Rocket Capital Investment

Smart Contract Audit Report

September 12, 2021

IMMUNE BYTES

Introduction

1. About Rocket Capital Investment

The Rocket Capital Investment Competition is a decentralized platform to source and incentivize the best in machine-learning applications for finance, ultimately leading to a decentralized autonomous fund management ecosystem driven by the community.

In this first implementation, participants stake tokens, submit their predictions and earn rewards based on their stake and on the performance of their submissions.

Visit these links to learn more about:
Company Website: https://www.rocketcapital.ai
Whitepaper: https://rocket-capital-investment.gitbook.io/rci-competition/white-paper
Smart Contract Documentation:
https://rocket-capital-investment.gitbook.io/competition-dapp/overviews/competition-overview
Web Application: https://competition.rocketcapital.ai/

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 75+ 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 RCI team has provided the following doc for the purpose of audit:

1. https://rocket-capital-investment.gitbook.io/rci-competition/

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: Rocket Capital Investment
- Contracts Name: Child.sol, Token.sol, Multisig.sol, Competition.sol, CompetitionStorage.sol, Registry.sol
- Languages: Solidity(Smart contract)
- Github commit for initial audit: 26ea641b959a17b0a987f429a50d7f0fecde37ad
- Platforms and Tools: Remix IDE, Truffle, Truffle Team, Ganache, Solhint, VScode, Contract Library, Slither, SmartCheck, SFuzz

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:

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	7
Closed	-	-	-

Contract: Competition.sol, CompetitionStorage.sol

High Severity Issues

No issues were found.

Medium severity issues

1. **Function visibility issue found in updateChallengeAndTournamentScores() function**
   Line no -387-389, 394-396

   Explanation:
   The **Competition** contract includes a function called **updateChallengeAndTournamentScores** to store the challenge and tournament scores of participants on-chain.

   As per the current architecture of the contract, most of the **onlyAdmin** functions are usually divided into two parts where the first one is marked external which allows the admin to access the function. While the 2nd part with the actual function logic is made private, thus only accessible by its respective external function.

   However, the same pattern wasn't found with the **updateChallengeAndTournamentScores** function as the function with the logic, in this case, is assigned **Public visibility** (Line 394 to 396), thus making both functions with similar names accessible from outside the contract.

   Moreover, while the function with external visibility reads the **_challengeCounter** directly from the contract, the **public function** demands it to be passed by the admin which might not be a very effective mechanism.

```
386
387     function updateChallengeAndTournamentScores(address[] calldata participants, uint256[] calldata challengeScores, uint256
388     external override
389     returns (bool success)
390     {
391         success = updateChallengeAndTournamentScores(_challengeCounter, participants, challengeScores, tournamentScores);
392     }
393
394     function updateChallengeAndTournamentScores(uint32 challengeNumber, address[] calldata participants, uint256[] calldata
395     public override onlyAdmin
 96     returns (bool success)
 97     {
```

   Recommendation:
   If the above-mentioned scenario is not intended, the function visibility of the function should be updated accordingly.

2. **Violation of Check_Effects_Interaction Pattern in the Withdraw function**
   Line no - 292-305

   Explanation:
   The **Competition** contract includes function, **sponsor()**, that update some of the very imperative state variables of the contract after the external calls are being made.

   An external call within a function technically shifts the control flow of the contract to another contract for a particular period of time. Therefore, as per the Solidity Guidelines, any modification of the state variables in the base contract must be performed before executing the external call.
   Although in this case, the call is being made to the native token contract itself, it's imperative to not violate the best security practices.

   The following function in the contract update the state variables after making an external call at the lines mentioned below:

   - **sponsor** at Line **696**

```
689     function sponsor(uint256 amountToken)
690     external override
691     returns (bool success)
692     {
693         require(_challenges[_challengeCounter].phase == 4, "Competition -
694         _token.transferFrom(msg.sender, address(this), amountToken);
695         uint256 currentCompPoolAmt = _competitionPool;
696         _competitionPool = currentCompPoolAmt + amountToken;
697         success = true;
```

   Recommendation:
   Check Effects Interaction Pattern must be followed while implementing external calls in a function.

Low Severity Issues

1. **Adequate use of Return Value of an External Call was not found**
   Line no - 80, 694

   Explanation:
   The external calls made in the above-mentioned lines do return a boolean value that indicates whether or not the external call made was successful.

These boolean return values can be used in the function as a check to ensure that the further execution of the function is only allowed if the external is successfully made.
However, the **Competition** contract never uses these return values throughout the contract.

Recommendation:
Effective use of all the return values from external calls must be ensured within the contract.

2. **External Visibility should be preferred**

Explanation:
Functions that are never called throughout the contract should be marked as **external** visibility instead of **public** visibility.
This will effectively result in Gas Optimization as well.

During the automated testing of the **Competition** contract, it was found that the following functions could be marked as **external** within the contract:

- **updateSubmission()**
- **updateResults()**

Recommendation:
If the **PUBLIC** visibility of the above-mentioned functions is not intended, then the **EXTERNAL** Visibility keyword should be preferred.

Recommendations

--

Contract: MultiSig.sol

High Severity Issues

No issues were found.

Medium severity issues

No issues were found.

Low Severity Issues

1. **Redundant Require Statement found in removeOwner function**
   Line - 134

   Explanation:
   As per the current architecture of the removeOwner function, it was found that it contains a **notNull()** modifier which ensures that the address of the owner is not a **zero address**.

   However, this validation has already been performed while adding a particular owner, in the **addOwner function** at Line 120.

   This makes the **notNull modifier** in the **removeOwner function** redundant and badly affects the gas optimization of the function.

```
128     /// @dev Allows to remove an owner. Transaction has
129     /// @param owner Address of owner.
130     function removeOwner(address owner)
131         public
132         onlyWallet
133         ownerExists(owner)
134         notNull(owner)
135         validRequirement(owners.length - 1, required)
136     {
```

   Recommendation:
   Redundant require statements and validations should be avoided.

2. **Absence of Error messages in Require Statements**
   Line no - 49-95, 106

   Explanation:
   The **Multisig** contract includes a few functions(at the above-mentioned lines) that don't contain any error message in the **require** statement.

   While this makes it troublesome to detect the reason behind a particular function revert, it also reduces the readability of the code.

   Recommendation:
   Error Messages must be included in every require statement in the contract.

3. **External Visibility should be preferred**

   Explanation:
   Functions that are never called throughout the contract should be marked as **external** visibility instead of **public** visibility.
   This will effectively result in Gas Optimization as well.

   During the automated testing of the **MultiSig** contract, it was found that the following functions could be marked as **external** within the contract:

   - **addOwner**
   - **removeOwner**
   - **replaceOwner**
   - **submitTransaction**
   - **revokeConfirmation**
   - **getConfirmationCount**
   - **getTransactionCount**
   - **getOwners**
   - **getConfirmations**
   - **getTransactionIds**

   Recommendation:
   If the **PUBLIC** visibility of the above-mentioned functions is not intended, then the **EXTERNAL** Visibility keyword should be preferred.

Recommendations

1. **Internal function isContract() is never used within the contract**
   Line no 427-439

   Explanation:
   The Multisig Contract includes an internal function called **isContract()** at the above-mentioned line. However, the function is never used as the contract uses the Address library imported in the contract.

   Recommendation:
   Unnecessary state variables and functions must be removed.

2. **Commented codes must be wiped out before deployment**

   Explanation:
   The Multisig contract includes quite a few commented codes. This affects the readability of the code.

   Recommendation:
   If these instances of code are not required in the current version of the contract, then the commented codes must be removed before deployment.

Contract: Token.sol

High Severity Issues

No issues were found.

Medium severity issues

No issues were found.

Low Severity Issues

1. **Absence of Zero Address Validation**
   Line no- 78, 85

   Explanation:
   The **Token** Contract includes quite a function called **authorizeCompetition**, which updates an imperative mapping, i.e., **_authorizedCompetitions**, in the contract.

```
78      function authorizeCompetition(address competitionAddress)
79      external
80      onlyAdmin
81      {
82          _authorizedCompetitions[competitionAddress] = true;
83
84          emit CompetitionAuthorized(competitionAddress);
85      }
```

   However, during the automated testing of the contract, it was found that no Zero Address validation is implemented before updating the address of the mapping.

   Although the function has already been assigned an **onlyOwner** modifier, keeping in mind the immutable nature of the smart contract, its imperative to implement input validations in function.

   Recommendation:
   A require statement should be included in such functions to ensure no invalid address is passed in the arguments.

Recommendations

--

Contract: Registry.sol

High Severity Issues

No issues were found.

Medium severity issues

No issues were found.

Low Severity Issues

1. **Absence of Input Validation found in few functions**
   Line no - 33-41, 71-79

   Explanation:
   The registry contract includes functions like **registerNewCompetition** and **registerNewExtension** that doesn't involve any input validation for the following arguments:
   A. **competitionAddress**
   B. **rulesLocation**
   C. **extensionAddress**
   D. **informationLocation**

   It's imperative to implement adequate input validation to avoid unwanted behavior during contract execution.

   Recommendation:
   Effective input validations should be included.

Recommendations

--

Contract: ChildToken.sol

High Severity Issues

No issues were found.

Medium severity issues

No issues were found.

Low Severity Issues

No issues were found.

Recommendations

--

Automated Test Results

1. Competition.sol

```
Compiled with solc
Number of lines: 2101 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 13 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 5
Number of informational issues: 48
Number of low issues: 4
Number of medium issues: 2
Number of high issues: 2
ERCs: ERC165

+--------------+------------+--------+------------+--------------+--------------------+
|     Name     | # functions | ERCS  | ERC20 info | Complex code |      Features      |
+--------------+------------+--------+------------+--------------+--------------------+
|    IToken    |     14     |        |            |      No      |                    |
| EnumerableSet |     20     |        |            |      No      |                    |
|    Address   |     11     |        |            |      No      |    Send ETH        |
|              |            |        |            |              |   Delegatecall     |
|              |            |        |            |              |     Assembly       |
|  Competition |    141     | ERC165 |            |      No      | Tokens interaction |
|              |            |        |            |              |    Upgradeable     |
+--------------+------------+--------+------------+--------------+--------------------+
INFO:Slither:myFlats/ComptetitionFlat.sol analyzed (13 contracts)
```

2. CompetitionStorage.sol

```
Compiled with solc
Number of lines: 381 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 3 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 11
Number of informational issues: 36
Number of low issues: 0
Number of medium issues: 0
Number of high issues: 0


+---------------------+-------------+------+------------+--------------+----------+
|        Name         | # functions | ERCS | ERC20 info | Complex code | Features |
+---------------------+-------------+------+------------+--------------+----------+
|       IToken        |      14     |      |            |      No      |          |
|   EnumerableSet     |      20     |      |            |      No      |          |
| CompetitionStorage  |      0      |      |            |      No      |          |
+---------------------+-------------+------+------------+--------------+----------+
INFO:Slither:myFlats/ComptetitionStorageFlat.sol analyzed (3 contracts)
```

3. MultiSig.sol

```
Compiled with solc
Number of lines: 629 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 2 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 10
Number of informational issues: 23
Number of low issues: 1
Number of medium issues: 0
Number of high issues: 0


+----------+-------------+------+------------+--------------+--------------+
|   Name   | # functions | ERCS | ERC20 info | Complex code |   Features   |
+----------+-------------+------+------------+--------------+--------------+
| Address  |      11     |      |            |      No      |   Send ETH   |
|          |             |      |            |              | Delegatecall |
|          |             |      |            |              |   Assembly   |
| MultiSig |      19     |      |            |     Yes      |   Send ETH   |
|          |             |      |            |              |   Assembly   |
+----------+-------------+------+------------+--------------+--------------+
INFO:Slither:myFlats/MultiSigFlat.sol analyzed (2 contracts)
```

4. Token.sol

```
Compiled with solc
Number of lines: 1305 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 12 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 14
Number of informational issues: 5
Number of low issues: 3
Number of medium issues: 3
Number of high issues: 0

ERCs: ERC20, ERC165

+-------------+-------------+-------------+--------------------+--------------+----------+
|     Name    | # functions |     ERCS    |     ERC20 info     | Complex code | Features |
+-------------+-------------+-------------+--------------------+--------------+----------+
| ICompetition |      55     |             |                    |      No      |          |
|    Token    |      53     | ERC20,ERC165 |     No Minting     |      No      |          |
|             |             |             | Approve Race Cond. |              |          |
|             |             |             |                    |              |          |
+-------------+-------------+-------------+--------------------+--------------+----------+
INFO:Slither:myFlats/TokenFlat.sol analyzed (12 contracts)
```

5. Registry.sol

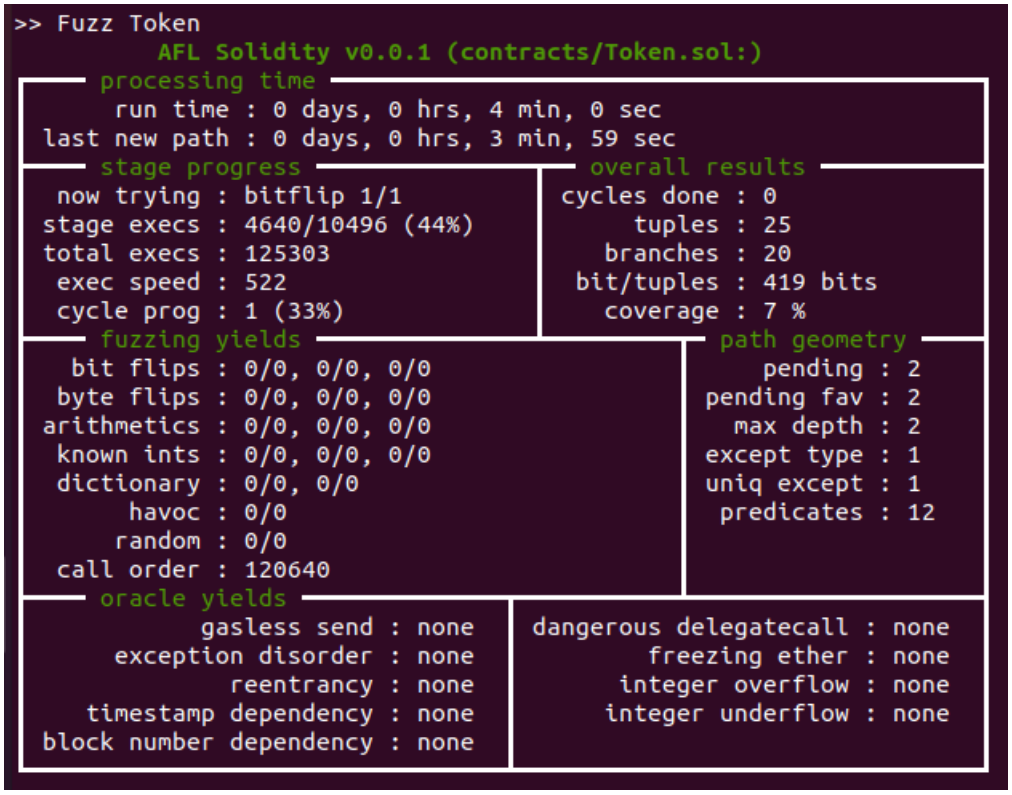
```
Compiled with solc
Number of lines: 530 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 8 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 3
Number of informational issues: 5
Number of low issues: 1
Number of medium issues: 0
Number of high issues: 0

ERCs: ERC165

+----------+-------------+--------+-----------+--------------+----------+
|   Name   | # functions | ERCS   | ERC20 info | Complex code | Features |
+----------+-------------+--------+-----------+--------------+----------+
| Registry |      55     | ERC165 |           |      No      |          |
+----------+-------------+--------+-----------+--------------+----------+
INFO:Slither:myFlats/RegistryFlat.sol analyzed (8 contracts)
```

6. ChildToken.sol

```
Compiled with solc
Number of lines: 1356 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 13 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 14
Number of informational issues: 5
Number of low issues: 6
Number of medium issues: 3
Number of high issues: 0

ERCs: ERC20, ERC165

+-------------+------------+-------------+--------------------+--------------+----------+
|     Name    | # functions |     ERCS    |     ERC20 info     | Complex code | Features |
+-------------+------------+-------------+--------------------+--------------+----------+
| ICompetition |      55    |             |                    |      No      |          |
|  ChildToken  |      58    | ERC20,ERC165 |     No Minting     |      No      |          |
|              |            |             | Approve Race Cond. |              |          |
|              |            |             |                    |              |          |
+-------------+------------+-------------+--------------------+--------------+----------+
INFO:Slither:myFlats/ChildFlat.sol analyzed (13 contracts)
```

Fuzz Testing

1. **Token.sol**: -
    a. Terminal Output
    [With use of : " -g -r 0 -d 240 "]

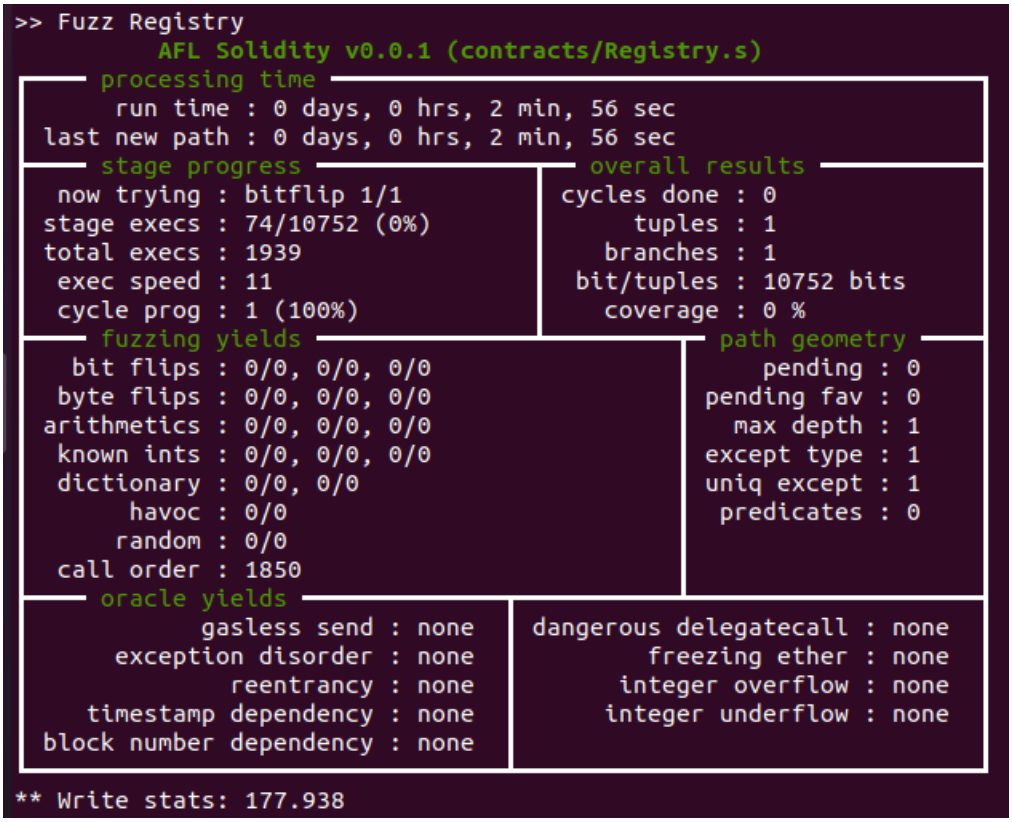


- Excel Sheet of States for the Output of Fuzz Testing

  [With use of : " -g -r 1 -d 240 "]

  https://drive.google.com/file/d/1bTGUrxeSAyqB1yLJzvzrUdETQuDrVXIF/view?usp=sharing

2. **Registry.sol**: -
    a. Terminal Output
    [With use of : " -g -r 0 -d 180 "]

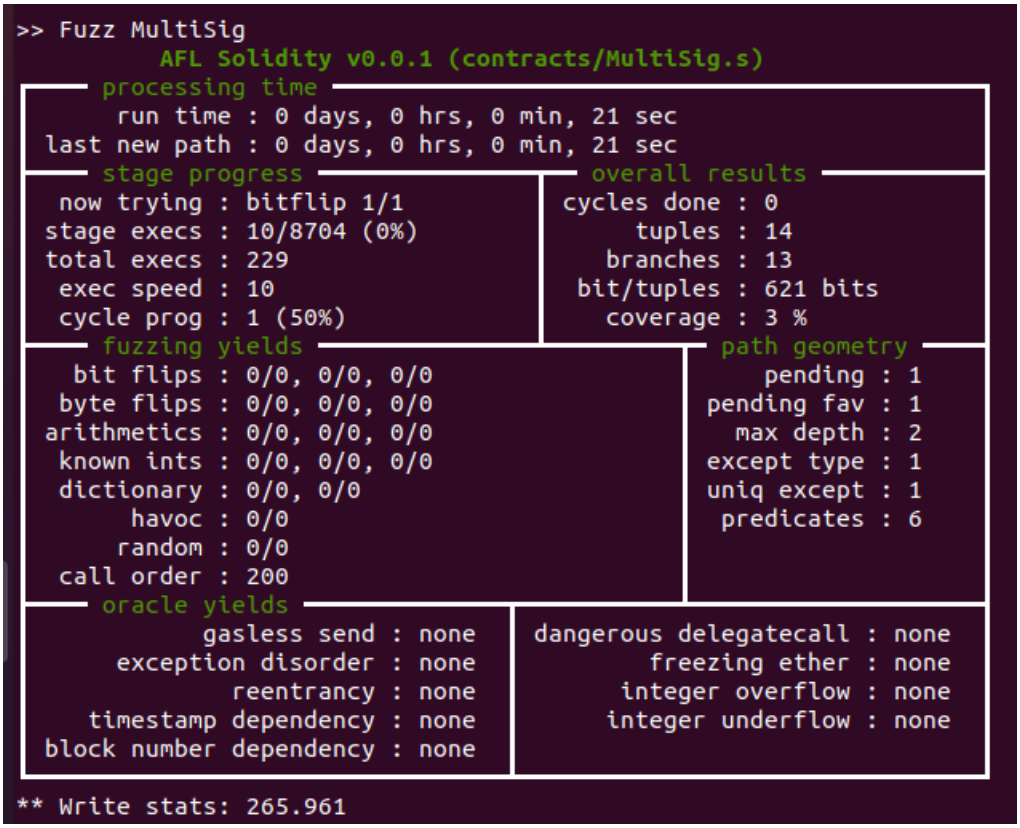


- Excel Sheet of States for the Output of Fuzz Testing

  [With use of : " -g -r 1 -d 180 "]

  https://drive.google.com/file/d/1WBPagUHcW75IZVL_9JwpRrCTVD--JMVJ/view?usp=sharing

3. **MultiSig.sol**: -
    a. Terminal Output
    [With use of : " -g -r 0 -d 260 "]

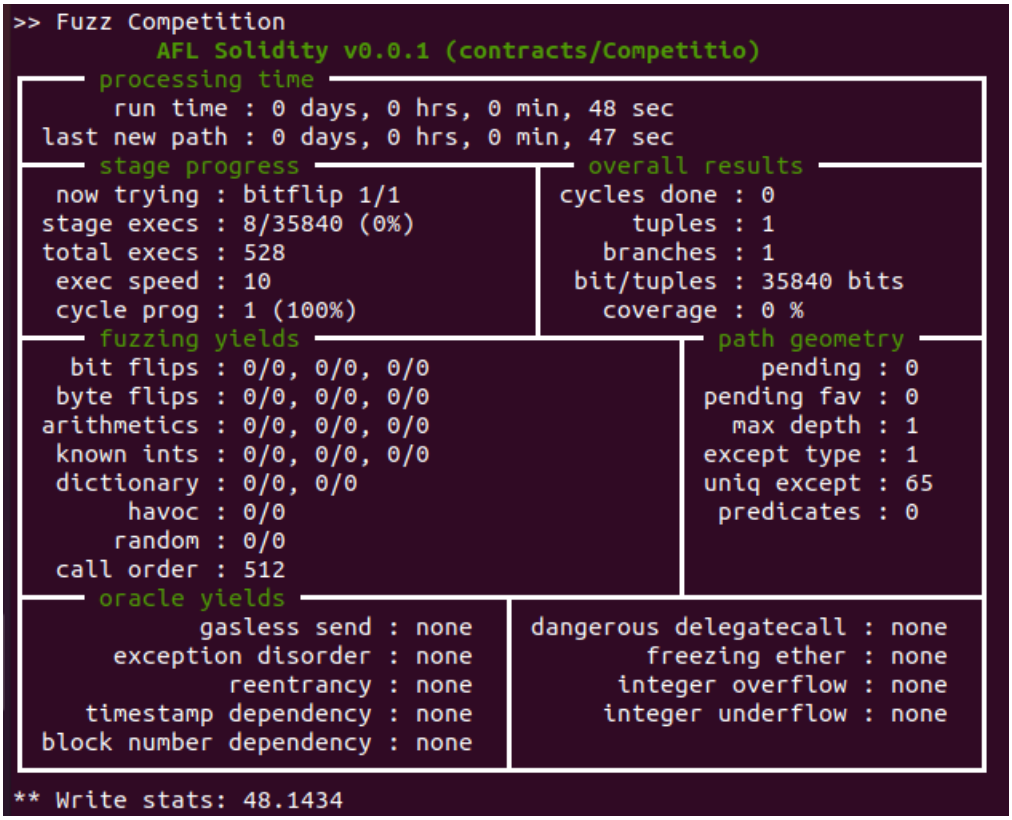


- Excel Sheet of States for the Output of Fuzz Testing

  [With use of : " -g -r 1 -d 240 "]

  https://drive.google.com/file/d/1EYhpZA2Nfpj6onuje1ziNcnOT520_w41/view?usp=sharing

4. **Competition.sol**: -
    a. Terminal Output
    [With use of : " -g -r 0 -d 120 "]



- Excel Sheet of States for the Output of Fuzz Testing

  [With use of : " -g -r 1 -d 120 "]

  https://drive.google.com/file/d/1w_2g-bRxu0su7rTobSo5c5qNw9bBQoxK/view?usp=sharing

Concluding Remarks

While conducting the audits of the Rocket Capital Investment smart contracts, it was observed that the contracts contained Medium and Low severity issues with a few areas of recommendations. No High severity is found.

Our auditors suggest that Medium and Low severity issues and recommendations 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 Rocket Capital Investment 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