CERTIK

Security Assessment

rocket-science

Apr 28th, 2022

Table of Contents

Summary

This report has been prepared for rocket-science to discover issues and vulnerabilities in the source code of the rocket-science project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Static Analysis and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

Overview

Project Summary

Project Name	rocket-science
Platform	BSC
Language	Solidity
Codebase	https://github.com/JaJayLee/rocket-science
Commit	504dc5354ab0a349f2f8678466b65bc26d966787

Audit Summary

Delivery Date	Apr 28, 2022 UTC
Audit Methodology	Static Analysis, Manual Review

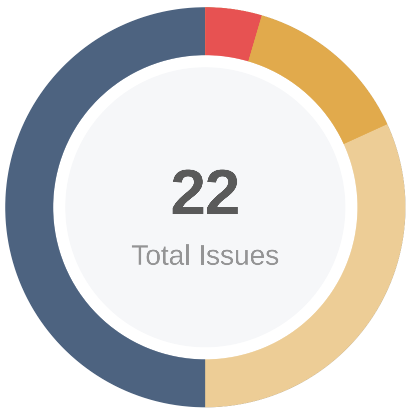
Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Mitigated	Partially Resolved	Resolved
● Critical	1	0	0	0	0	0	1
● Major	0	0	0	0	0	0	0
● Medium	3	0	0	0	0	0	3
● Minor	7	0	0	3	0	1	3
● Informational	11	0	0	2	0	1	8
● Discussion	0	0	0	0	0	0	0

Audit Scope

ID	File	SHA256 Checksum
RSJ	contracts/RocketScience.sol	9772fcdd645dc90d4bc00a31b34b2563ee065abe014cdee83474c8d8190a37cc
CJJ	contracts/Context.sol	95cec669ef4ebf990dc641ed2880b675f03e5ef9017e8c12d08f5fe6704c1f5b
OJJ	contracts/Ownable.sol	f37e7e1f1ff52659fe00db9a92074934d45011658dbf4f1f96e83d7aa7b8f8bb
OJL	contracts/Ownable.sol	f37e7e1f1ff52659fe00db9a92074934d45011658dbf4f1f96e83d7aa7b8f8bb
RSL	contracts/RocketScience.sol	26c61ff9b32e48296a34caa7b860cbdbcdbd7942baf1b4b687572e4cee5ddfe2
CJL	contracts/Context.sol	95cec669ef4ebf990dc641ed2880b675f03e5ef9017e8c12d08f5fe6704c1f5b

Findings



🟥	**Critical**	**1** (4.55%)
🟧	**Major**	**0** (0.00%)
🟧	**Medium**	**3** (13.64%)
🟨	**Minor**	**7** (31.82%)
🟦	**Informational**	**11** (50.00%)
🟩	**Discussion**	**0** (0.00%)

ID	Title	Category	Severity	Status
RSJ-01	All Funds Can Be Stolen Via The `takeInvestment()` Function	Logical Issue	● Critical	✓ Resolved
RSJ-02	The Referral Reward May Be Written To A Null Address Via The `investByRef()` Function	Logical Issue	● Medium	✓ Resolved
RSJ-03	Incorrect ROI Value	Logical Issue	● Medium	✓ Resolved
RSJ-04	`getActivePackets()` And `getCompletedPackets()` Functions May Return Incorrect Values	Logical Issue	● Medium	✓ Resolved
RSJ-05	Usage Of `transfer()` For Sending BNB	Volatile Code	● Minor	✓ Resolved
RSJ-06	User-Defined Getters	Gas Optimization	● Minor	Partially Resolved
RSJ-07	Unnecessary Fallback Function	Language Specific	● Minor	ⓘ Acknowledged
RSJ-08	Usage Of Magic Number	Coding Style	● Minor	✓ Resolved
RSJ-09	Code Repetition	Coding Style	● Minor	✓ Resolved
RSJ-10	Unnecessary `transfer()` Function	Language Specific	● Minor	ⓘ Acknowledged

ID	Title	Category	Severity	Status
RSJ-11	Events Should Be Used	Language Specific	● Minor	ⓘ Acknowledged
RSJ-12	Redundant SafeMath Usage	Language Specific	● Informational	✓ Resolved
RSJ-13	Unlocked Compiler Version	Language Specific	● Informational	ⓘ Acknowledged
RSJ-14	Function Should Be Declared External	Gas Optimization	● Informational	Partially Resolved
RSJ-15	Too Many Digits	Coding Style	● Informational	✓ Resolved
RSJ-16	Variable Names Too Similar	Coding Style	● Informational	ⓘ Acknowledged
RSJ-17	`uint256 t` Should Be Replaced With Enum Values	Coding Style	● Informational	✓ Resolved
RSJ-18	Bad Constant Name `REF`	Coding Style	● Informational	✓ Resolved
RSJ-19	Commented Out Code	Coding Style	● Informational	✓ Resolved
RSJ-20	Usage Of Magic Number To Declare The Percentage Of The Owner's Reward	Coding Style	● Informational	✓ Resolved
RSJ-21	Typo In The Error Message	Coding Style	● Informational	✓ Resolved
RSJ-22	`_isNew` Can Be Avoided	Gas Optimization	● Informational	✓ Resolved

RSJ-01 | All Funds Can Be Stolen Via The `takeInvestment()` Function

Category	Severity	Location	Status
Logical Issue	● Critical	contracts/RocketScience.sol (base): 176~217	✓ Resolved

Description

Calling of `takeInvestment()` at exactly `block.timestamp == userPackets[msg.sender][_packetId].finishTime` allows to extract any amount of money by a subsequent call.

The scenario:

1. The attacker invests funds in `_packetId`.
2. The attacker calls `takeInvestment()` with the corresponding `_packetId` exactly at the moment of packet expiration. Since in BSC blocks are produced every 3 seconds, it requires about 3 attempts to get lucky.
3. The transaction is included into block with `block.timestamp == userPackets[msg.sender][_packetId].finishTime`
4. The attacker gets the reward for 15 days. `lastUpdate[msg.sender][_packetId]` is assigned with `finishTime`.
5. The attacker waits for some time and calls `takeInvestment()` again with the same `_packetId`.
6. Since `lastUpdate[msg.sender][_packetId] == userPackets[msg.sender][_packetId].finishTime` the attacker gets the reward for elapsed time.

Recommendation

We recommend rewriting the code this way:

```
        uint256 _end = min(block.timestamp, userPackets[msg.sender][_packetId].finishTime);
        uint256 _elapsed = 0;
        if (_end > lastUpdate[msg.sender][_packetId])
            _elapsed = _end - lastUpdate[msg.sender][_packetId];

        uint256 _earned = userPackets[msg.sender][_packetId].invested * DAILY_REWARD * _elapsed / DAY / 100;

        lastUpdate[msg.sender][_packetId] = block.timestamp;
        investors[msg.sender].earned += _earned;

        userPackets[msg.sender][_packetId].paid += _earned;
```

RSJ-02 | The Referral Reward May Be Written To A Null Address Via The `investByRef()` Function

Category	Severity	Location	Status
Logical Issue	● Medium	contracts/RocketScience.sol (base): 143~150	✓ Resolved

Description

`investByRef()` can be called with `_referrer = 0` so the next time a call from the same address will be treated as from a new investor and referral rewards will be credited to a null address. `totalInvestors` will also be incremented.

Recommendation

We recommend validating the address passed as `_referrer`.

RSJ-03 | Incorrect ROI Value

Category	Severity	Location	Status
Logical Issue	● Medium	contracts/RocketScience.sol (base): 298, 307	✓ Resolved

Description

224% ROI is supposed to be 225%.

Recommendation

We recommend calculating of magic values directly using the declared constants.

RSJ-04 | `getActivePackets()` And `getCompletedPackets()` Functions May Return Incorrect Values

Category	Severity	Location	Status
Logical Issue	● Medium	contracts/RocketScience.sol (base): 289~313	✓ Resolved

Description

In case `allPackets[i].finishTime` == `block.timestamp` or `_allPackets[i].paid` == `_allPackets[i].invested.mul(224).div(100)` packets are not returned by both `getActivePackets()` and `getCompletedPackets()`.

Recommendation

We recommend treating of just finished and fully paid packets as completed and return them in `getCompletedPackets()`.

RSJ-05 | Usage Of `transfer()` For Sending BNB

Category	Severity	Location	Status
Volatile Code	● Minor	contracts/RocketScience.sol (base): 62, 109, 173, 208, 258, 270	✓ Resolved

Description

After EIP-1884 was included in the Istanbul hard fork, it is not recommended to use `.transfer()` or `.send()` for transferring native tokens as these functions have a hard-coded value for gas costs making them obsolete as they are forwarding a fixed amount of gas, specifically `2300`. This can cause issues in case the linked statements are meant to be able to transfer funds to other contracts instead of EOAs.

Recommendation

We advise that the linked `.transfer()` and `.send()` calls are substituted with the utilization of the `sendValue()` function from the `Address.sol` implementation of OpenZeppelin either by directly importing the library or copying the linked code.

RSJ-06 | User-Defined Getters

Category	Severity	Location	Status
Gas Optimization	● Minor	contracts/RocketScience.sol (base): 273~279, 341~343, 345~351, 353~359	Partially Resolved

Description

The linked functions are equivalent to the compiler-generated getter functions for the respective variables.

Recommendation

We recommend declaring of mentioned state variables as `public`.

Alleviation

`[CertiK]`: Fixes missed in functions `getWithdrawals()` and `getReferralsRewards()`

RSJ-07 | Unnecessary Fallback Function

Category	Severity	Location	Status
Language Specific	● Minor	contracts/RocketScience.sol (base): 61~63	ⓘ Acknowledged

Description

The `fallback()` function in the `RocketScience` contract is unnecessary as it allows nonstandard function selectors and does not contain any actual logic.

Recommendation

Consider removing the `fallback()` function in the `RocketScience` contract.

Alleviation

`[CertiK]`: The client acknowledged the Issue but decided to do not address it in current version.

RSJ-08 | Usage Of Magic Number

Category	Severity	Location	Status
Coding Style	● Minor	contracts/RocketScience.sol (base): 191	✓ Resolved

Description

The `takeInvestment()` function uses the magic number `225` as the return on investment after the full investment cycle based on a payout of 15% per day. The value will not be updated if constants are modified.

Recommendation

We recommend replacing the magic number `225` with `PACKET_LIFETIME / DAY * DAILY_REWARD` or introducing of a new constant.

RSJ-09 | Code Repetition

Category	Severity	Location	Status
Coding Style	● Minor	contracts/RocketScience.sol (base): 219~251	✓ Resolved

Description

Functions `totalClaimable()` and `takeInvestment()` share the same code.

Recommendation

We recommend calling `totalClaimable()` by `takeInvestment()` to improve code maintainability.

RSJ-10 | Unnecessary `transfer()` Function

Category	Severity	Location	Status
Language Specific	● Minor	contracts/RocketScience.sol (base): 269~271	ⓘ Acknowledged

Description

The `transfer()` function in the `RocketScience` contract is unnecessary and does not contain any actual logic.

Recommendation

Consider removing the `transfer()` function in the `RocketScience` contract.

Alleviation

`[CertiK]`: The client acknowledged the Issue but decided to do not address it in current version.

RSJ-11 | Events Should Be Used

Category	Severity	Location	Status
Language Specific	● Minor	contracts/RocketScience.sol (base): 53	ⓘ Acknowledged

Description

Events should be used instead of `referralsRewards`.

Recommendation

We recommend using of events instead of `referralsRewards` by calling emit in functions `invest()` and `investByRef()` to record referral reward values.

Alleviation

`[CertiK]`: The client acknowledged the Issue but decided to do not address it in current version.

RSJ-12 | Redundant SafeMath Usage

Category	Severity	Location	Status
Language Specific	● Informational	contracts/RocketScience.sol (base): 5	✓ Resolved

Description

Solidity version >=0.8.0 includes checked arithmetic operations and underflow/overflow by default, making SafeMath redundant.

Recommendation

We recommend removing the SafeMath library and use standard arithmetic operators to reduce code complexity.

RSJ-13 | Unlocked Compiler Version

Category	Severity	Location	Status
Language Specific	● Informational	contracts/RocketScience.sol (base): 2	ⓘ Acknowledged

Description

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.8.6` the contract should contain the following line:

```
pragma solidity 0.8.6;
```

Alleviation

`[CertiK]`: The client acknowledged the Issue but decided to do not address it in current version.

RSJ-14 | Function Should Be Declared External

Category	Severity	Location	Status
Gas Optimization	● Informational	contracts/RocketScience.sol (base): 73, 176, 219, 253, 273, 289, 315, 341, 345, 353	Partially Resolved

Description

The functions which are never called internally within the contract should have `external` visibility for gas optimization.

Recommendation

We advise to change the visibility of the aforementioned functions to `external`.

Alleviation

`[CertiK]`: Fixes missed in functions `getWithdrawals()` and `getReferralsRewards()`

RSJ-15 | Too Many Digits

Category	Severity	Location	Status
Coding Style	● Informational	contracts/RocketScience.sol (base): 74~78, 137~141	✓ Resolved

Description

Literals with many digits are difficult to read and review.

```
        require(
            msg.value >= 100000000000000000 &&
                msg.value <= 1000000000000000000000,
            "Wrong amount"
        );
```

Recommendation

We recommended to use the scientific notation or denomination suffix to improve readability.

RSJ-16 | Variable Names Too Similar

Category	Severity	Location	Status
Coding Style	● Informational	contracts/RocketScience.sol (base): 52, 210~214, 260~264	ⓘ Acknowledged

Description

Similar names make the code more difficult to review.

`_withdrawal` is too similar to `withdrawals`.

Recommendation

We recommend renaming variables so their names are not too similar.

Alleviation

`[CertiK]`: The client acknowledged the Issue but decided to do not address it in current version.

RSJ-17 | `uint256 t` Should Be Replaced With Enum Values

Category	Severity	Location	Status
Coding Style	● Informational	contracts/RocketScience.sol (base): 26, 213, 263	✓ Resolved

Description

Magic numbers 1, 2 are indicating type of withdrawals (earned, referrals).

Recommendation

We recommend declaring

`enum withdrawalsType { earned, referrals }`

RSJ-18 | Bad Constant Name `REF`

Category	Severity	Location	Status
Coding Style	● Informational	contracts/RocketScience.sol (base): 40	✓ Resolved

Description

`REF` is actually used as a percentage of the investment when calculating referrals rewards, which does not correspond to its name.

Recommendation

We recommend renaming `REF` to make its meaning clear to users.

RSJ-19 | Commented Out Code

Category	Severity	Location	Status
Coding Style	● Informational	contracts/RocketScience.sol (base): 60, 281~287	✓ Resolved

Description

Commented out code is redundant.

Recommendation

We recommend removing the commented out code.

RSJ-20 | Usage Of Magic Number To Declare The Percentage Of The Owner's Reward

Category	Severity	Location	Status
Coding Style	● Informational	contracts/RocketScience.sol (base): 109, 173	✓ Resolved

Description

The `invest()` function uses the magic number `10` as the owner reward.

Recommendation

We recommend introducing the `PERCENTAGE_OF_OWNER_REWARD` constant.

RSJ-21 | Typo In The Error Message

Category	Severity	Location	Status
Coding Style	● Informational	contracts/RocketScience.sol (base): 136	✓ Resolved

Description

The linked error message string contains a typo.

Recommendation

We advise to update the linked message string. You probably meant "You can't invest to yourself".

RSJ-22 | `_isNew` Can Be Avoided

Category	Severity	Location	Status
Gas Optimization	● Informational	contracts/RocketScience.sol (base): 143~150	✓ Resolved

Description

`_isNew` can be replaced by checking `investors[msg.sender].referrer == address(0)` by combining two logical blocks into one.

Recommendation

We recommend rewriting the code this way:

```
if (investors[msg.sender].referrer == address(0)) {
            investors[msg.sender].referrer = _referrer;
            totalInvestors++;
        }
        else _referrer = investors[msg.sender].referrer;
```

Appendix

Finding Categories

Gas Optimization

Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Logical Issue

Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of private or delete.

Coding Style

Coding Style findings usually do not affect the generated byte-code but rather comment on how to make the codebase more legible and, as a result, easily maintainable.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

Disclaimer

“AS AVAILABLE” AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER’S OR ANY OTHER PERSON’S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER’S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK’S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER’S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED “AS IS” AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK’S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

About

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.