123         _userStartTime[_msgSender()] = block.timestamp;
124         _totalStaked -= amount;
```

   This might lead to a very unexpected scenario as the same state variable is updated twice within the function.

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Recommendation:
If the above-mentioned scenario is not intentional, it is highly recommended to reconsider function design and only update a specific state variable once, unless there is a very strong reason to include multiple modifications of the same variable.

2. **withdrawAll() function's natspec documentation doesn't match with actual function implementation:**
Line no - 133, 139
Explanation:

The NatSpec documentation of the **withdrawAll()** function mentions that the function shall only execute if **block.timestamp** is greater than the **end period**.

However, the actual function body checks a different condition, i.e., **block.timestamp >= lockupPeriod**.
The exact intended behavior of the function is not clear and this affects code readability as well might be quite confusing to the community.

Recommendation:
The function's design or the NatSpec documentation should be updated as per the exact behavior that is expected from the contract.

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Initial audit

High Severity Issues

1. **withdrawAmount() function might fail in some cases.**

   Explanation:
   The current function design of the **withdrawAmount()** function does not represent a very strong logic. The function calls the **deposit()** function within itself to re-deposit the amount of token that wasn't supposed to be withdrawn back to the pool.

   Now, this leads to a very unexpected scenario because of the way the **deposit()** function is designed currently.

```
156     function deposit(uint amount) public override {
157         require(
158             endPeriod == 0 || endPeriod > block.timestamp,
159             "Staking period ended"
160         );
161         require(
162             _totalStaked + amount <= stakingMax,
163             "Amount staked exceeds MaxStake"
164         );
165         require(amount >= 1E18, "Amount must be greater than 1E18");
166
```

   The **deposit()** function includes a strict check at line **165** to ensure that the amount passed is greater than **1e18**.

   However, when the deposit function is called from the **withdrawAmount()**, then the **amount** argument passed for calling **deposit()** function might not be equal to or greater than **1e18**, as it entirely depends on the amount being withdrawn by the user.

   In such cases, the **require** statement at **Line 165** of the **deposit()** function might revert back thus preventing the whole withdrawal operation.

   Recommendation:
   1. It is strongly recommended to modify the **withdrawAmount()** function to handle each and every scenario that might occur during the execution of the smart contract.
   2. It's also recommended to add proper test cases covering all conditions for this function.

   Amended (November 14th 2021): Issue was fixed by the **Retreeb** team and is no longer present in commit [d482164125e65a36e652d7bb5df7475bd4bcb50b]

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

2. **Missing imperative validation in withdrawAmount() function**

Explanation:
As per the NatSpec annotations of the **withdrawAmount()** function in the contract, the function should only be called when the **block.timestamp** is greater than or equal to the **end period**.

```
/**
 * @notice function that allows a user to withdraw its initial deposit
 * @dev must be called only when `block.timestamp` >= `endPeriod`
 * @dev `block.timestamp` higher than `lockupPeriod` (lockupPeriod finished)
 * @param amount, amount to withdraw
 * withdraw reset all states variable for the `msg.sender` to 0, and claim rewards
 * if rewards to claim
 */
```

However, no such validations were found in the function.

Moreover, the only check similar to it is available in the withdraw() function at lines 187 to 190 which checks the current time against the **lockupPeriod** instead of **endPeriod**.

It seems the **withdrawAmount()** function somehow assumes that the **endPeriod** and **lockPeriod** are always similar values while it might not be true.

As a result, there can be different types of pools like **mid or quick pools**, where the values for lock period and end period of the stakes are not the same.

Recommendation:

1. Adequate validations must be implemented in the functions.

2. Additionally, It's imperative clearly document the intended behavior of the function. The NatSpec documentation of the withdraw() function also doesn't resemble the function design. The documentations should either be modified or the functions should be redesigned to match the expected behavior.

Amended (November 14th 2021): Issue was fixed by the **Retreeb** team and is no longer present in commit [d482164125e65a36e652d7bb5df7475bd4bcb50b]

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Medium severity issues

1. **withdrawAmount() function includes inadequate logic**

Explanation:
The **withdrawAmount()** function of the protocol doesn't represent a very strong logic as per the current architecture.

```
function withdrawAmount(uint amount) external override {
    uint userStaking = staked[_msgSender()];
    uint result = userStaking - amount;
    withdraw();
    deposit(result);
}
```

The **deposit()** function that is called at the end of this function will require additional approval of tokens from the user in specific cases.

For instance, consider the following scenario:

A. User approves **1000 tokens** staking contract
B. Calls **deposit()** function for 1000 tokens.
C. Current approval of the User for Staking contract becomes ZER0
D. User then decides to call the **withdrawAmount()** function for only **200 Tokens.**
E. This function actually transfers back all the Staked amount back to the user, i.e., 1000 Tokens.
F. Then this function calls the **deposit()** function for re-depositing the remaining unwithdrawn tokens, i.e., **800 Tokens**.

However, for the **deposit()** to work effectively, the user needs to first approve 800 tokens to the staking contract again so that the staking contract uses the **transferFrom()** function and transfers the tokens from the user to the contract. Otherwise, the **transferFrom()** function shall revert thus causing this whole operation to revert back.

Hence, due to the current design of the **withdrawAmount()** function, users might have to approve tokens even at the time of withdrawing tokens which is not really a very standard smart contract practice.

Recommendation:
It is recommended to redesign this function and add effective test cases for the same.

Amended (November 14th 2021): Issue was fixed by the **Retreeb** team and is no longer present in commit [d482164125e65a36e652d7bb5df7475bd4bcb50b]

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Low severity issues

1. **StartStaking event should emit out all imperative arithmetic state variables**
   Line no - 62

   Explanation:
   The **StartStaking** event emits out the **startPeriod** as well as the **endPeriod** state variables, while it doesn't emit the **lockPeriod** value.

```
57      function startStaking() external override onlyOwner {
58          require(startPeriod == 0, "Staking has already started");
59          startPeriod = block.timestamp;
60          lockupPeriod = block.timestamp + lockupDuration;
61          endPeriod = block.timestamp + stakingDuration;
62          emit StartStaking(startPeriod, endPeriod);
63      }
```

   It's considered standard practice to emit all crucial arithmetic state variables when they are initialized or modified.

   Recommendation:
   It is recommended to emit out events whenever crucial state variables are modified.

   Amended (November 14th 2021): Issue was fixed by the **Retreeb** team and is no longer present in commit [d482164125e65a36e652d7bb5df7475bd4bcb50b]

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Recommendations/Informational

1. **Input validations and checkpoints can be improved in the withdrawAmount() function**

   Explanation:
   As per the current architecture of the protocol, the **withdrawAmount()** function doesn't impose an adequate validation on the **amount** argument being passed to the function.

   Moreover, it doesn't include an imperative checkpoint that ensures that only valid **stakers**, with a staked value greater than zero, call this function. This will result in a badly handled event as the users might get unclear error messages if the function reverts due to a subtraction error.

   Recommendation:
   It's imperative to include all necessary input validations and strict conditions within a function.
   Amended (November 14th 2021): Issue was fixed by the **Retreeb** team and is no longer present in commit d482164125e65a36e652d7bb5df7475bd4bcb50b

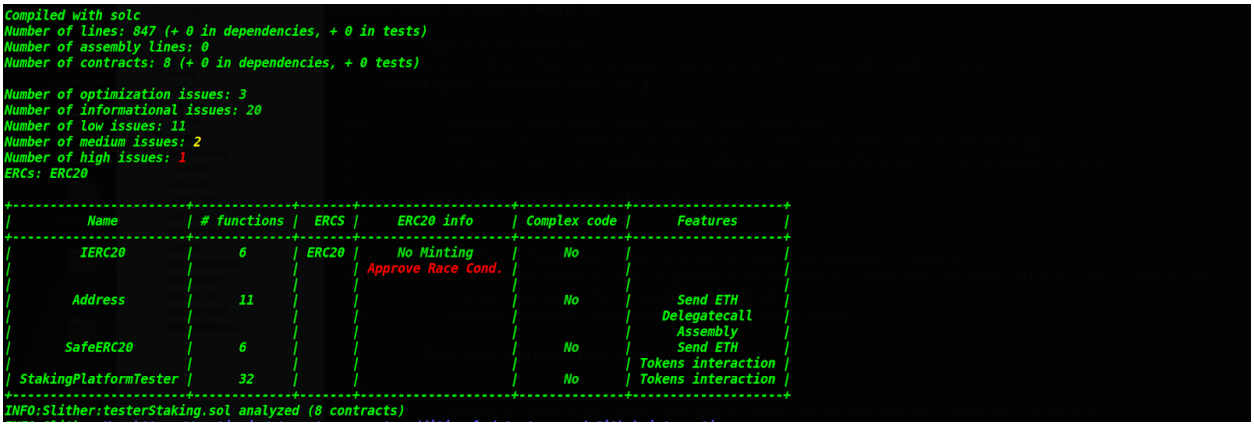
This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Automated Audit Result

1. StakingPlatform.sol

```
Compiled with solc
Number of lines: 819 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 7 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 3
Number of informational issues: 19
Number of low issues: 11
Number of medium issues: 2
Number of high issues: 1
ERCs: ERC20

+----------------+-------------+-------+-------------------+-------------+--------------------+
|      Name      | # functions | ERCS  |     ERC20 info    | Complex code |      Features      |
+----------------+-------------+-------+-------------------+-------------+--------------------+
|     IERC20     |      6      | ERC20 |     No Minting    |      No     |                    |
|                |             |       | Approve Race Cond. |             |                    |
|                |             |       |                   |             |                    |
|    Address     |      11     |       |                   |      No     |      Send ETH      |
|                |             |       |                   |             |    Delegatecall    |
|                |             |       |                   |             |      Assembly      |
|   SafeERC20    |      6      |       |                   |      No     |      Send ETH      |
|                |             |       |                   |             | Tokens interaction |
| StakingPlatform |     28      |       |                   |      No     |      Send ETH      |
|                |             |       |                   |             | Tokens interaction |
+----------------+-------------+-------+-------------------+-------------+--------------------+
INFO:Slither:stakingFlat.sol analyzed (7 contracts)
```

2. TesterStakingPlatform.sol



```
Compiled with solc
Number of lines: 847 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 8 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 3
Number of informational issues: 20
Number of low issues: 11
Number of medium issues: 2
Number of high issues: 1
ERCs: ERC20

+----------------------+-------------+-------+-------------------+-------------+--------------------+
|         Name         | # functions | ERCS  |     ERC20 info    | Complex code |      Features      |
+----------------------+-------------+-------+-------------------+-------------+--------------------+
|        IERC20        |      6      | ERC20 |     No Minting    |      No     |                    |
|                      |             |       | Approve Race Cond. |             |                    |
|                      |             |       |                   |             |                    |
|       Address        |      11     |       |                   |      No     |      Send ETH      |
|                      |             |       |                   |             |    Delegatecall    |
|                      |             |       |                   |             |      Assembly      |
|      SafeERC20       |      6      |       |                   |      No     |      Send ETH      |
|                      |             |       |                   |             | Tokens interaction |
| StakingPlatformTester |     32      |       |                   |      No     | Tokens interaction |
+----------------------+-------------+-------+-------------------+-------------+--------------------+
INFO:Slither:testerStaking.sol analyzed (8 contracts)
```

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Unit Test

All unit tests provided by the team are passing without issues.

1. Staking Platform for DEEP POOL

```
StakingPlatform - Deep Pool
  ✓ Should deploy the new Token (437ms)
  ✓ should distribute tokens among users (91ms)
  ✓ Should deploy the new staking platform
  ✓ Should send tokens to staking platform
  ✓ Should deposit to staking platform (140ms)
  ✓ Should return the amount staked
  ✓ Should start Staking and ending period should last 1 year
  ✓ Should fail if trying to start Staking twice
  ✓ Should return the amount staked
  ✓ Should return and claim rewards staked after 1 day (93ms)
  ✓ Should return the amount staked after 1 day (85ms)
  ✓ Should return the amount of rewards for a specific user after 1 day
  ✓ Should revert if exceed the max staking amount
  ✓ Should deposit 100 000 tokens
  ✓ Should deposit 900 000 tokens
  ✓ Should fail deposit tokens
  ✓ Should fail withdraw residual before ending period
  ✓ Should fail withdraw tokens before ending period
  ✓ Should fail claiming tokens
  ✓ Should not withdraw tokens before lockup period
  ✓ Should not withdraw tokens after 200days lockup still active
  ✓ Should fail claiming tokens
  ✓ Should withdraw tokens after ending period
  ✓ Should return the amount staked after 1000 days (94ms)
  ✓ Should withdraw initial deposit (85ms)
  ✓ Should withdraw residual balances
  ✓ Should fail withdraw residual if nothing to withdraw after increasing 1000days
  ✓ Should return the amount staked once staking finished and withdrew
  ✓ Should fail deposit after staking ended
  ✓ Should return the amount staked


30 passing (1s)
```

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

2. Staking Platform for MID POOL

```
StakingPlatform - Mid Pool
  ✓ Should deploy the new Token (491ms)
  ✓ should distribute tokens among users
  ✓ Should deploy the new staking platform
  ✓ Shoud increase precision
  ✓ Should send tokens to staking platform
  ✓ Should deposit to staking platform (50ms)
  ✓ Should return the amount staked
  ✓ Should start Staking and ending period should last 1 year
  ✓ Should fail if trying to start Staking twice
  ✓ Should return the amount staked
  ✓ Should revert if exceed the max staking amount
  ✓ Should claim rewards and stake for 183 days (2463ms)
  ✓ Should claim rewards and stake for 182 (total 1year) days (2221ms)
  ✓ Should not withdraw residual balances before endingperiod + 1 year
  ✓ Should withdraw residual balances
  ✓ Should fail withdraw initial deposit after withdrawResidualBalance
  ✓ Should withdraw initial deposit
  ✓ Should withdraw residual after tokens sent to contract
  ✓ Should fail withdraw residual if no residual balance


19 passing (5s)
```

3. Staking Platform for QUICK POOL

```
StakingPlatform - Quick Pool
  ✓ Should deploy the new Token (379ms)
  ✓ should distribute tokens among users (113ms)
  ✓ Should deploy the new staking platform
  ✓ Shoud increase precision
  ✓ Should send tokens to staking platform
  ✓ Should deposit to staking platform (141ms)
  ✓ Should return the amount staked
  ✓ Should start Staking and ending period should last 1 year
  ✓ Should fail if trying to start Staking twice
  ✓ Should return the amount staked
  ✓ Should return and claim rewards staked after 1 day (89ms)
  ✓ Should revert if exceed the max staking amount
  ✓ Should deposit 100 000 tokens
  ✓ Should deposit 900 000 tokens
  ✓ Should fail deposit tokens
  ✓ Should fail withdraw tokens before ending period
  ✓ Should withdraw tokens after lockup period
  ✓ Should withdraw tokens after lockup period
  ✓ Should return the amount staked after 185 days (total 366 passed days) (92ms)
  ✓ Should withdraw initial deposit (86ms)
  ✓ Should return the amount staked once staking finished and withdrew
  ✓ Should not withdraw residual balances before endingperiod + 1 year
  ✓ Should withdraw residual balances
  ✓ Should fail withdraw residual if nothing to withdraw
  ✓ Should fail deposit after staking ended
  ✓ Should return the amount staked


26 passing (1s)
```

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

4. Pool Test

```
StakingPlatform - Pool
  ✓ Should deploy the new Token (369ms)
  ✓ should distribute tokens among users (66ms)
  ✓ Should deploy the new staking platform
  ✓ Should set precision to 2
  ✓ Should send tokens to staking platform
  ✓ Should deposit to staking platform for user1 and user2 (51ms)
  ✓ Should return the amount staked
  ✓ Should withdraw tokens before staking starts
  ✓ Should re-deposit to staking platform for user1 and user2
  ✓ Should start Staking and ending period should last 1 year
  ✓ Should fail if trying to start Staking twice
  ✓ Should return the amount staked
  ✓ Should return and claim rewards staked after 1 day (40ms)
  ✓ Should return the amount staked after 1 day (2days passed)
  ✓ Should return the amount staked after 10 days (12days passed) (65ms)
  ✓ Should deposit balance for user1 & user2
  ✓ Should return the amount staked after increasing staked for 10days (22days passed) (80ms)
  ✓ Should deposit balance for user1 & user2
  ✓ Should deposit balance for user3 & user4 (122days passed)
  ✓ Should return the amount staked after 10 days (132days passed) (57ms)
  ✓ Should deposit balance for user1 & user2
  ✓ Should return the amount staked after 10 days (142 days passed) (55ms)
  ✓ Should return the amount of rewards for a specific user after 1 day (143days passed)
  ✓ Should revert if exceed the max staking amount
  ✓ Should deposit 100 000 tokens
  ✓ Should deposit 900 000 tokens
  ✓ Should fail deposit tokens
  ✓ Should fail withdraw residual before ending period
  ✓ Should fail withdraw tokens before ending period
  ✓ Should fail claiming tokens (144 days passed)
  ✓ Should not withdraw tokens after 200days lockup still active (344 days passed)
  ✓ Should deposit to staking platform for user3 and user4
  ✓ Should return the total amount staked
  ✓ Should return and claim rewards staked after 1 day (345 days passed) (44ms)
  ✓ Should withdraw tokens after ending period (845days passed)
  ✓ Should withdraw tokens after ending period
  ✓ Should return the amount staked after 1000 days
  ✓ Should withdraw initial deposit (70ms)
```

5. Withdraw Tests

```
StakingPlatform - Withdraw Amount
  ✓ Should return rewards at start
  ✓ Should return rewards after one day
  ✓ Should return rewards after 50 days
  ✓ Should return rewards after 100 days
  ✓ Should withdraw after 50 days
  ✓ Should withdraw 90% after 50 days and returns rewards after endPeriod (66ms)


6 passing (1s)
```

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

6. Pool Restake Tests

```
StakingPlatform - Restake
  ✓ Should deploy the new Token (369ms)
  ✓ should distribute tokens among users (61ms)
  ✓ Should deploy the new staking platform
  ✓ Should send tokens to staking platform
  ✓ Should deposit to staking platform for user1 and user2 (51ms)
  ✓ Should return the amount staked
  ✓ Should start Staking and ending period should last 1 year
  ✓ Should fail if trying to start Staking twice
  ✓ Should return the amount staked
  ✓ Should return and claim rewards staked after 1 day (41ms)
  ✓ Should return the amount staked after 1 day (2days passed)
  ✓ Should return the amount staked after 10 days (12days passed) (69ms)
  ✓ Should deposit balance for user1 & user2
  ✓ Should return the amount staked after increasing staked for 10days (22days passed) (68ms)
  ✓ Should return the amount staked after increasing staked for 10days (32days passed) (59ms)
  ✓ Should deposit a second time balance for user1 & user2
  ✓ Should withdraw after 100days for user1 and user2
  ✓ Should have 0 rewards for user1 & user2 after 10days
  ✓ Should deposit balance for user3 & user4 (122days passed) (50ms)
  ✓ Should return the amount staked after 10 days (132days passed) (101ms)
  ✓ Should deposit balance for user3 & user4 (47ms)
  ✓ Should return the amount staked after 30 days (213ms)
  ✓ Should re-deposit balance for user3 & user4
  ✓ Should return the amount staked after 30 more days (199ms)
  ✓ Should return the amount of rewards for a specific user after 1 day
  ✓ Should revert if exceed the max staking amount
  ✓ Should deposit 1 000 000 tokens with user5
  ✓ Should deposit 100 000 tokens
  ✓ Should deposit 900 000 tokens
  ✓ Should fail deposit tokens
  ✓ Should fail withdraw residual before ending period
  ✓ Should return 0 rewards after 0 days
  ✓ Should revert if nothing to claim after 0 days
  ✓ Should loop and try to withdraw / deposit (469ms)
  ✓ Should return 0 rewards after 0 days
  ✓ Should return rewards after 1 day
  ✓ Should deposit 1 000 000 tokens with user6
  ✓ Should deposit 1 000 000 tokens with user7
  ✓ Should return rewards after 100 day
  ✓ Should return rewards for user7 (119ms)
```

7. Initial Tests

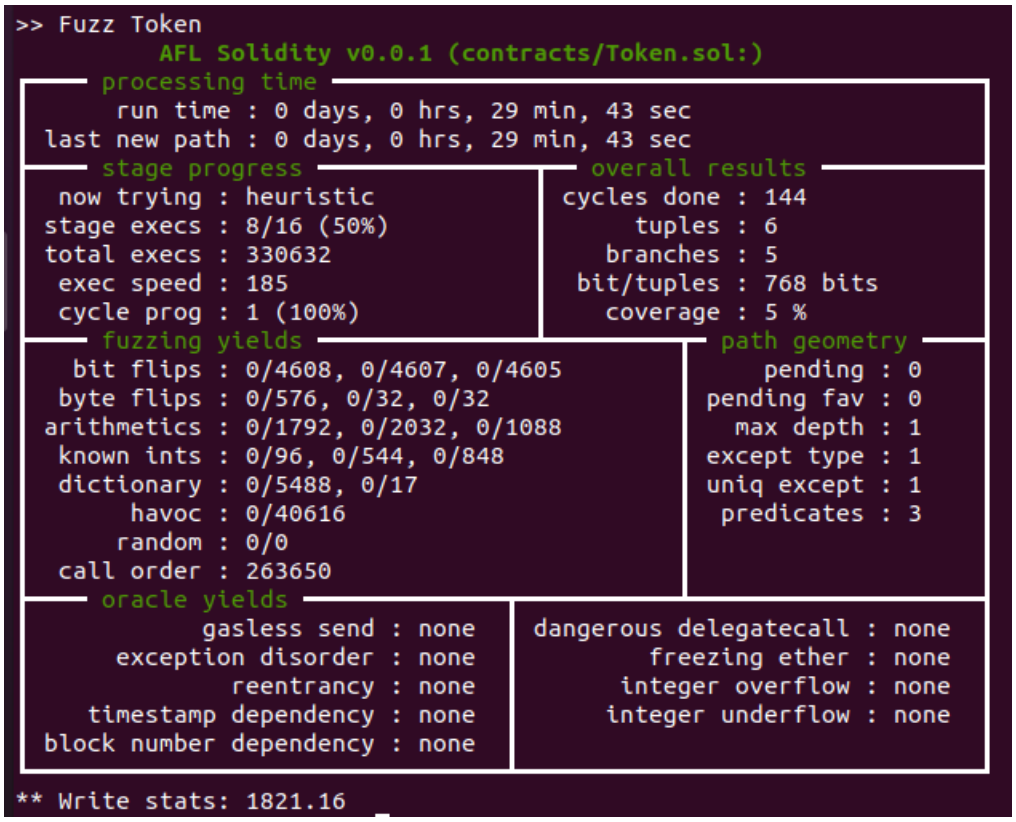
```
StakingPlatform - PoolTests
  ✓ Should return rewards at start
  ✓ Should return rewards after one day
  ✓ Should return rewards after 50 days
  ✓ Should return rewards after 100 days
  ✓ Should return rewards after 200 days (endPeriod + 100days)
  ✓ Should deposit after 50 days and farm until endPeriod (51ms)
  ✓ Should deposit after 50 days and farm until endPeriod: precision(20) (59ms)
  ✓ Should withdraw and deposit & farm all together at the same time until endPeriod (82ms)
  ✓ Should withdraw after 99 days and re-deposit and farm until endPeriod, withdraw scenario (116ms)
  ✓ Should withdraw after 99 days and re-deposit and farm until endPeriod, claimRewards scenario (135ms)
  ✓ Should withdraw after 99 and farm until endPeriod (1day) (73ms)
  ✓ Should withdraw after 99 and farm half a day and then withdraw before ending (102ms)
  ✓ Should test with 0 (70ms)
  ✓ Should test with low value (52ms)
  ✓ Should fail with very low value
  ✓ Should withdraw residual if nobody claimedRewards
  ✓ Should withdraw residual if rewards claimed
  ✓ Should withdraw residual if rewards claimed (with low values) (65ms)
  ✓ Should withdraw residual if nobody claimedRewards (with low values)


19 passing (3s)
```

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Fuzz Testing

1. Token.sol: -

    a. Terminal Output
    [With the use of: " -g -r 0 -d 1800 "]



- Excel Sheet of States for the Output of Fuzz Testing

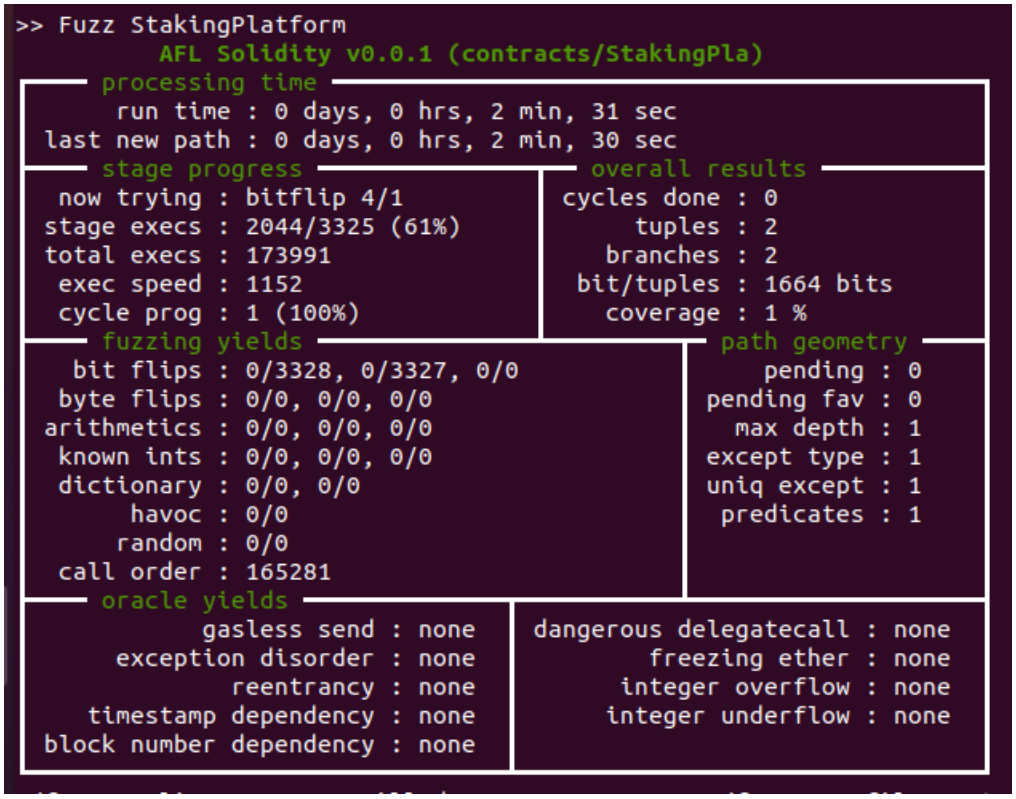
  [With the use of: " -g -r 1 -d 1800 "]

  https://drive.google.com/file/d/1YuRtUaj7ul8yrAAqLsNmiwYC8UBzfcOo/view?usp=sharing

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

2. StakingPlatform.sol: -

    a. Terminal Output
    [With the use of: " -g -r 0 -d 1800 "]



- Excel Sheet of States for the Output of Fuzz Testing

  [With the use of: " -g -r 1 -d 1800 "]

  https://drive.google.com/file/d/1YuRtUaj7uI8yrAAqLsNmiwYC8UBzfcOo/view?usp=sharing

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Vulnerability Checks

TYPES	ORACLES	WHEN A VULNERABILITY IS DETECTED	WHY IT IS VULNERABLE	Results
Error	Gasless Send	Function sends or transfer is called and receiver has a costly fallback function	RunOufOfGasexception	PASSED
Error	Exception Disorder	There is an exception in the call chain but the. These functions hide exceptions	Root of the call chain does not throw exception	PASSED
Error	Timestamp Dependency	The test case evaluates a condition based on timestamp and then sends ether	Miners control the values of timestamp	PASSED
Error	Block Number Dependency	The test case evaluates a condition based on block number and then sends ether	Miners control the values of block number.	PASSED
Error	Danger Delegate Call	delegatecall is executed via msg.data.	The attacker can call any function.	PASSED
Error	Reentrancy	A contract function is called via fallback function from another contract and sends ether.	Refer to the DAO vulnerability	PASSED
Error	Integer Overflow/Underflow	If b >0 and a + b < a or b > 0 and a − b > b or ⋯	Arithmetic error	PASSED
Error	Integer Overflow/Underflow	If b >0 and a + b < a or b > 0 and a − b > b or ⋯	Arithmetic error	PASSED
Warning	Freezing Ether	After all test case, nosend()or transfer() function is executed	The contract is a blackhole for ether	PASSED

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Concluding Remarks

While conducting the audits of the Retreeb smart contracts, it was observed that the contracts contained High, Medium and Low severity issues.

Our auditors suggest that High, Medium and Low severity issues should be resolved by the developers. The recommendations given will improve the operations of the smart contract.

Disclaimer

ImmuneBytes's audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Our team does not endorse the Retreeb platform or its product nor this audit is investment advice.
Notes:

- Please make sure contracts deployed on the mainnet are the ones audited.
- Check for the code refactor by the team on critical issues.

ImmuneBytes