HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Rebased
Date: August 19, 2020
Platform: Ethereum
Language: Solidity

This document may contain confidential information about IT systems and the intellectual property of the customer as well as information about potential vulnerabilities and methods of their exploitation.

The report containing confidential information can be used internally by the customer or it can be disclosed publicly after all vulnerabilities fixed - upon a decision of the customer.

Document

Name	Smart Contract Code Review and Security Analysis Report for Rebased
Platform	Ethereum / Solidity
Initial Audit	
Repository	https://github.com/UFrags/rebased
Commit	DB08C81C12468A8E36EEBF1597AEA65249FE39E4
Date	19 AUG 2020

Table of contents

Document....2
Table of contents....3
Introduction....4
Scope....4
Executive Summary....5
Severity Definitions....5
AS-IS overview....6
Audit overview....10
Conclusion....11
Disclaimers....12

Introduction

Hacken OÜ (Consultant) was contracted by Rebased (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of Customer`s smart contract and its code review conducted between August 17, 2020 - August 19, 2020.

Scope

The scope of the project is smart contracts in the repository:

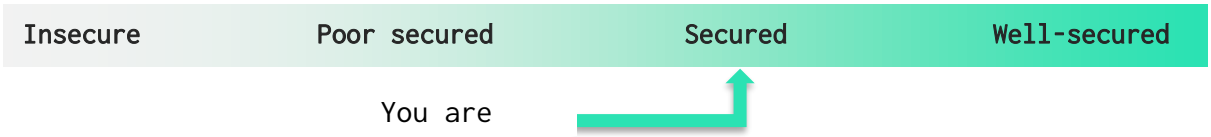
```
Audit Repository - https://github.com/UFrags/rebased
Commit - DB08C81C12468A8E36EEBF1597AEA65249FE39E4
```

We have scanned this smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that are considered (the full list includes them but is not limited to them):

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

Executive Summary

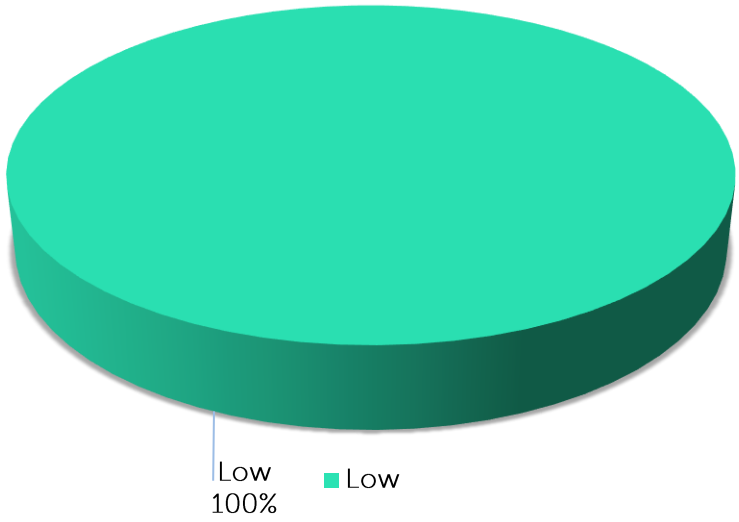
According to the assessment, Customer`s smart is well secured and can be deployed to the mainnet.



Our team performed an analysis of code functionality, manual audit, and automated checks with Mythril and Slither. All issues found during automated analysis were manually reviewed and important vulnerabilities are presented in the Audit overview section. A general overview is presented in AS-IS section and all found issues can be found in the Audit overview section.

We found 3 low severity and a bunch of code style issues in smart contract code during the audit.

Graph 1. The distribution of vulnerabilities.



Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to tokens lose etc.
High	High-level vulnerabilities are difficult to exploit; however, they also have significant impact on smart contract execution, e.g. public access to crucial functions
Medium	Medium-level vulnerabilities are important to fix; however, they can't lead to tokens lose

Low	Low-level vulnerabilities are mostly related to outdated, unused etc. code snippets, that can't have significant impact on execution
Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations and info statements can't affect smart contract execution and can be ignored.

AS-IS overview

Rebased.sol

Rebased.sol contains multiple contracts:

- SafeMath
- IERC20
- ERC20Detailed
- SafeMathInt
- Rebased

SafeMath is a copy of *SafeMath* library from *OpenZeppelin*. The difference is that the version in *OpenZeppelin* contains messages when validations are failed and Rebased version does not have those messages.

IERC20 is an *ERC20* specification. The exact copy of OpenZeppelin version.

ERC20Detailed is an abstract contract that implements *IERC20* contract and add constructor and functions with token information: name, symbol, decimals.

SafeMathInt is a library that provides safe operations with *int256* data type.

Rebased is *ERC20Detailed*. It's using *SafeMath* and *SafeMathInt*

Contract Rebased has 10 fields and constants:

- *address public monetaryPolicy* - store address of the MonetaryPolicy contract. Initialized in the constructor. Could not be changed after initialization.
- *uint256 private constant DECIMALS = 9* - provide token decimals.
- *uint256 private constant MAX_UINT256 = ~uint256(0)* - used to determine max value of uint256 data type.
- *uint256 private constant INITIAL_FRAGMENTS_SUPPLY = 25 * 10**5 * 10**DECIMALS* - an initial supply of tokens.

- *uint256 private constant TOTAL_GONS = MAX_UINT256 - (MAX_UINT256 % INITIAL_FRAGMENTS_SUPPLY)* - total “gons”. Used in calculations of *_gonsPerFragment* during contract construction and rebase.
- *uint256 private constant MAX_SUPPLY = ~uint128(0)* - max tokens supply.
- *uint256 private _totalSupply* - actual total tokens supply.
- *uint256 private _gonsPerFragment* - actual value of “gons” per fragment. Used to calculate balance of an address.
- *mapping(address => uint256) private _gonBalances* - actual “gons” balance of an address.
- *mapping (address => mapping (address => uint256))* private _allowedFragments - stores allowance for tokens transfer.

Contract Rebased defines 1 event:

- LogRebase - emitted rebase function call. Contains time and total tokens supply.

Contract Rebased has 2 modifiers:

- onlyMonetaryPolicy - check if the function caller is *MonetaryPolicy* contract.
- validRecipient - check if an address of tokens receiver is valid.

Contract Rebased has 10 functions:

- **constructor** - sets *_monetaryPolicy* address and provides contract name, symbol and decimals.
- rebase - an external function that initiates new rebase cycle. Can be called only from the *MonetaryPolicy* contract address. Changes *_gonsPerFragment* value according to the provided *supplyDelta* parameter and total tokens supply.
- totalSupply - an external function that returns total tokens supply. Part of ERC20 specification.
- balanceOf - an external function that calculates balance of a provided address depending on *_gonsPerFragment* value. Part of ERC20 specification.
- transfer - an external function used to transfer tokens. to parameter should pass *validRecipient* validation. Part of ERC20 specification.

- allowance - an external function to check the quantity of tokens that an owner has allowed to a spender. Part of ERC20 specification.
- transferFrom - an external function to transfer tokens from one address to another. to parameter should pass *validRecipient* validation. Part of *ERC20* specification.
- approve - an external function to approve the passed address to spend the specified quantity of tokens on behalf of msg.sender. Part of *ERC20* specification.
- increaseAllowance - an external function to increase the quantity of tokens that an owner has allowed to a spender. Part of *ERC20* specification.
- decreaseAllowance - an external function to decrease the quantity of tokens that an owner has allowed to a spender. Part of *ERC20* specification.

MonetaryPolicy

MonetaryPolicy.sol contains multiple contracts:

- SafeMath
- SafeMathInt
- UInt256Lib
- IERC20
- IRebased
- IOracle
- Ownable
- MonetaryPolicy

SafeMath, SafeMathInt, IERC20 are the same as in **Rebased.sol**.

UInt256Lib is a library with one function:

- toInt256Safe - safely converts a uint256 to an int256

IRebased is a part of specification of the Rebased contract.

IOracle is a specification for a market data provider.

Ownable is a contract responsible for access control.

MonetaryPolicy is *Ownable.*

MonetaryPolicy has 12 fields and constants:

- *IRebased public rebased* - Rebased contract address.
- *IOracle public cpiOracle* - CPI oracle address. Provides the current CPI.
- *IOracle public marketOracle* - Market oracle address.
- *uint256 private baseCpi* - CPI value at the time of launch.
- *uint256 public deviationThreshold* - Deviation threshold. If the current exchange rate is within this fractional distance from the target, no supply update is performed. Fixed point number - same format as the rate.
- *uint256 public rebaseLag* - the rebase lag parameter, used to dampen the applied supply adjustment by 1 / rebaseLag. The value set 20 to in the constructor. Could not be changed.
- *uint256 public minRebaseTimeIntervalSec* - minimum time should pass before each rebase. The value is set to 12 hours in the constructor. Could not be changed.
- *uint256 public lastRebaseTimestampSec* - Block timestamp of last rebase operation.
- *uint256 public epoch* - the number of rebase cycles since inception.
- *uint256 private constant DECIMALS* = 18 - decimals of the token and of a data received from data oracles.
- *uint256 private constant MAX_RATE* = 10**6 * 10**DECIMALS - max token exchange rate.
- *uint256 private constant MAX_SUPPLY* = ~(uint256(1) << 255) / MAX_RATE - max tokens supply

MonetaryPolicy has 9 functions:

- constructor - initializes contract default values.
- canRebase - public view function. Returns true if at least minRebaseTimeIntervalSec seconds have passed since last rebase.
- rebase - external function. initiates a new rebase operation, provided the minimum time period has elapsed.
- getRebaseValues - public view function. Calculates the *supplyDelta* and returns the current set of values for the rebase.
- computeSupplyDelta - internal view function. Computes the total supply adjustment in response to the exchange rate

- withinDeviationThreshold - internal view function. Checks if the rate is within the deviation threshold from the target rate.
- setRebased - external function with *onlyOwner* modifier. Sets the reference to the Rebased token governed.
- setCpiOracle - external function with *onlyOwner* modifier. Sets the reference to the CPI oracle.
- setMarketOracle - external function with *onlyOwner* modifier. Sets the reference to the market oracle.

Audit overview

Critical

No critical issues were found.

High

No high severity issues were found.

Medium

No medium severity issues were found.

Low

1. Files *SimpleOracle.sol* and *MonetaryPolicy.sol* have equal content.
2. *MonetaryPolicy* relies on data providers (oracles) that are not provided in the Audit scope. The Customer should ensure that a data fetched from them is strictly in line with *MonetaryPolicy* logic.
3. *MAX_SUPPLY* constants in *MonetaryPolicy* and *Rebased* are different. *MonetaryPolicy* values is 5.7896 × 10^52 and *Rebased* value is 3.4028 × 10^38. Though, it cannot lead to any losses or errors because of validations in *rebase* function of the *Rebase* contract.

Lowest / Code style / Best Practice

1. It's recommended to add error messages to validation in SafeMathInt and SafeMath.
2. Instead of using constants *MIN_INT256* and *MAX_INT256* in *SafeMathInt* and *MAX_UINT256* in *Rebased* it's possible to use

type(int256).min, *type(int256).max*, *type(uint256).max* that has been introduced in 0.6.8 version of the compiler.

3. It's recommended to move common contracts and interfaces to separate files to reduce code duplication.
4. Function *canRebase* of the *MonetaryPolicy* can be modifier.

Conclusion

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

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

Reviewed smart contracts are of the good quality. Security engineers found 3 low severity issues during audit. Also, it's recommended to follow best practices of solidity programming: make truffle project with tests, move common components to separate files, provide messages during assertation errors, etc.

The Customer should pay attention to the correctness of data providers (oracles) that are used in MonetaryPolicy contract. Their behavior can affect the correctness of the rebase flow.

Disclaimers

Hacken Disclaimer

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

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

Technical Disclaimer

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