Crowdfund Smart Contract Audit

Etherparty™, 06 April 2018

Table of Contents

Disclaimer

This is a limited report on our findings based on our analysis, in accordance with good industry practice as at the date of this report, in relation to: (i) cybersecurity vulnerabilities and issues in the smart contract source code analysed, the details of which are set out in this report, (Source Code); and (ii) the Source Code compiling, deploying and performing the intended functions. In order to get a full view of our findings and the scope of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the below disclaimer below – please make sure to read it in full.

DISCLAIMER: By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any and all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis, and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and Zenoic Proprietary Limited trading as "Iosiro" and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers and other representatives) (Iosiro) owe no duty of care towards you or any other person, nor does Iosiro make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties or other terms of any kind except as set out in this disclaimer, and Iosiro hereby excludes all representations, warranties, conditions and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, Iosiro hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against Iosiro, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of use of this report, and any reliance on this report.

1. Introduction

iosiro was commissioned by Etherparty™ to conduct an audit on their crowdfund smart contracts. The audit was performed between 28 March 2018 and 04 April 2018. On 06 April 2018, a verification was performed to confirm that issues identified in this report were sufficiently addressed.

This report is organized into the following sections.

- **Section 2 - Executive Summary:** A high-level description of the findings of the audit.
- **Section 3 - Audit Details:** A description of the scope and methodology of the audit.
- **Section 4 - Smart Contract Descriptions:** Descriptions of the intended functionality of the smart contracts.
- **Section 5 - Detailed Findings:** Detailed descriptions of the findings of the audit.

The information in this report should be used to understand the risk exposure of the smart contracts, and as a guide to improve the security posture of the smart contracts by remediating the issues that were identified. The results of this audit are only a reflection of the source code reviewed at the time of the audit and of the source code that was determined to be in-scope.

2. Executive Summary

This report presents the findings of an audit performed by iosiro on the Etherparty™ crowdfund smart contracts. The purpose of the audit was to achieve the following.

- Ensure that the smart contracts functioned as intended.
- Identify potential security flaws.

Due to the unregulated nature and ease of transfer of cryptocurrencies, operations that store or interact with these assets are considered very high risk with regards to cyber attacks. As such, the highest level of security should be observed when interacting with these assets. This requires a forward-thinking approach, which takes into account the new and experimental nature of blockchain technologies. There are a number of techniques that can help to achieve this, some of which are described below.

- Security should be integrated into the development lifecycle.
- Defensive programming should be employed to account for unforeseen circumstances.
- Current best practices should be followed wherever possible.

No major security issues were identified during the audit. Additionally, the audit found the smart contracts to be of a high standard. The code was generally well designed and clearly written. It separated token and crowdfund logic and made use of commonly used libraries where possible.

Several issues were identified during the audit, however, their severities were limited to low and informational risk levels, and were related to minor discrepancies between the code and the specification and best practice issues that did not expose the contracts to significant security risk. At the conclusion of the audit, all of the identified issues had been addressed sufficiently.

The risk posed by the smart contracts can be further mitigated by using the following controls prior to releasing the contracts to a production environment.

- Extend the coverage of the test suite.
- Use a public bug bounty program to identify security vulnerabilities.
- Perform additional audits using different teams.

3. Audit Details

3.1 Scope

The source code considered in-scope for the assessment is described below. Code from any other files are considered to be out-of-scope.

3.1.1 Etherparty

Project Name: eth-smart-contracts
Commit: 206c794 (link)
Files: `Crowdfund.sol, Migrations.sol, Token.sol, library/BasicToken.sol, library/CanReclaimToken.sol, library/ERC20.sol, library/ERC20Basic.sol, library/NonZero.sol, library/Ownable.sol, library/SafeERC20.sol, library/SafeMath.sol, library/StandardToken.sol`

3.2 Methodology

A variety of techniques were used to perform the audit, these are outlined below.

3.2.1 Dynamic Analysis

The contracts were compiled, deployed, and tested using both Truffle tests and manually on a local test network. A number of pre-existing tests were included in the project. The results of the tests and the coverage can be found in Appendix II.

3.2.2 Automated Analysis

Tools were used to automatically detect the presence of potential vulnerabilities, such as reentrancy, timestamp dependency bugs, transaction-ordering dependency bugs, and so on. Static analysis was conducted using Mythril and Oyente. Additional tools, such as the Remix IDE, compilation output and linters were used to identify potential security flaws.

3.2.3 Code Review

Source code was manually reviewed to identify potential security flaws. This type of analysis is useful for detecting business logic flaws and edge-cases that may not be detected through dynamic or static analysis.

3.3 Risk Ratings

Each Issue identified during the audit is assigned a risk rating. The rating is dependent on the criteria outlined below..

- **High Risk -** The issue could result in a loss of funds for the contract owner or users.
- **Medium Risk -** The issue results in the code specification operating incorrectly.
- **Low Risk -** A best practice or design issue that could affect the security standard of the contract.
- **Informational -** The issue addresses a lapse in best practice or a suboptimal design pattern that has a minimal risk of affecting the security of the contract.
- **Closed -** The issue was previously identified and addressed to a satisfactory level to remove the potential risk that it posed.

4. Smart Contract Descriptions

The following section outlines the intended specification for the smart contracts.

4.1 Crowdfund.sol

The crowdfund logic was implemented using a combination of custom code and the popular OpenZeppelin[1] library. Notable design decisions are outlined below.

Token Allocations
Tokens could be allocated to different addresses.

Dynamic Vesting Period
Each token allocation could be locked for a different amount of time.

Dynamic Token Allocation
Each token allocation could receive a different amount of tokens.

Dynamic Pricing Rounds
Different exchange rates could be set for the crowdfund depending on the current time.

Burn or Move unsold tokens
It was possible to set a forwarding address, where tokens would be sent to after the crowdfund. In the event of setting the forwarding address to 0x0, the tokens would effectively be burned.

Schedule and Reschedule
It was possible to schedule and reschedule the crowdfund before the sale had started.

Presale
It was possible for the crowdfund owner to freely send presale tokens before the crowdfund had started from the initial crowdfund allocation.

4.2 Token.sol

The token logic was implemented using a combination of custom code and the popular OpenZeppelin library. Notable design decisions are outlined below.

[1] https://github.com/OpenZeppelin/zeppelin-solidity/

ERC20 Token
The token implemented the ERC20 specification[2].

Whitelist
It was possible, but not required, to use a whitelist that would require participants to be whitelisted in order to purchase tokens.

Fixed Total Supply
A fixed supply of tokens could be set.

[2] https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md

5. Detailed Findings

The following section includes in-depth descriptions of the findings of the audit.

5.1 High Risk

No high risk issues were present at the conclusion of the audit.

5.2 Medium Risk

No medium risk issues were present at the conclusion of the audit.

5.3 Low Risk

No low risk issues were present at the conclusion of the audit.

5.4 Informational

No informational risk issues were present at the conclusion of the audit.

5.5 Closed

5.5.1 Excessive Token Allocation Rounds Permitted (Low)

Token.sol: line 84

Description

While the specification described that “less than 10” dynamic token allocations could take place, it was found that up to a total of ten allocations were permitted. This was in contrast to the dynamic pricing round, which was also described in the specification as having “less than 10” rounds, which correctly enforced a limit of up to 9 pricing rounds.

Remedial Action

It is recommended that either the specification is amended to specify, “less than or equal to 10” rounds, or the code is adjusted to correctly enforce the limit.

Implemented Action

Fixed in f63b52c.

5.5.2 Unnecessary Usage of `assert()` (Low)

Crowdfund: lines 61, 124

Description

The manual code review revealed that `assert()` was being used unnecessarily. Strictly speaking, `assert()` should only be used to check invariants, while `require()` should be used for validating inputs. Correct usage of these functions can improve the performance of source code analyzers as the analyzer will be able to better infer the expected value. Additionally, unnecessary use of `assert()` can result in wasted funds, as it will consume all of the available gas.

Remedial Action

It is recommended that the following changes are made.

- *Crowdfund: line 61* should use require.
- *Crowdfund: line 124* should use require.

Implemented Action

Fixed in e18c897.

5.5.3 Design Comments (Informational)

General

The following describes possible actions to improve the functionality and readability of the codebase.

Using `enum` for states

The state of the crowdfund was found to be controlled using a combination of modifiers and assertions. While this approach achieved its purpose, it can lead to less readable and less maintainable code. For example, `closeCrowdfund()` had a custom state that existed between the end of the crowdfund and before the crowdfund was closed. A more elegant solution would make use of the `enum` type to determine the current state, which could allow for a modifier `inState(state)`. An example of this pattern can be found in the TokenMarket crowdsale implementation[3]. This would also make it easier to add functionality that could return the current state that the contract is in.

Implemented Action

[3] https://github.com/TokenMarketNet/ico/blob/master/contracts/CrowdsaleBase.sol

No action required.

Additionally, this approach would encourage stronger state checks. Before the crowdfund has been scheduled, `duringCrowdsale` relies only on the `endsAt` variable to default to 0, while a stronger check would include `activated` to ensure that the crowdfund had in fact been activated before allow tokens to be purchased. Having a centralized state management system can help to ensure that states are correctly enforced.

Presale and sale allocation amounts combined

When initializing a crowdfund, the creator must specify the total number of tokens that can be sold in both the presale and sale rounds in a single number. As a result, if tokens are unsold in the presale round, they will become available in the sale round. This is not necessarily a security issue, however, it's simply stated to highlight the possibility of an unintuitive action from happening.

Implemented Action

No action required.

No option to change timelocks

Although it was possible to reschedule the crowdfund, it was not possible to update the timelocks of the allocations. This may become a problem if the crowdfund is postponed for an excessive period of time. For example, if a company announces that tokens will be locked for one year, and then decides to postpone the crowdfund by one month, they would need to deploy a whole new contract to accurately reflect the intended dates. It may be advantageous to allow the owner of the crowdfund to update the timelocks until the crowdfund has begun. However, this does add additional complexity to the contract, which may not be desirable.

Implemented Action

Fixed in 2831156.

No `removeManyFromWhitelist()`

While it was found to be possible to add an array of addresses to the whitelist, addresses had to be individually removed from the whitelist. While this is strictly not required, it would allow an owner to remove multiple addresses from the whitelist without sending multiple transactions to the contract.

Implemented Action

Fixed in a509897.

Usage of `var` deprecated

Lines 211 and 265 were found to use the `var` keyword to infer the return value of a function. This functionality is intended to be deprecated[4], and from a readability perspective, requires the reader to determine the type of value being returned. It is recommended that an explicit type (`uint256`) is used in its place.

Implemented Action

Fixed in 10a359d.

Scope

It was found that an overly permissive scope was used in some instances, this could lead to wasted gas as an external scope can consume less gas than a public scope in some instances. Additionally, from a readability perspective a more explicit scope can be helpful to determine who will be calling the function.

It is recommended that the following changes are considered.

- *Line 161* can be set to external.
- *Line 174* can be set to external.
- *Line 183* can be set to external.
- *Line 191* can be set to external.

Implemented Action

Fixed in 04f61ba.

Inexact solidity compiler version used

The pragma version was not fixed to a specific version, as it specified `^0.4.18`, which would result in using the highest non-breaking version (highest version below `0.5.0`). According to best practice, where possible, all contracts should use the same compiler version, which should be fixed to a specific version. This helps to ensure that contracts do not accidentally get deployed using an alternative compiler, which may pose the risk of unidentified bugs. An explicit version also helps with code reuse, as users would be able to see the author's intended compiler version. It is recommended that the pragma version is changed to a fixed value, for example `0.4.18`.

[4] https://github.com/ethereum/solidity/issues/3301

Implemented Action

Fixed in 42287af.

Timestamp

The `now` keyword (an alias for `block.timestamp`) is used to determine the current time. This value can be marginally affected by miners (by up to 900 seconds), so a common best practice is to rather use `block.number` to determine the time. However, the risk posed in this circumstance is inconsequential, and it is simply listed for completeness. No action is necessary.

Implemented Action

No action required.

5.5.4 Crowdfund Can Exist In Multiple States Simultaneously (Informational)

Crowdfund.sol: lines 61, 67

Description

It was found that the modifiers `duringCrowdfund()` and `onlyAfterCrowdfund()` could both return true at the same time. This was due to the fact that `duringCrowdfund()` checked whether it was less than or equal to the end of the crowdfund while `onlyAfterCrowdfund()` checked whether it was greater than or equal to the end of the crowdfund. No significant risks were found to be associated with this, the worst case being that the crowdfund would be closed before some potentially valid purchases were made.

Remedial Action

It is recommended that one of the two modifiers removes the “or equal to” component, so that only one modifier will return true when the current time is equal to `now`.

Implemented Action

Fixed in 216c098.

Appendix I: Tester Profiles

Kyle Riley

Kyle is a co-founder of iosiro, a blockchain security startup. He has five years of experience as a security consultant and researcher, previously heading up the research team at MWR InfoSecurity, South Africa. He won the Mobile Application category at Mobile Pwn2Own 2014 held using an exploit chain of three 0-day vulnerabilities. He has presented at a number of international conferences, including the invite-only Qualcomm Mobile Security Summit in San Diego, USA. Kyle contributes to several open-source projects, including multiple levels to the OpenZeppelin Ethereum CTF, Ethernaut.

More information can be found here.

Appendix II: Test Coverage

The results of the test coverage generated using solidity-coverage are given below.

Contract: Crowdfund

✓ Init: The contract is initialized with the right variables (1006ms)

✓ Schedule and Reschedule crowdfund: It should schedule the crowdfund and not let me reschedule after the crowdfund is active (1771ms)

✓ Rates per epoch: It should return the right price when buying tokens (2447ms)

✓ Change Forward and Wallet Address: It should let only the owner to change those addresses (610ms)

✓ BuyTokens() and function (): It should let a buyer buy tokens (2015ms)

✓ BuyTokens(): It should not let a buyer buy tokens after there is no more crowdfund allocation (1201ms)

✓ closeCrowdfund(): It should let me close the crowdfund at the appropriate time (1315ms)

✓ closeCrowdfund(): It should let me burn tokens (1320ms)

✓ deliverPresaleTokens(): It should let me deliver the presale tokens at an appropriate time (1342ms)

✓ kill(): It should kill the contract under certain circumstances (1189ms)

✓ WHITELISTED BuyTokens() and function (): It should let a buyer buy tokens only if whitelisted (3013ms)

Contract: Token

✓ Init: The contract is initialized with the right variables (870ms)

✓ Transfer: It tests the transfer function (1403ms)

✓ TransferFrom: It tests the transferFrom function (1492ms)

✓ MoveAllocation: It tests the moveAllocation function (1796ms)

15 passing (23s)

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	95.51	76.92	100	95.83	
Crowdfund.sol	93.65	77.5	100	94.2	200,215,220,256
Token.sol	100	75	100	100	
contracts/library/	64.15	41.67	59.09	63.79	
BasicToken.sol	100	50	100	100	
CanReclaimToken.sol	0	100	0	0	22,23
ERC20.sol	100	100	100	100	
ERC20Basic.sol	100	100	100	100	
NonZero.sol	66.67	50	66.67	66.67	14,15
Ownable.sol	40	50	66.67	50	37,38,39
SafeERC20.sol	0	0	0	0	15,19,23
SafeMath.sol	91.67	62.5	100	91.67	15
StandardToken.sol	52.38	37.5	40	52.38	... 92,94,96,97
All files	83.8	62.5	80.43	83.77	

Appendix III: Static Analysis Results

The following section reproduces the output of different static analysis tools. Issues identified below have been manually investigated and would be included in Section 5 if they presented a security risk.

Mythril

The following output was produced by Mythril.

==== Exception state ====
Type: Informational
Contract: StandardToken
Function name: _function_0xfc0c546a
PC address: 1975
A reachable exception (opcode 0xfe) has been detected. This can be caused by type errors, division by zero, out-of-bounds array access, or assert violations. This is acceptable in most situations. Note however that assert() should only be used to check invariants. Use require() for regular input checking.

In file: contracts/Crowdfund.sol:61

assert(now >= startsAt && now <= endsAt)

==== Exception state ====
Type: Informational
Contract: StandardToken
Function name: scheduleCrowdfund(uint256)
PC address: 3735
A reachable exception (opcode 0xfe) has been detected. This can be caused by type errors, division by zero, out-of-bounds array access, or assert violations. This is acceptable in most situations. Note however that assert() should only be used to check invariants. Use require() for regular input checking.

In file: contracts/Crowdfund.sol:153

assert(startsAt >= now && endsAt > startsAt)

==== Exception state ====
Type: Informational
Contract: StandardToken
Function name: reScheduleCrowdfund(uint256)
PC address: 7189
A reachable exception (opcode 0xfe) has been detected. This can be caused by type errors, division by zero, out-of-bounds array access, or assert violations. This is acceptable in most situations. Note however that assert() should only be used to check invariants. Use require() for regular input checking.

In file: contracts/library/SafeMath.sol:45

assert(c >= a)

==== Exception state ====
Type: Informational
Contract: StandardToken
Function name: _function_0xfc0c546a
PC address: 4521
A reachable exception (opcode 0xfe) has been detected. This can be caused by type errors, division by zero, out-of-bounds array access, or assert violations. This is acceptable in most situations. Note however that assert() should only be used to check invariants. Use require() for regular input checking.

In file: contracts/Crowdfund.sol:237

rates[i]

==== Exception state ====
Type: Informational
Contract: StandardToken
Function name: getRate()
PC address: 7424
A reachable exception (opcode 0xfe) has been detected. This can be caused by type errors, division by zero, out-of-bounds array access, or assert violations. This is acceptable in most

situations. Note however that assert() should only be used to check invariants. Use require() for regular input checking.

In file: contracts/library/SafeMath.sol:36

assert(b <= a)

==== Exception state ====
Type: Informational
Contract: StandardToken
Function name: addManyToWhitelist(address[])
PC address: 5835
A reachable exception (opcode 0xfe) has been detected. This can be caused by type errors, division by zero, out-of-bounds array access, or assert violations. This is acceptable in most situations. Note however that assert() should only be used to check invariants. Use require() for regular input checking.

In file: contracts/Crowdfund.sol:285

_beneficiaries[i]

==== Exception state ====
Type: Informational
Contract: StandardToken
Function name: reScheduleCrowdfund(uint256)
PC address: 6262
A reachable exception (opcode 0xfe) has been detected. This can be caused by type errors, division by zero, out-of-bounds array access, or assert violations. This is acceptable in most situations. Note however that assert() should only be used to check invariants. Use require() for regular input checking.

In file: contracts/Crowdfund.sol:166

assert(startsAt >= now && endsAt > startsAt)

==== Exception state ====
Type: Informational
Contract: StandardToken
Function name: _function_0xdd418ae2
PC address: 6286
A reachable exception (opcode 0xfe) has been detected. This can be caused by type errors, division by zero, out-of-bounds array access, or assert violations. This is acceptable in most situations. Note however that assert() should only be used to check invariants. Use require() for regular input checking.

In file: contracts/Crowdfund.sol:48

Rate[] public rates

==== Message call to external contract ====
Type: Warning
Contract: StandardToken
Function name: _function_0x17ffc320
PC address: 7377
This contract executes a message call to an address provided as a function argument. Generally, it is not recommended to call user-supplied adresses using Solidity's call() construct. Note that attackers might leverage reentrancy attacks to exploit race conditions or manipulate this contract's state.

In file: contracts/library/SafeERC20.sol:15

token.transfer(to, value)

==== Message call to external contract ====
Type: Warning
Contract: StandardToken
Function name: _function_0x17ffc320
PC address: 2936
This contract executes a message call to an address provided as a function argument. Generally, it is not recommended to call user-supplied adresses using Solidity's call()

construct. Note that attackers might leverage reentrancy attacks to exploit race conditions or manipulate this contract's state.

In file: contracts/library/CanReclaimToken.sol:22

token.balanceOf(this)

==== Message call to external contract ====
Type: Informational
Contract: StandardToken
Function name: _function_0x18b7fed8
PC address: 3459
This contract executes a message call to to another contract. Make sure that the called contract is trusted and does not execute user-supplied code.

In file: contracts/Crowdfund.sol:255

token.moveAllocation(_batchOfAddresses[i], _amountOfTokens[i])

==== Message call to external contract ====
Type: Informational
Contract: StandardToken
Function name: kill()
PC address: 4046
This contract executes a message call to to another contract. Make sure that the called contract is trusted and does not execute user-supplied code.

In file: contracts/Crowdfund.sol:265

token.allocations(this)

==== Message call to external contract ====
Type: Informational
Contract: StandardToken

Function name: closeCrowdfund()
PC address: 5258
This contract executes a message call to to another contract. Make sure that the called contract is trusted and does not execute user-supplied code.

In file: contracts/Crowdfund.sol:214

token.moveAllocation(forwardTokensTo, amount)

==== Message call to external contract ====
Type: Informational
Contract: StandardToken
Function name: closeCrowdfund()
PC address: 4974
This contract executes a message call to to another contract. Make sure that the called contract is trusted and does not execute user-supplied code.

In file: contracts/Crowdfund.sol:211

token.allocations(this)

==== Integer Underflow ====
Type: Warning
Contract: StandardToken
Function name: _function_0x17ffc320
PC address: 7376
A possible integer underflow exists in the function _function_0x17ffc320.
The substraction may result in a value < 0.

In file: contracts/library/SafeERC20.sol:15

token.transfer(to, value)

==== Integer Underflow ====

Type: Warning
Contract: StandardToken
Function name: _function_0x17ffc320
PC address: 2935
A possible integer underflow exists in the function _function_0x17ffc320.
The substraction may result in a value < 0.

In file: contracts/library/CanReclaimToken.sol:22

token.balanceOf(this)

==== Integer Underflow ====
Type: Warning
Contract: StandardToken
Function name: _function_0x18b7fed8
PC address: 3458
A possible integer underflow exists in the function _function_0x18b7fed8.
The substraction may result in a value < 0.

In file: contracts/Crowdfund.sol:255

token.moveAllocation(_batchOfAddresses[i], _amountOfTokens[i])

==== Integer Underflow ====
Type: Warning
Contract: StandardToken
Function name: kill()
PC address: 4045
A possible integer underflow exists in the function kill().
The substraction may result in a value < 0.

In file: contracts/Crowdfund.sol:265

token.allocations(this)

==== Integer Underflow ====
Type: Warning
Contract: StandardToken
Function name: closeCrowdfund()
PC address: 5257
A possible integer underflow exists in the function closeCrowdfund().
The substraction may result in a value < 0.

In file: contracts/Crowdfund.sol:214

token.moveAllocation(forwardTokensTo, amount)

==== Integer Underflow ====
Type: Warning
Contract: StandardToken
Function name: closeCrowdfund()
PC address: 4973
A possible integer underflow exists in the function closeCrowdfund().
The substraction may result in a value < 0.

In file: contracts/Crowdfund.sol:211

token.allocations(this)

Oyente

The output of using Oyente on the smart contracts can be found in the following table.

File	Analysis Results
Crowdfund.sol	EVM Code Coverage: 90.9% Integer Underflow: False Integer Overflow: True Parity Multisig Bug 2: False Callstack Depth Attack Vulnerability: False

	Transaction-Ordering Dependence (TOD): False Timestamp Dependency: False Re-Entrancy Vulnerability: False
Token.sol	EVM Code Coverage: 84.6% Integer Underflow: True Integer Overflow: True Parity Multisig Bug 2: False Callstack Depth Attack Vulnerability: False Transaction-Ordering Dependence (TOD): False Timestamp Dependency: False Re-Entrancy Vulnerability: False
library/BasicToken.sol	EVM Code Coverage: 99.5% Integer Underflow: False Integer Overflow: True Parity Multisig Bug 2: False Callstack Depth Attack Vulnerability: False Transaction-Ordering Dependence (TOD): False Timestamp Dependency: False Re-Entrancy Vulnerability: False
library/CanReclaimToken.sol	EVM Code Coverage: 97.6% Integer Underflow: False Integer Overflow: False Parity Multisig Bug 2: False Callstack Depth Attack Vulnerability: False Transaction-Ordering Dependence (TOD): False Timestamp Dependency: False Re-Entrancy Vulnerability: False
library/NonZero.sol	EVM Code Coverage: 100.0% Integer Underflow: False Integer Overflow: False Parity Multisig Bug 2: False Callstack Depth Attack Vulnerability: False Transaction-Ordering Dependence (TOD): False Timestamp Dependency: False Re-Entrancy Vulnerability: False
library/Ownable.sol	EVM Code Coverage: 99.5% Integer Underflow: False Integer Overflow: False Parity Multisig Bug 2: False Callstack Depth Attack Vulnerability: False Transaction-Ordering Dependence (TOD): False Timestamp Dependency: False Re-Entrancy Vulnerability: False

library/SafeERC20.sol	EVM Code Coverage: 100.0% Integer Underflow: False Integer Overflow: False Parity Multisig Bug 2: False Callstack Depth Attack Vulnerability: False Transaction-Ordering Dependence (TOD): False Timestamp Dependency: False Re-Entrancy Vulnerability: False
library/StandardToken.sol	EVM Code Coverage: 84.6% Integer Underflow: False Integer Overflow: True Parity Multisig Bug 2: False Callstack Depth Attack Vulnerability: False Transaction-Ordering Dependence (TOD): False Timestamp Dependency: False Re-Entrancy Vulnerability: False