

# Smart Contract Source Code Audit

Prepared for IOVLabs • October 2018



# Smart Contract Source Code Audit

Prepared for IOVLabs • October 2018

v2006

## 1. Table Of Contents

1. Table Of Contents

2. Executive Summary

3. Introduction

3.1. Audit Timeline

4. Summary Of Findings

5. Findings

RIF-001 - Contributors can transfer tokens before redeemed

RIF-002 - Anyone can steal tokens of shareholders

RIF-003 - Potential unexpected lockout states

RIF-004 - Redeem multiple contributors to same address increases paid bonus

RIF-005 - Missing visibility modifiers in TransferAndCall

RIF-006 - redeemToSameAddress does not return the expected value

RIF-007 - fromAsciiString does not fail for invalid addresses

6. Testing

7. Appendix

8. Disclaimer

## 2. Executive Summary

Between September and October 2018, IOVLabs engaged Coinspect to perform source code reviews of the RIF Token smart contracts. The objective of the audits was to evaluate the security of the smart contracts. During the assessments, Coinspect identified seven security issues. The high risk issues identified compromised the integrity of the token. External attackers could have abused RIF-002 to steal tokens belonging to shareholders, and initial contributors could have exploited RIF-004 to obtain bonus amounts higher than expected. Coinspect verified that **all the identified security issues were correctly fixed** in the revision `rc3` (git: 6194d7edca0abbcb5275350da7b225edd18b7573) of RIF Token contracts.

## 3. Introduction

The RIFToken is an ERC20/677 compatible token designed to run on the RSK smart contract platform.

Before the token starts working as such there is an initialization phase where the token is set up, IOVLabs and shareholders are awarded a number of tokens which are stored in individual lockup contracts which control how these are distributed in stages as the different milestones are reached. Additionally, contributors may be added which receive the tokens upon distribution, but are encouraged to keep them by a bonus payout divided in different time lapses. All of these tasks are performed by a contract called TokenManager, which is specifically authorized to perform such actions in the RIFToken contract.

Contributors must choose where should their tokens and bonuses be distributed at the time of redeeming them. They can simply specify that their tokens should be left on their original address, or they can choose to have the tokens redirected to a different address by sending a specially crafted message signed with their private key which specifies the new address. A third redeem method is available which can only be called by the RIFToken contract owner, which uses a message with the text "DELEGATION" signed by the original contributor public key. This message must be generated by the contributor in advance and kept safely stored in case they lose access to their original account.

After the distribution is over, the contract works as a regular ERC20 token, which also implements the [ERC677](#) transferAndCall as a backward compatible enhancement of ERC20.

### 3.1. Audit Timeline

In September 2018, IOVLabs engaged Coinspect to perform the first source code review of the RIF Token smart contracts. Coinspect identified four issues, two high risk issues, one medium risk and one low risk. The high risk issues identified compromised the integrity of the token. External attackers could have abused RIF-002 to steal tokens belonging to

shareholders, and initial contributors could have exploited RIF-004 to obtain bonus amounts higher than expected.

In October 2018, after the identified issues were addressed by the IOVLabs team, a new security audit of the contracts was performed. The fixes for the previous issues were verified and two new low risk issues and one medium issue were identified.

The initial audit included the rc-1 tag of a private Git repository up to commit e7d81ef93256a613faef540df9bffb5aac396b74, comprising the following Solidity files with their respective SHA-256 hash:

612911d9f5dd5976e58ea2bd8b93398aaa9d8f66a30e32520f7f67552cc016f0	Migrations.sol
76bb9fea09e190e04cb7e8546d5ed39871405fc817e14558c0b971ab27aec352	ERC223/BasicERC223.sol
91ee52d0be81ab83d0b7852fb474a4284beb0aaecbea0c779a98b56f633d8119	ownership/TemporaryOwnable.sol
5928aa91e8f6e5813e94085e9cefe0e64a9b3e9a5eb62bc2e1ff28b0d10d597a4	RIF/AddressLinker.sol
5d369ca650b14808f905182e7b7fbbc4ef7f71c614d0627c34383d1e9cad17fb	RIF/Contributions.sol
806a1bb49de24c84a8877c7696c8261f8eb704ab3c9eb3cc5278244c2220f040	RIF/LockupAccount.sol
50b6ddde3afc751ba52676d0986eb963a4f599bbe0d62012b55eb87b9043f752	RIF/PreSale.sol
aa11de0b2157b00f1a8c08b07aecc41f859510279e4202052aef334217a5b761	RIF/RIFToken.sol
106382525232058357b7782d7e6823f4c1358724961e53984d27382e1367da21	RIF/Shareholders.sol
7731bb8529770d7e82fbb0d07b140f47dd55de697db04b17e5a22f1262ea9ef9	RIF/TokenManager.sol
b78e3540f99e491c098aba6c66dc52d579707243d83a48bfd4e30055a14c92bc	util/AddressHelper.sol
02e93d6b435b391e518316034bffd3643d05822fa51a51071fa77488c09d70d	util/ECRecovery.sol
third-party/ cb03353de178d1918772e1ba8190f435f35ab37e45c1a7e61a7cf2abae1d000f	ERC223/ContractReceiver.sol
53c4b57591897575938d7ace7c62b8376cb284d38efd1b76137c55594596d507	ERC223/ERC223.sol
third-party/openzeppelin/ ded51820c1b27c42946247c0c8e5947012f0c0283020a49beb044b2b3e99d94f	check-scripts.sh
9f0f23c056677dcd933c2d0f0a4b69f68f04bca3b73dd0984cd72a4e1e8cc462	lifecycle/Destructible.sol
a95bf355855dc532a61db3016d196da9f82fbc20b19d2341ae6c571b38b967e2	lifecycle/Pausable.sol
5799af0837f330c6cb09e28f2d3034149ca6ca9616eb77d395eed9285405a414	lifecycle/TokenDestructible.sol
fce117495e123ae259daa4357467aa39ab2fde8d8c405ea6e3a3d9c1d1888855	math/Math.sol
2992be99ec79983fab97b08158bbad475f55e02ec8c5293d663fa124a9b75c66	math/SafeMath.sol
663c278d96c39b144b0c334a4d0df1d873b2063fcc72c769e4b163595cf12900	ownership/Claimable.sol
4b27fe9f08c170b4e1e193ecc2223a89802828caf1273fccaa0f04239af8ff97	ownership/Contactable.sol
e134f31c45d9c69e082399be7659a526bb023cc74180b522aba3ef7b63bda0e	ownership/DelayedClaimable.sol
20d9c902314825c533b9737c46a85b6d363ae499685da068b282e0a059665ecb	ownership/HasNoContracts.sol
a84fb3cd22778e8c0c7703182b137f752e08123eb41d2a7786e2b86095e84aa2	ownership/HasNoEther.sol
35feff96ea2ff782dfa0d35815b2d394cfff31bbb2270b77aab4abac1bf6e4b9b	ownership/Ownable.sol
da19ee1ff8357b2ad58052d25858348528f29677817788ed918eddaa67b49922	token/ERC20/BasicToken.sol
91e5ad2bfd2aac60ffcc40a274751cdf0de31d91b69423e97f07e2fc6ab3e43c	token/ERC20/DetailedERC20.sol
8f09e53364787fdff9a9f701c4c5c35b8786d26aed5cce84ca460cd854e8a130	token/ERC20/ERC20Basic.sol
1570b37daa43d61c3f045639f96f63ca687ac0a8444ff944cd1ed1c33c0141e9	token/ERC20/ERC20.sol
acd133a147acb788b7cf8bd0fa87bbc734c2fbb58ba202a9b3134d6931f2c6f66	token/ERC20/MintableToken.sol
e9ccb3aa3c3b5c5ad4c9aa9ce3091584360214ae13332650d1ab27d6995d4c6a	token/ERC20/PausableToken.sol
31223bf5aa427aae272951b3f95d3a4b1c78cac0270284b19f3e5dd8112ad400	token/ERC20/SafeERC20.sol
17f3420015158148d711851a1f8a266ca417d25645f8ad37569ed9ba34ad351b	token/ERC20/StandardToken.sol
a9a33ad845aa436b53f425d22701c6f9296d9ffbbaea8c31c8315fc39e0db892	token/ERC20/TokenVesting.sol

The second audit included the rc-2 tag of a private Git repository up to commit b045729bb556fd02399ca5ca3ddc247b5e35d908, comprising the following Solidity files with their respective SHA-256 hash:

612911d9f5dd5976e58ea2bd8b93398aaa9d8f66a30e32520f7f67552cc016f0	Migrations.sol
a7b00e2c4118c6dbdb0a6e98d82c9a67033fb0df1469a8897c8a4c369c97e1b2e	ERC677/ERC677TransferReceiver.sol
bb676e6d9c6f6efa2d9a6e72aa113537ddd0082482602504ddd22b8a5a77ce5	RIF/AddressLinker.sol
5d369ca650b14808f905182e7b7fbbc4ef7f71c614d0627c34383d1e9cad17fb	RIF/Contributions.sol
8ac86c3d229bd5f7a7e68e8fc73415837e5e99bf4d6c30fc44f9b3374dbaf2de	RIF/LockupAccount.sol
50b6ddde3afc751ba52676d0986eb963a4f599bbe0d62012b55eb87b9043f752	RIF/PreSale.sol
6d4a4f568ace981444a08f8ea8400e65e242db9052dd0a2d4e901ef521e0293a	RIF/RIFToken.sol

106382525232058357b7782d7e6823f4c1358724961e53984d27382e1367da21	RIF/Shareholders.sol
0ac192208a6ee6f336a1247d3b1df7dad539b94dff7e7057815dcdabbbb55189	RIF/TokenManager.sol
048edc94dad6d7692fa59f1149d92b06f7f4db6bd92ef3a9979eb9540e3509f8	util/AddressHelper.sol
02e93d6b435b391e518316034bffd3643d05822fa51a51071fa77488c09d70d	util/ECRecovery.sol
third-party/openzeppelin/	
ded51820c1b27c42946247c0c8e5947012f0c0283020a49beb044b2b3e99d94f	check-scripts.sh
9f0f23c056677dcd933c2d0f0a4b69f68f04bca3b73dd0984cd72a4e1e8cc462	lifecycle/Destructible.sol
5799af0837f330c6cb09e28f2d3034149ca6ca9616eb77d395eed9285405a414	lifecycle/TokenDestructible.sol
fce117495e123ae259daa4357467aa39ab2fde8d8c405ea6e3a3d9c1d1888855	math/Math.sol
2992be99ec79983fab97b08158bbad475f55e02ec8c5293d663fa124a9b75c66	math/SafeMath.sol
663c278d96c39b144b0c334a4d0df1d873b2063fcc72c769e4b163595cf12900	ownership/Claimable.sol
4b27fe9f08c170b4e1e193ecc2223a89802828caf1273fccaa0f04239af8ff97	ownership/Contactable.sol
e134f31c45d9ce69e082399be7659a526bb023cc74180b522aba3ef7b63bda0e	ownership/DelayedClaimable.sol
20d9c902314825c533b9737c46a85b6d363ae499685da068b282e0a059665ecb	ownership/HasNoContracts.sol
a84fb3cd22778e8c0c7703182b137f752e08123eb41d2a7786e2b86095e84aa2	ownership/HasNoEther.sol
35feff96ea2ff782dfa0d35815b2d394cff31bbb2270b77aab4abac1bf6e4b9b	ownership/Ownable.sol
da19ee1ff8357b2ad58052d25858348528f29677817788ed918eddaa67b49922	token/ERC20/BasicToken.sol
91e5ad2bfd2aac60ffcc40a274751cdf0de31d91b69423e97f07e2fc6ab3e43c	token/ERC20/DetailedERC20.sol
1570b37daa43d61c3f045639f96f63ca687ac0a8444ff944cd1ed1c33c0141e9	token/ERC20/ERC20.sol
8f09e53364787dfdf9a9f701c4c5c35b8786d26aed5cce84ca460cd854e8a130	token/ERC20/ERC20Basic.sol
acd133a147acb788b7c7f8bd0fa87bbc734c2fbb58ba202a9b3134d6931f2cf66	token/ERC20/MintableToken.sol
31223bf5aa427aae272951b3f95d3a4b1c78cac0270284b19f3e5dd8112ad400	token/ERC20/SafeERC20.sol
17f3420015158148d711851a1f8a266ca417d25645f8ad37569ed9ba34ad351b	token/ERC20/StandardToken.sol
a9a33ad845aa436b53f425d22701c6f9296d9ffbbaea8c31c8315fc39e0db892	token/ERC20/TokenVesting.sol

Coinspect verified that **all the identified issues were fixed in the rc-3 tag** of a private Git repository up to commit 6194d7edca0abbcb5275350da7b225edd18b7573, comprising the following Solidity files with their respective SHA-256 hash:


612911d9f5dd5976e58ea2bd8b93398aaa9d8f66a30e32520f7f67552cc016f0	./Migrations.sol
a7b00e2c4118c6ddb0a6e98d82c9a67033fb0df1469a8897c8a4c369c97e1b2e	./ERC677/ERC677TransferReceiver.sol
6397f30f7d5800ab0195e0859ff249bcd9fd3917aa46757a28665cad587a74af	./util/AddressHelper.sol
02e93d6b435b391e518316034bffd3643d05822fa51a51071fa77488c09d70d	./util/ECRecovery.sol
5d369ca650b14808f905182e7b7fbbc4ef7f71c614d0627c34383d1e9cad17fb	./RIF/Contributions.sol
d371155257a94f3e943e9156b34163f3ef95a3f9d3e38139ebb2c1e0db34ccd4	./RIF/AddressLinker.sol
8ac86c3d229bd5f7a7e68e8fc73415837e5e99bf4d6c30fc44f9b3374dbaf2de	./RIF/LockupAccount.sol
67ac5d9bbaff0eb910b2a0f3c6e8048cac682d6fe5910169b95255c347ea53f0	./RIF/RIFToken.sol
a2c33f257ee4dc099f9f40484e2805c4679efda5aeceac9a47d5ac46b1e805ea	./RIF/TokenManager.sol
106382525232058357b7782d7e6823f4c1358724961e53984d27382e1367da21	./RIF/Shareholders.sol
50b6dde3af751ba52676d0986eb963a4f599bbe0d62012b55eb87b9043f752	./RIF/PreSale.sol
./third-party/openzeppelin/	
663c278d96c39b144b0c334a4d0df1d873b2063fcc72c769e4b163595cf12900	ownership/Claimable.sol
4b27fe9f08c170b4e1e193ecc2223a89802828caf1273fccaa0f04239af8ff97	ownership/Contactable.sol
e134f31c45d9ce69e082399be7659a526bb023cc74180b522aba3ef7b63bda0e	ownership/DelayedClaimable.sol
a84fb3cd22778e8c0c7703182b137f752e08123eb41d2a7786e2b86095e84aa2	ownership/HasNoEther.sol
35feff96ea2ff782dfa0d35815b2d394cff31bbb2270b77aab4abac1bf6e4b9b	ownership/Ownable.sol
20d9c902314825c533b9737c46a85b6d363ae499685da068b282e0a059665ecb	ownership/HasNoContracts.sol
1570b37daa43d61c3f045639f96f63ca687ac0a8444ff944cd1ed1c33c0141e9	token/ERC20/ERC20.sol
8f09e53364787dfdf9a9f701c4c5c35b8786d26aed5cce84ca460cd854e8a130	token/ERC20/ERC20Basic.sol
acd133a147acb788b7c7f8bd0fa87bbc734c2fbb58ba202a9b3134d6931f2cf66	token/ERC20/MintableToken.sol
a9a33ad845aa436b53f425d22701c6f9296d9ffbbaea8c31c8315fc39e0db892	token/ERC20/TokenVesting.sol
17f3420015158148d711851a1f8a266ca417d25645f8ad37569ed9ba34ad351b	token/ERC20/StandardToken.sol
91e5ad2bfd2aac60ffcc40a274751cdf0de31d91b69423e97f07e2fc6ab3e43c	token/ERC20/DetailedERC20.sol
31223bf5aa427aae272951b3f95d3a4b1c78cac0270284b19f3e5dd8112ad400	token/ERC20/SafeERC20.sol
da19ee1ff8357b2ad58052d25858348528f29677817788ed918eddaa67b49922	token/ERC20/BasicToken.sol
9f0f23c056677dcd933c2d0f0a4b69f68f04bca3b73dd0984cd72a4e1e8cc462	lifecycle/Destructible.sol
5799af0837f330c6cb09e28f2d3034149ca6ca9616eb77d395eed9285405a414	lifecycle/TokenDestructible.sol
2992be99ec79983fab97b08158bbad475f55e02ec8c5293d663fa124a9b75c66	math/SafeMath.sol
fce117495e123ae259daa4357467aa39ab2fde8d8c405ea6e3a3d9c1d1888855	math/Math.sol

The content of the files Contributors[1-5].sol with addresses and balance of contributors was not reviewed.

## 4. Summary Of Findings

ID	Description	Risk	Fixed
RIF-001	Contributors can transfer tokens before redeemed	Medium	✓
RIF-002	Anyone can steal tokens of shareholders	High	✓
RIF-003	Potential unexpected lockout states	Low	✓
RIF-004	Redeem multiple contributors to same address increases paid bonus	High	✓
RIF-005	Missing visibility modifiers in TransferAndCall	Low	✓
RIF-006	redeemToSameAddress does not return the expected value	Low	✓
RIF-007	fromAsciiString does not fail for invalid addresses	Medium	✓

## 5. Findings

RIF-001 Contributors can transfer tokens before redeemed		
Total Risk <b>Medium</b>	Impact Medium	Location RIFToken.sol
Fixed 	Likelihood Medium	

### Description

Contributors are not supposed to be able to move their tokens before they redeem them. But function `transferFrom` in `RIFToken` doesn't enforce this:

```
function transferFrom(address _from, address _to, uint256 _value) public
returns (bool) {
    bool result = super.transferFrom(_from, _to, _value);
    if (!result) return false;

    doTrackMinimums(_from);

    return true;
}
```

And `RIFToken` inherits function `approve` from `StandardToken.sol`:

```
function approve(address _spender, uint256 _value) public returns (bool) {
    allowed[msg.sender][_spender] = _value;
    emit Approval(msg.sender, _spender, _value);
    return true;
}
```

A contributor could take advantage of this vulnerability to move funds before redeeming them, by calling `approve` with a destination address and then calling `transferFrom` to move token to the destination address.

There were already tests in place for `approve` and `transferFrom` to make sure this is not possible, but the tests were broken.

### Recommendations

Add checks in the `approve` function as well as `increaseApproval` and `decreaseApproval` to require that contributors calling the function had already redeemed the tokens.



It is advisable to add similar checks in the `transferFrom` function too, even though it should be unnecessary if the functions changing allowance already do the checks.

`RIFToken` inherits from `StandardToken`, and re-implements some functions in order to add controls. Special care must be taken to be sure that controls are added to all functions that need it, or it might be possible to bypass controls by using alternative inherited functions (such as `approve` and `increaseApproval`).

Additionally, check that the addresses specified in these functions are not ones that have been redirected to a new address, as in this case the funds may be lost.

## RIF-002 Anyone can steal tokens of shareholders

Total Risk

**High**

Fixed



Impact

**High**

Likelihood

**High**

Location

TokenManager.sol

### Description

The contract `TokenManager` implements the function `setShareholderAddress` to assign a shareholder wallet address to an available token distribution. This function has no access controls, anyone could call it to steal tokens destined to shareholders.

### Recommendations

Add the `onlyOwner` modifier to the function `setShareholderAddress`.

## RIF-003 Potential unexpected lockout states

Total Risk <b>Low</b>	Impact Medium	Location RIFToken.sol
Fixed 	Likelihood Low	

### Description

The contract RIFToken implements two access control rules where some functions may only be called by the contract owner and some others may only be called by the `authorizedManagerContract`.

The contract owner is used to deploy and setup the token so that it works correctly, and one of the required tasks is to assign an `authorizedManagerContract`. The `authorizedManagerContract` is assigned to another contract which is used after the deploy to manage the different actions required by the contract such as transfer funds to contributors and shareholders, redirect funds from one user address to another and pay bonuses.

The contract owner is also capable of disabling the functions of the `authorizedManagerContract` by calling the `disableManagerContract` function. However, once this function is called once, a new manager contract can not be set. If this function is called before the manager contract completes all the tasks it is supposed to complete, some critical tasks such as bonus payments, unclaimed tokens recovery will never be completed.

### Recommendations

Ensure the `disableManagerContract` function is not called before the `authorizedManagerContract` completes all the tasks it is required to do.

## RIF-004 Redeem multiple contributors to same address increases paid bonus

Total Risk  
**High**

Fixed  
✓

Impact  
High

Location  
RIFToken.sol

Likelihood  
Medium

### Description

The contract RIFToken implements several functions to perform the redeem of the funds of a contributor. The function `contingentRedeem` allows the contract owner to assign an address different from the contributors' original address as its redeem address.

When performing this function, the value of the variable `minimumLeftFromSale` which is used to calculate the bonus assigned to each contributor is changed. Furthermore, if a second contributor chooses to redeem to the same address, the value of said variable will be overwritten:

```
function contingentRedeem(
    (...))

    // Now we must move the funds from the old address to the new address
    minimumLeftFromSale[redeemAddress] =
minimumLeftFromSale[contributorAddress];
    minimumLeftFromSale[contributorAddress] = 0;
    (...)
```

Taking advantage of this, a contributor may cheat in order to get more bonus tokens. The steps to follow to achieve that are the following:


1. Contributor A (with `minimumLeftFromSale[A]=100`) is redeemed to address Z:  
-> `minimumLeftFromSale[Z] = minimumLeftFromSale[A] = 100`
2. Contributor B (with `minimumLeftFromSale[B]=500`) is redeemed to address Z:  
-> `minimumLeftFromSale[Z] = minimumLeftFromSale[B] = 500`
3. `payBonus` is called in the `TokenManager` contract. Now the bonus for Contributor A is calculated and should be `100*bonus_percentage`, but as this is calculated using `minimumLeftFromSale[Z]` which is now 500. The bonus will be of `500*bonus_percentage`. Then the bonus for Contributor B is calculated, and the bonus is as expected `500*bonus_percentage`.

Step 1 can be repeated N-times with different contributor addresses before performing Step 2 using the address with the highest contribution amount, in order to maximize the amount of extra tokens awarded to all of them. An attacker could create a smart contract to exploit this weakness and invite contributors to redeem to the attacker's contract and share the benefit.

## Recommendations

Either forbid one address to be used by two different contributions to redeem, or track the `minimumLeftFromSale` variable from the original contributor address in order to prevent problems at the time of calculating the bonus.

## RIF-005 Missing visibility modifiers in TransferAndCall

Total Risk <b>Low</b>	Impact Low	Location RIFToken.sol
Fixed 	Likelihood Low	

### Description

The public method `TransferAndCall` in the `RIFToken` contract lacks visibility modifiers, such as `public`. It's a good smart-contract programming practice to clearly distinguish between private and public methods, to prevent mistakes.

### Recommendations

Add the `public` modifier to the public methods.

## RIF-006 redeemToSameAddress does not return the expected value

Total Risk

Low

Fixed



Impact

Low

Likelihood

Low

Location

RIFToken.sol

### Description

The method `redeemToSameAddress` in the `RIFToken` contract specifies that a `bool` result will be returned, however there is no `return` statement in the function. As it is a public method, a contributor may include a call to that method within his own code expecting a `bool` result which will never be returned.

### Recommendations

Add the `"return True"` statement to the function in case it runs correctly.

## RIF-007 fromAsciiString does not fail for invalid addresses

Total Risk  
**Medium**

Fixed



Impact  
Medium

Likelihood  
Medium

Location  
AddressHelper.sol

### Description

The function `AddressHelper.fromAsciiString` accepts invalid hexadecimal strings and returns an `address()` with a 0 in the position of each invalid hexadecimal character. This function is used to obtain the destination address for the tokens when the `RIFToken.redeem` function is called by a contributor. When this redeem method is used, the contributor signs a message containing the destination RSK address encoded in hexadecimal and calls `RIFToken.redeem` passing the address as a string and the `r,s` and `v` values of the signature.

If the contributor makes a mistake and signs an address that is not even a valid hexadecimal string, the error is not detected by the contract and the funds are lost.

### Recommendations

Revert the transaction when an invalid address string is passed to `RIFToken.redeem`.



## 6. Testing

The contracts are accompanied by a good set of tests. Coinspect reviewed the tests, and found some problems that prevented some tests of running properly. In fact, finding RIF-001 should have been spotted by a test case but the test didn't run properly and was marked as *passed*.

The problems are some improper checks to assert that a function call throws an exception. For example in RIFTokenTransfer\_test.js:

```
it('cannot transferFrom to a contributor', async function () {
  [...]
  try {
    await this.token.transferFrom(shareholderAccount,
      contributorAccount, 200, { from: anotherAccount });
    assert.fail();
  } catch (ex) {
  }
});
```

This code is incorrect because `assert.fail()` works by throwing an exception that the test runner is expecting to catch to mark the test as failed, but the try/catch sequence included in the test code actually catches the exception and the test runner never gets it, so the test always passes.

Another problem with using a try/catch in this fashion is that it might hide other errors in the tests. For example in RIFToken\_test.js:

```
it('cannot disable track minimum after release ownership', async function
() {
  await this.token.releaseOwnership();
  try {
    await this.token.disableTrackMinimum();
    assert.fail();
  } catch (ex) {
  }
});
```

The test above calls the function `disableTrackMinimum`, but that function doesn't exist in the contract (the correct name is `disableTrackMinimums`), and this throws an error that is caught and ignored. Similarly, tests 'only owner can disable redeem' and 'only owner can disable manager contract' fail silently (because they call functions with an extra parameter that no longer exists in the contracts), but the problems are hidden by the try/catch.

In order to check function throws, it is appropriate to use `expectThrow` as in tests for `LookupAccount` and `TokenManager`, for example:

```
it('cannot transferFrom to a contributor', async function () {
  await this.token.transferToShareholder(shareholderAccount, 1000, {
    from: managerContract });
  await this.token.transferToContributor(contributorAccount, 1000, {
    from: managerContract });
  await this.token.approve(anotherAccount, 200, { from:
    shareholderAccount });
  await expectThrow(this.token.transferFrom(shareholderAccount,
    contributorAccount, 200, { from: anotherAccount }));
});
```

It is recommended to increase test coverage. For example, in RIF-001 it was found that a contributor can use approve/transferFrom before redeeming the tokens. After fixing the try/catch problem, two existing tests find this issue. But the increaseApproval function is not tested at all, and it serves the same purpose as approve. Similarly, some tests about the behaviour of transfers don't cover the several flavors of transfer and transferFrom.

The test "can recover no beneficiary shareholders" (included in TokenManager.tests.js) is not actually checking anything as no comparison or expected result is stated in order to verify the contracts are behaving correctly. We recommend completing this test.

## 7. Appendix

In order to confirm issue RIF-004, a truffle test was developed. Please note that to make this test work it's necessary to modify the `contingentRedeem` function in the `RIFToken` contract to the following, just to remove the signature checking part of the function:

```
function contingentRedeem(address contributorAddress,
    address redeemAddress) public onlyTemporaryOwner returns (bool) {

    if (!redeemAllowed) return false;

    // only an original contributor could be redeemed
    if (!isInitialContributor[contributorAddress]) return false;

    // avoid to redeem a already accepted or redeemed address
    if (isRedeemed[contributorAddress]) return false;

    // Now we must move the funds from the old address to the new address
    minimumLeftFromSale[redeemAddress] =
minimumLeftFromSale[contributorAddress];
    minimumLeftFromSale[contributorAddress] = 0;

    // Mark as redirected and redeemed
    redirect[contributorAddress] = redeemAddress;
    isRedeemed[contributorAddress] = true;

    // Once the contributorAddress has moved the funds to the new RSK address,
    what to do with the old address?
    // Users should not receive RIFs in the old address from other users. If
    they do, they may not be able to access
    // those RIFs.
    return transferAll(contributorAddress, redeemAddress);
}
```

To verify the existence of the issue you may run the following test:

```
const { expectThrow } = require('../helpers/expectThrow');
const { latestTime } = require('../helpers/latestTime');
const { increaseTimeTo, duration } = require('../helpers/increaseTime');
const { ethGetBlock, ethGetCode } = require('../helpers/web3');
const { zeroes, addr } = require('../helpers/util');
const { getTokenDistributions } = require('../helpers/tokenManager');

const BigNumber = web3.BigNumber;

require('chai')
    .use(require('chai-bignumber')(BigNumber))
    .should();

const LockupAccount = artifacts.require('LockupAccount');
const RIFToken = artifacts.require('RIFTokenForTest');
```

```

// We use a version of the token manager that allows mocking the pre-sale
// information
const TokenManager = artifacts.require('TokenManager');
const PreSale = artifacts.require('MockablePreSale');

const ZERO_ADDRESS = `0x${zeroes(40)}`;

const EXPECTED_TOTAL_SUPPLY = new BigNumber(1e+27);
const EXPECTED_VESTING_PARAMETERS = {
  shareholder: {
    initialInstallments: 0,
    cliff: 6,
    installments: 42,
    installmentDuration: (365/12)*24*60*60, // # of seconds in a 365/12 day
month
    recoveryTime: 6*(365/12)*24*60*60
  },
  riflabs: {
    initialInstallments: 1,
    cliff: 0,
    installments: 59,
    installmentDuration: (365/12)*24*60*60, // # of seconds in a 365/12 day
month
    recoveryTime: 0
  }
}

const EXPECTED_MONTH_TIME = duration.hours(730);
const EXPECTED_BONUSES = {
  STAGE_ONE: 0.2,
  STAGE_TWO: 0.05,
  STAGE_THREE: 0.05
}
const EXPECTED_BONUS_TIMES = {
  STAGE_ONE: 3*EXPECTED_MONTH_TIME,
  STAGE_TWO: 6*EXPECTED_MONTH_TIME,
  STAGE_THREE: 9*EXPECTED_MONTH_TIME
}

const EXPECTED_RECOVERY_TIME = duration.days(365);
const EXPECTED_LOCKUP_RECOVERY_TIME = duration.days(180);
const RECOVERY_ADDRESS = addr("0xff");

const KIND = {
  RIFLABS: "riflabs",
  CONTRIBUTOR: "contributor",
  SHAREHOLDER: "shareholder"
};

const MIN_GAS_NEEDED_DISTRIBUTE_LOOP = 1000000;
const MIN_GAS_NEEDED_DISTRIBUTE_CALL = 230000;
const MIN_GAS_NEEDED_BONUS_LOOP = 250000;
const MIN_GAS_NEEDED_RECOVER_FUNDS_LOOP = 250000;
const MIN_GAS_NEEDED_RECOVER_SHAREHOLDERS_LOOP = 50000;

```

```

var assertDistributions = (tokenDistributions, kind, howMany) => {
  var filteredDistributions = tokenDistributions.filter(d => d.kind === kind);
  var filteredMocks = mockedData.filter(d => d.kind === kind);
  howMany = howMany != null ? howMany : filteredMocks.length;

  (filteredDistributions.length).should.equal(howMany);

  for (var i = 0; i < howMany; i++) {
    var mock = filteredMocks[i];
    filteredDistributions[i].beneficiary.should.equal(addr(mock.address));
    if (kind !== KIND.CONTRIBUTOR) {
      filteredDistributions[i].escrow.should.not.equal(ZERO_ADDRESS);
    } else {
      filteredDistributions[i].escrow.should.equal(ZERO_ADDRESS);
    }
    filteredDistributions[i].amount.should.bignumber.equal(mock.amount);
  }
};

var mockedData = [
  { kind: KIND.RIFLABS, address: RECOVERY_ADDRESS, amount: 250000 },
  { kind: KIND.CONTRIBUTOR, address: '0xaa', amount: 100 },
  { kind: KIND.CONTRIBUTOR, address: '0xbb', amount: 500 },
];

contract('TokenManager', function ([_, owner, payer, other, yetother]) {
  before(async function() {
    // Give the owner enough gas to run all the tests
    await web3.eth.sendTransaction({ from: other, to: owner, value:
989717679000000 });
    await web3.eth.sendTransaction({ from: yetother, to: owner, value:
989717679000000 });
  });

  context('Conspect is', function () {
    it('testing bonus cheats', async function () {
      // deploy token and presale contracts
      this.token = await RIFToken.new({ from: owner });
      this.presale = await PreSale.new({ from: owner });

      // load riflabs and shareholder mockedData in PreSale contract
      await this.presale.setRifLabs(mockedData[0].address,
mockedData[0].amount, { from: owner });
      for (var i = 1; i < mockedData.length; i++) {
        var mock = mockedData[i];
        switch(mock.kind) {
          case KIND.SHAREHOLDER:
            await this.presale.addShareholder(mock.address, mock.amount,
{ from: owner });
            break;
          case KIND.CONTRIBUTOR:
            await this.presale.addContributor(mock.address, mock.amount,
{ from: owner });

```

```

        break;
    }
}

// deploy token manager and add it as the manager contract for RIFToken
this.tokenManager = await TokenManager.new(this.token.address,
this.presale.address, { from: owner });
    await this.token.setAuthorizedManagerContract(this.tokenManager.address,
{ from: owner });

// distribute tokens
this.expectedDistributionBlockTime = null;
this.numberOfTransactions = 0;
while(!(await this.tokenManager.hasDistributed())) {
    txReceipt = await
this.tokenManager.distributeTokens(MIN_GAS_NEEDED_DISTRIBUTE_LOOP, { from: owner
});
    this.numberOfTransactions++;
    if (!this.expectedDistributionBlockTime) {
        this.expectedDistributionBlockTime = (await
ethGetBlock(txReceipt.receipt.blockNumber)).timestamp;
    }
}
    this.tokenDistributions = await
getTokenDistributions(this.tokenManager);

// make sure it finished distributing
this.distributionTime = null;
while(!(await this.tokenManager.hasDistributed())) {
    txReceipt = await
this.tokenManager.distributeTokens(MIN_GAS_NEEDED_DISTRIBUTE_LOOP, { from: owner
});
    if (!this.distributionTime) {
        this.distributionTime = (await
ethGetBlock(txReceipt.receipt.blockNumber)).timestamp;
    }
}
//(await this.tokenManager.hasDistributed()).should.be.true

// make sure balances are OK
(await this.token.balanceOf('0xaa')).should.bignumber.equal(100);
(await this.token.balanceOf('0xbb')).should.bignumber.equal(500);

// redeem for both contributors to the same address
await this.token.contingentRedeem('0xaa', '0xcc', { from: owner });
await this.token.contingentRedeem('0xbb', '0xcc', { from: owner });

//advance time until bonus can be payed, and try to pay them
this.distributionTime = (await
ethGetBlock(txReceipt.receipt.blockNumber)).timestamp;
    await increaseTimeTo(this.distributionTime +
EXPECTED_BONUS_TIMES.STAGE_ONE + duration.days(5));
    await this.tokenManager.payBonus(MIN_GAS_NEEDED_BONUS_LOOP);

```

```
        // now the balance for '0xcc' should be the contribution of both
        // 0xaa and 0xbb added and multiplied by the bonus percentage (20%).
        // (100 + 500) * 1.2
        (await this.token.balanceOf('0xcc')).should.bignumber.equal(720);
    });
});
});
```

## 8. Disclaimer

The present security audit is limited to smart contract code. It does not cover the technologies and designs related to these smart contracts, nor the frameworks and wallets that communicate with the contracts, nor the general operational security of the company whose contracts have been audited. This document should not be read as investment advice or an offering of tokens.