Smart Contract Source Code Audit rLending

Prepared for ThinkAndDev • February 2021

Final Report v210223

1. Table Of Contents

2. Executive Summary

In December 2020, Think & Dev engaged Coinspect to perform a source code review of rLending, a new borrowing and lending platform. The objective of the audit was to evaluate the security of their smart contracts.

rLending is based on the Compound Protocol and this audit focused on the modifications performed to it.

The changes introduced by rLending to the forked project did not introduce any vulnerabilities. Most issues reported in this document do not represent a high risk to the platform users.

However, Coinspect identified a single point of failure: the price feeds. Coinspect observed rLending is completely dependent on the MoC oracle infrastructure, which price feeds are blindly trusted.

In February 2021, Coinspect reviewed the fixes performed by rLending and new changes backported from the Compound project. All the previously reported findings have been correctly addressed but RLD-001 Absolute blind trust in MoC price feed oracle. In this case, the rLending team acknowledged the risks and decided to wait for the new OMoC oracle version to be deployed; this version is more decentralized and incorporates several improvements. The team will contemplate adding more sources of price feeds as these become available in the RSK network in the future. Also, two new issues were identified in this second phase of the audit.

This report has been updated to reflect the current project state.

The following issues were discovered during the two assessments:

High Risk	Medium Risk	Low Risk	Zero Risk
1	**1**	**1**	**5**

It is worth observing the project's governance model consists of a single 2-of-3 multi-signature contract that has unlimited power over all the project's smart contracts.

Several suggestions to help further improve rLending security are proposed throughout this document and a summary can be found in 5. Conclusions and Recommendations.

3. Introduction

rLending's main goal is to allow users to lend crypto currencies as collateral and to borrow crypto assets based on interest rates set by real-time supply and demand smart contracts on running the RSK network.

The audit started on December 8th, 2020 and was conducted on the https://github.com/riflending/rlending-protocol Git repository as of commit: `e6bf2cacdcde2d5e7ab3a2d8ca00bbdac2d02f7d` of **December 8th.**

```
commit e6bf2cacdcde2d5e7ab3a2d8ca00bbdac2d02f7d
Merge: 46a1972 4168c6e
Author: Pedrow <pmprete@users.noreply.github.com>
Date:   Tue Dec 8 11:21:11 2020 -0300

   Merge pull request #23 from riflending/fixOracleAdapterMoc

   fix PriceOracleAdapterMoC to return usdc value
```

A second review was performed in February 2021 as was conducted on the same repository as of commit:

```
commit dc6b071a27c92c367c21eb05ca553ede3d3cac10
Merge: ae507a7 9898d08
Author: Pedrow <pmprete@users.noreply.github.com>
Date:   Wed Feb 10 12:33:58 2021 -0300

    Merge pull request #32 from riflending/networks

    Add Rsk Mainnet network
```

rLending is implemented as a set of changes performed to a fork of the Compound Protocol version 2.8.13 (https://github.com/compound-finance/compound-protocol/tree/v2.8.1).

The project architecture, mostly inherited from Compound, is composed of the following components:

1. CTokens
2. Price Oracle
3. Comptroller
4. Governance
5. Interest rate

The scope of the first phase of this engagement was limited to the following pull requests as requested by ThinkAndDev:

- https://github.com/riflending/rlending-protocol/pull/13
- https://github.com/riflending/rlending-protocol/pull/14
- https://github.com/riflending/rlending-protocol/pull/16

The scope of the second phase of the engagement was limited to the following pull requests and commits as requested by the client:

- https://github.com/riflending/rlending-protocol/commit/3ee55a5b58ac2a5dcef35ab947be4be95acc7313#diff-f7ff27454e66afcb14698e79c6af290da8a66d623c60376a6ee86eadf5933ba8R400
- https://github.com/riflending/rlending-protocol/pull/31/commits/5db0eeb0c7281e4262e37564079878d193bb0d6b
- https://github.com/riflending/rlending-protocol/pull/31/commits/5db0eeb0c7281e4262e37564079878d193bb0d6b
- https://github.com/riflending/rlending-protocol/pull/31/commits/a3e8638a62fc2f80a621c2a02382706bdbee51e1
- https://github.com/riflending/rlending-protocol/pull/31/commits/e7045ce9b1dd7b375943c9e7c9c87ba05b5acbc6
- https://github.com/riflending/rlending-protocol/pull/31/commits/c6179f423b0d5edb62d086bfebc6ffd5b0269201
- https://github.com/riflending/rlending-protocol/pull/31/commits/aa49c6d45cb9b758d8fabb7fea0f28776278516e
- https://github.com/riflending/rlending-protocol/pull/31/commits/768d2c3e946b627bb221b98a06606f9cba4b1261
- https://github.com/riflending/rlending-protocol/pull/31/commits/aa49c6d45cb9b758d8fabb7fea0f28776278516e
- https://github.com/riflending/rlending-protocol/commit/7c7b83d24f804d15fb669f049cf65b2b6fa2ebd0

Neither Compound security nor the Money on Chain oracle infrastructure were evaluated during this audit.

4. Assessment

4.1. Phase 1

All contracts are compiled with solidity version 0.5.16. A few build errors are reported not related to rLending additions to the forked project.
A complete set of tests, including complex scenarios, are included in the project repository. These tests were adapted to the rLending platform and all the executed tests pass:

```
Test Suites: 2 skipped, 82 passed, 82 of 84 total
Tests:       38 skipped, 7 todo, 969 passed, 1014 total
```

Coinspect auditors were not able to obtain a coverage report as tests timeout when coverage is run as instructed in the documentation.

The following modifications introduced by rLending to the forked Compound repository were reviewed during this phase of the engagement.

1.1. RSK related updates

This changes were introduced by

- https://github.com/riflending/rlending-protocol/pull/13

This is the biggest set of changes, including 78 commits and 176 files changed.

These modifications were made:

- Changes in Price Oracles
- Renaming of cTokens
- Updated blocks per year
- Removal of unused features
- The new `BaseJumpRateModelV2` was introduced
- Added per cToken market borrow capping in `Comptroller.sol`
- Added `setCompAddress` function that allows the admin to change the rLEN token address in `Comptroller.sol`
- Oracles are managed by a “guardian” role. Guardians can change, this is initiated by the current guardian and accepted by the new one.
- Renamed cETH and it’s underlying Ether, to cRBTC and RBTC respectively (`contracts/RBTC.sol`)
- Renamed COMP token to rLEN (`contracts/Governance/RLEN.sol`)
- Removed support for DAI since it had a special CToken delegation and custom interest rate model rLending won’t be supporting at launch.
- Removed contracts: – contracts/CDaiDelegate.sol – contracts/DAIInterestRateModelV2.sol
- Removed support for Certora

- Halved the `blocksPerYear` constant and updated the relevant tests to accomodate for RSK slower block time (`contracts/JumpRateModel.sol` line 18 and `contracts/WhitePaperInterestRateModel.sol` line 19)

1.2. Cherry picking Compound updates

This changes were introduced by:

- https://github.com/riflending/rlending-protocol/pull/14

These modifications were made:

- Patch second order issues in governance token https://github.com/compound-finance/compound-protocol/commit/a1eb7a64d8b2e3e65149737d74572fed181020ab
- Added Tether Support and Gas Optimizations https://github.com/compound-finance/compound-protocol/commit/cce8c8836d21ac307df918a4ca46f0b83dbe2757
- Updated support for ComptrollerG4.sol -> esto es en realidad Add Market Borrow Caps https://github.com/compound-finance/compound-protocol/pull/65
- Refactored interest code to share code https://github.com/compound-finance/compound-protocol/pull/66
- Bumped the controller to the G4 version
- Added the `JumpRateModelV2` interest model that is used for stable coins

1.3. Gnosis support for initial setup

This changes were introduced by:

- https://github.com/riflending/rlending-protocol/pull/16

These modifications were made:

- Gnosis support for initial setup

rLending will use a multisig contract initially, based on Gnosis Multisig15 (https://github.com/gnosis/MultiSigWallet/blob/master/contracts/MultiSigWallet.sol) and will move to a Governance based scheme after some time on mainnet.

This means that the initial rLending deploy will have the Governance admin/guardian addresses pointing to Gnosis Multisig contract.

1.4. Oracles

rLending replaced Compound's original PriceProxyOracle functionality with a modified implementation that allows setting an independent oracle for each asset. The deployment script currently setups markets to utilize MoC oracle infrastructure.

Coinspect identified the use of oracles as a single point of failure. Because rLending is highly dependent on the price feeds for all its operations, and because there is only one configured source of off-chain data, if an invalid price is provided there is currently no way for rLending

to protect user funds from abuse. This is further detailed in Absolute blind trust in MoC price feed oracle.

1.5. Deployment

The rLending team developed a new deployment script located in `script/deploy/deploy.js`.
This script is responsible for deploying and initializing all the platform components. Currently, it deploys one oracle proxy with 3 adapters and their price providers for: RIF, RBTC and a mock one for DAI only for Testnet.
The original `WhitePaperInterestRateModel` is used for the cRIF and rBTC markets, and the newer `JumpRateModelV2` is used for the rDAI market.
The script stops the rLEN drip during setup by calling the `setCompRate` function with `compRate` set to zero.

Coinspect observed the script creates the Multisig contract which owns all the contracts deployed with only one signer as detailed in Insecure deployment: one signer multisig owns all contracts.

4.2. Phase 2

During February 2021, Coinspect reviewed several modifications performed by the rLending team since the initial audit. These changes include fixes to previously reported issues and a set of modifications backported from the Compound project repository.

A few build errors are reported by the Solidity compiler and are reported for informative purposes in Function state mutability can be restricted to pure, but do not represent an actual risk for rLending.

2.1 Remediations to findings from phase 1

All previously reported findings have been addressed but RLD-001. In that respect, the rLending team is aware of the potential problems the current oracle infrastructure represents and is waiting for the new MoC upgrade to their oracles in order to incorporate the new features. It is worth observing that this upgrade will still mean rLending relies on a unique source of price information and this could be a dangerous proposition for the platform users, but the team stated that more data sources will be added as the platform evolves.

2.2 rLending specific changes

The deployment script has been cleaned up. The new script uses the hardhat framework instead of buidler and was modified to fix the issue previously reported Insecure deployment script: one signer multisig owns all contracts.

The deployment script was updated to create a multisig wallet with 3 signers:

```
const owners =  isLocal ? [deployer] : [deployer, admin1, admin2]
const multiSigResult = await deploy("MultiSigWallet", {
```

```
        args: [owners, 1],
        contract: "MultiSigWallet",
        from: deployer,
        skipIfAlreadyDeployed: true
    }
```

And when the deployment is over, the minimum signature for a transaction requirement is changed to 2:

```
console.log("\n  changeRequirement Multisig ...")
let data = multiSigContract.interface.encodeFunctionData("changeRequirement",[2])
//submit transacion multisig
await multiSigContract.submitTransaction(multiSigContract.address, 0, data).then((tx) =>
    tx.wait())
```

However, auditors observed this will not work if the deployment script is passed an already existing wallet with only 1 signer. In that scenario, manual intervention will be needed to correct the mistake and leave the already deployed contracts in the intended state. This is further detailed in Insecure deployment script: one signer multisig owns all contracts.

Another modification introduced after the previous review is the replacing of cDAI with cUSDT. This change is observed in the deployment script, which now deploys cUSDT and the corresponding oracle adapter.

The new `deploy.js` uses the `setGuardian` function, recently added to the contracts, to transfer ownership to the multisig wallet right after each contract is created.

Additionally, Coinspect checked `package.json` included in the repo utilizes the correct name for the `@nomiclabs/hardhat-waffle` package which was subject to an malicious npm module attack on Feb 19: Hardhat on Twitter: "1/ PSA On Friday morning EST we identified an attempt to attack the Hardhat community via a malicious NPM package. Your private keys are most likely SAFU, but if you installed the hardhat-waffle plugin during the past week, read on.".

The oracles deployed in the current deployment script are mocks which always return the same value. Coinspect suggests the rLending platform should be tested when the new oracles are deployed and integrated into the project. To do so, a forked local version of the network could be used to replicate the real network with real oracles.

2.3 Changes from Compound project

A set of changes was backported from the Compound project:

- https://github.com/riflending/rlending-protocol/pull/30
- https://github.com/riflending/rlending-protocol/commit/7ff8bc4f08fa9501858038f6b7acfab8e84592d2
- https://github.com/compound-finance/compound-protocol/pull/83

The most important modifications introduced in these pull requests were:

- Comptroller contract was updated to version G6
- Saddle testing framework updates
- COMP token distribution speed changes

The changes in the following files were reviewed:

- `contracts/CCompLikeDelegate.sol`
- `contracts/CErc20Delegator.sol`
- `contracts/Comptroller.sol`
- `contracts/ComptrollerG5.sol`
- `contracts/ComptrollerG6.sol`
- `contracts/ComptrollerStorage.sol`
- `contracts/Exponential.sol`
- `contracts/ExponentialNoError.sol`

The modifications in the governance related contracts are not relevant for the rLending project. Also, the `COMP` distribution speed related functions were changed: these do not affect rLending as it is currently set to 0 in the deployment script.

Coinspect suggests removing code that is not going to be used in production. This will result in gas cost savings during deployment and daily operations for the managing wallet and the users as well.

The new `ComptrollerG5` and `ComptrollerG6` introduced in the repository were diffed to verify the rLending contracts match the ones in Compound.

2.4 New findings and suggestions

During this engagement, Coinspect auditors found the code taken from Compound was not always modified to take into account the differences between the Ethereum and RSK blockchains. Specifically, a finding is reported regarding the different error values returned by `ecrecover`. See Incorrect ecrecover return value checks for more details.

As a part of this engagement, Coinspect auditors reviewed the latest changes in the Compound repository. An interesting modification to the `CToken` contract was identified which enables the protocol owner to recover funds accidently sent by users to the contract:

https://github.com/compound-finance/compound-protocol/commit/b198cb4dac977c61fa793ffe441c932438e83cdc#diff-a7644339c2b78447d82ac87f7bf4499bdd1d539360b883b033c41a90a778923cR113

This safeguard was observed in other projects as well and it is recommended that it is included in rLending CTokens before deployment in order to further protect the platform users.

Regarding the platform's governance model, Coispect observed the project's documentation website (https://test.rlending.app/docs/securityAudits) clearly indicates the managing rights the platform owner can exercise:

The following rights in the protocol are controlled by the Multi-signature smart contract:

- The ability to list a new cToken market
- The ability to update the interest rate model per market
- The ability to update the oracle or any adapter address

- The ability to withdraw the reserve of a cToken
- The ability to update the Comptroller contract
- The ability to choose a new admin

In order to limit potential damages in case the multi-signature smart contract is compromised, it might be desirable to split admin rights between different roles represented by different multi-signature contracts.

In case of an emergency, it is important that the reaction time is minimal and enough signers are available to pause the contract promptly.

For other administrative changes in the protocol, Coinspect recommends using the `Timelock` mechanism used by Compound (implemented in `Timelock.sol` and used by `GovernorAlpha.sol`). By using the time lock, the protocol owner changes are more transparent to the protocol users, which are able to decide if they want to withdraw their funds before a change is applied. This should be considered as a way to advance the platform towards a more decentralized governance model.

5. Conclusions and Recommendations

The changes introduced to the forked Compound project did not introduce any security relevant vulnerabilities and were well documented.

Coinspect identified the project's oracle dependency as a single point of failure. The trust deposited on the MoC oracle infrastructure, without any defense in depth measures or fallback mechanisms, means any weakness or failure in the price feeds could result in funds from rLending users being stolen.

The following list sums up the most important recommendations from this audit:

1. Improve the oracle integration (diversify to multiple sources of information, secure fallbacks during network congestion and/or feed not recently updated) as this external dependency could be seen as the weakest link in the platform.
2. Fix testing coverage reporting..
3. Consider adding the `sweepToken` functionality recently incorporated into Compound in order to be able to recover funds incorrectly transferred by users (https://github.com/compound-finance/compound-protocol/commit/b198cb4dac977c61fa793ffe441c932438e83cdc)
4. Constantly monitor vulnerabilities reported in the upstream project and backport the changes as needed.
5. Consider removing the code inherited from Compound that is not used in the rLending platform (e.g., Governance token, voting, vote delegation) in order to save deployment and operational gas costs and reduce the attack surface.
6. The oracles deployed in the current deployment script are mocks which always return the same value. Coinspect suggests the rLending platform should be tested when the new oracles are deployed and integrated into the project. To do so, a forked local version of the network could be used to replicate the real network with real oracles.
7. Evaluate utilizing different wallets to control the different contracts in the platform in order to limit the damage if one of the wallets is compromised.
8. Evaluate leveraging the `Timelock` mechanism used by Compound in order to make protocol changes more transparent for users of the platform.
9. Improve the deployment script to guarantee an appropriate number of signers in the multisig contract. Though the current script is intended to create a 2 of 3 multisig wallet, the contracts could end up owned by a 1 signer only multisig if the script uses an existing multisig provided in the configuration instead of deploying a new wallet. The `deploy.js` script could check the configured wallet has a minimum number of signers

6. Summary of Findings

ID	Description	Risk	Fixed
RLD-001	Absolute blind trust in MoC price feed oracle	High	✘
RLD-002	Insecure deployment script: one signer multisig owns all contracts	Medium	✔
RLD-003	Insecure Testnet price oracle	Info	✔
RLD-004	SimplePriceOracle accepts price updates from anybody	Info	✔
RLD-005	PriceOracleAdapterCompound contract never used	Info	✔
RLD-006	mintVerify “defense hook” is not implemented	Info	✔
RLD-007	Function state mutability can be restricted to pure	Info	✘
RLD-008	Incorrect ecrecover return value checks	Low	✘

7. Remediations

During February 2021 Coinspect verified the findings that the rLending team decided to address had been correctly fixed.

The following table lists the findings from Coinspect previous audit that were fixed and the corresponding fix status and commit:

ID	Description	Status / Commit
RLD-001	Absolute blind trust in MoC price feed oracle	Will fix
RLD-002	Insecure deployment script: one signer multisig owns all contracts	3ee55a5b58ac2a5dcef35ab947be4be95acc7313
RLD-003	Insecure Testnet price oracle	3575377eb293bb7a456360ae9be4ef3fe26561af
RLD-004	SimplePriceOracle accepts price updates from anybody	570e8e059496fcc980934e3aee81f644f7a3820c
RLD-005	PriceOracleAdapterCompound contract never used	363304ff5fc874dfd787cf86f76259fa5b488021
RLD-006	“Defense hook” mintVerify is not implemented	d717d9dc277a756b6061fbc24033beb28e119ed6

RLD-001, the only finding from our previous audit that has not been already fixed, is waiting on the new MoC Oracle infrastructure to be deployed. rLending will continue to rely on MoC as its sole price feed provider.

More detailed status information can be found in each finding.

8. Findings

RLD-001	Absolute blind trust in MoC price feed oracle

Total Risk	Impact	Location
High	High	PriceFeedsMoC.sol
Fixed	Likelihood	
✘	High	

Description

rLending relies solely on the MoC oracles platform for its price feeds.

rLending has no protection mechanisms in place (besides the ability for the owner to pause the operations) in order to protect itself from rogue oracles and/or oracle failures and vulnerabilities. For that reason, the values provided by the only price source in use are considered the absolute truth. As a consequence, the rLending platform is completely dependent on the MoC oracle infrastructure as all operations depend on the price feed.

Coinspect auditors dedicated time to understanding how Compound, the base project for rLending, protects itself in the Ethereum network. The Compound Protocol verifies reported prices fall within an acceptable bound of the time-weighted average price provided by Uniswap v2. For example, if a price being reported deviates over 20% from a 30-60 minute time-weighted average of Uniswap prices (the “anchor”), it is ignored. This protection mechanism also has its drawbacks that should be considered, for example, this oracle will not be able to react to sudden price changes. The mechanism described is implemented in https://github.com/compound-finance/open-oracle/blob/master/contracts/Uniswap/UniswapAnchoredView.sol.

Recommendations

Coinspect recommends the rLending platform improves their integration of off-chain oracles:

1. Consider adding redundancy for price feeds: utilizing more than one different price sources and different oracle providers and comparing their values.
2. Research latest advances in blockchain oracle technology (such as the new Maker Oracle Security Modules and Compound Open Oracle)
3. Implement a rLending owned oracle infrastructure in order to avoid depending completely on other providers.
4. Evaluate using TWAP (Time Weighted Average Prices) and/or rejecting price changes with a deviation above a certain threshold.

Status

The rLending development team has decided to wait until the new upgrade to the MoC oracle infrastructure is performed. Also, they commented that Compound had a unique oracle source in their first version, and so they believe the current setup will be enough until the new decentralized MoC oracles are deployed. ***As a result of this decision, rLending will continue to trust and rely on the MoC oracles as price feed providers for their platform.***

RLD-002	Insecure deployment script: one signer multisig owns all contracts

Total Risk	Impact	Location
Medium	High	script/deploy/deploy.js
Fixed	Likelihood	
✔	Low	

Description

The rLending deployment script creates a multisig contract with only one signer:

```
//deploy MultiSigWallet
async function multiSigWallet() {
   if (multiSigAddress) {
       multiSig = await saddle.getContractAt('MultiSigWallet', multiSigAddress);
   } else {
       multiSig = await saddle.deploy('MultiSigWallet', [[root], 1]);
   }
   generateLogAddress('MultiSigWallet', multiSig._address);
};
```

This multisig owns all the contracts that comprise the rLending platform. If the root account is exposed the whole system is compromised.

The system deployment script does not add further signers to the multisig contract.

Moreover, in the current deployment procedure the root private key is stored in a configuration file in the system file system, if only this host is compromised the multisig will allow attackers to perform any actions in the system.

Recommendation

Deploy the multisig with a more suitable number of signers. At least, a 2 of 3 is recommended. Consider implementing a secret management solution using hardware wallets to isolate the private key from the host.

Status

This issue was fixed in commit 3ee55a5b58ac2a5dcef35ab947be4be95acc7313.

The new deployment script creates a multi-signature wallet with 3 owners (if it wasn't created previously and passed in the configuration):

```
// if not set by named config
if (!multiSig) {
    console.log("\n  Deploying MultiSigWallet...")
```

```
    const owners =  isLocal ? [deployer] : [deployer, admin1, admin2]
    const multiSigResult = await deploy("MultiSigWallet", {
        args: [owners, 1],
        contract: "MultiSigWallet",
        from: deployer,
        skipIfAlreadyDeployed: true
    })
    multiSig = multiSigResult.address
}
const multiSigContract = await ethers.getContractAt(
    "MultiSigWallet",
    multiSig,
    signer
)
```

And when the deployment is over, the minimum signature for a transaction requirement is changed to 2:

```
console.log("\n  changeRequirement Multisig ...")
let data = multiSigContract.interface.encodeFunctionData("changeRequirement",[2])
//submit transacion multisig
await multiSigContract.submitTransaction(multiSigContract.address, 0, data).then((tx) =>
     tx.wait())
```

Note that If an existing multisig with only one owner is used with the deployment script, the `changeRequirement` function call will revert and all the platform's contracts will end up being owned by a multisig with 1 owner until more owners are added and the minimum signatures requirement is changed manually.

RLD-003 Insecure Testnet price oracle

Total Risk	Impact	Location
Info	None	TestnetPriceOracleAdapterMoc.sol
Fixed	**Likelihood**	
✔	None	

Description

The `has` boolean is not checked when the price is obtained from the MoC price provider:

```
function assetPrices(address) public view returns (uint256) {
    (bytes32 price, bool has) = priceProviderMoC.peek();
    return uint256(price);
```

On the other hand, the `PriceOracleAdapterMoC` contract deployed in Mainnet is correct:

```
function assetPrices(address) public view returns (uint256) {
    (bytes32 price, bool has) = priceProviderMoC.peek();
    require(has, "PriceOracleAdapterMoc: Oracle have no Price");
    return uint256(price);
```

This finding does not represent an immediate risk as this oracle is intended to be used only in Testnet.

However, if inadvertently a developer uses this file to base a new implementation the vulnerable construct could end up being used in a scenario where real funds are at risk.

Recommendation

Check the boolean `has` value returned by the MoC oracle in order to guarantee the price provided is valid. Explicitly document this contract is not to be copied for any purpose when handling real funds.

Status

This finding was fixed in commit 3575377eb293bb7a456360ae9be4ef3fe26561af.

The adapter contract has been removed. Now, the `PriceOracleAdapterMoc` is used as in mainnet.

RLD-004	SimplePriceOracle accepts price updates from anybody

Total Risk	Impact	Location
Info	None	SimplePriceOracle.sol
Fixed	**Likelihood**	
✔	None	

Description

The oracle implemented in this contract has public functions that allow anybody to set/update the prices for tokens.

```
    function getUnderlyingPrice(CToken cToken) public view returns (uint) {
        if (compareStrings(cToken.symbol(), "cRBTC")) {
            return prices[(address(cToken))];
        } else {
            return prices[address(CErc20(address(cToken)).underlying())];
        }
    }

    function setUnderlyingPrice(CToken cToken, uint underlyingPriceMantissa) public
{
        address asset = address(CErc20(address(cToken)).underlying());
        emit PricePosted(asset, prices[asset], underlyingPriceMantissa,
underlyingPriceMantissa);
        prices[asset] = underlyingPriceMantissa;
    }

    function setDirectPrice(address asset, uint price) public {
        emit PricePosted(asset, prices[asset], price, price);
        prices[asset] = price;
    }

    // v1 price oracle interface for use as backing of proxy
    function assetPrices(address asset) external view returns (uint) {
        return prices[asset];
    }
```

This finding does not represent an immediate risk as this oracle is currently used only in tests.

However, if inadvertently a developer uses this file to base a new implementation the vulnerable construct could end up being used in a scenario where real funds are at risk. There is no indication in the contract name or source code to warn developers about the contract's purpose, and the file is not stored in the directory where the tests are located.

Recommendation

Explicitly document this contract is not to be copied for any purpose when handling real funds. Change the contract name to indicate is only intended for testing purposes. Also, store the contract in the tests directory.

Status

This finding was fixed in commit 570e8e059496fcc980934e3aee81f644f7a3820c.

`SimplePriceOracle` was moved to the mocks folder, and a comment was added to make it clear it is only intended for tests.

RLD-005	PriceOracleAdapterCompound contract never used

Total Risk	Impact	Location
Info	None	PriceOracleAdapterCompound.sol
Fixed	**Likelihood**	
✔	None	

Description

The `PriceOracleAdapterCompound` contract in the repository is not used.

Also, the `saiPrice` variable in the `PriceOracleAdapterCompound` contract is never used, and seems intended to be consulted by an off-chain component. It can only be set one time:

```
/// @notice Frozen SAI price (or 0 if not set yet)
uint256 public saiPrice;


/**
 * @notice Set the price of SAI, permanently
 * @param price The price for SAI
 */
function setSaiPrice(uint256 price) public {
    require(msg.sender == guardian, "only guardian may set the SAI price");
    require(saiPrice == 0, "SAI price may only be set once");
    require(price < 0.1e18, "SAI price must be < 0.1 ETH");
    saiPrice = price;
}
```

The code shows confidence in the SAI price value will not change in the future, however, this leaves nothing to do in the case the price changes in the future.

Recommendation

Remove the contract from the repository if it is not going to be used in rLending. Evaluate if the benefits of fixing the SAI price once forever is worth the consequences at risk if the price changes.

Status

This issue was fixed in commit 363304ff5fc874dfd787cf86f76259fa5b488021.

The contract was moved to the mocks folder as it is only used for tests.

RLD-006 mintVerify “defense hook” is not implemented

Total Risk
Info

Fixed
✔

Impact
None

Likelihood
None

Location
Comptroller*.sol
CToken.sol

Description

The `mintFresh` function in the `CToken` contract calls the `mintVerify` hook after it is done minting new tokens:

```
        /* We emit a Mint event, and a Transfer event */
        emit Mint(minter, vars.actualMintAmount, vars.mintTokens);
        emit Transfer(address(this), minter, vars.mintTokens);

        /* We call the defense hook */
       comptroller.mintVerify(address(this), minter, vars.actualMintAmount, vars.mintTokens)
```

However, the `mintVerify` function implementations do nothing in the `Comptroller` contracts, for example:

```
    /**
     * @notice Validates mint and reverts on rejection. May emit logs.
     * @param cToken Asset being minted
     * @param minter The address minting the tokens
     * @param actualMintAmount The amount of the underlying asset being minted
     * @param mintTokens The number of tokens being minted
     */
      function mintVerify(address cToken, address minter, uint actualMintAmount, uint
mintTokens) external {
        // Shh - currently unused
        cToken;
        minter;
        actualMintAmount;
        mintTokens;

        // Shh - we don't ever want this hook to be marked pure
        if (false) {
            maxAssets = maxAssets;
        }
    }
```

The hook could be removed if it is not needed.

Recommendations

Remove the hook if it is not needed, or document the fact that the hook is doing nothing in the `CToken` contract.

Status

This finding was fixed in commit d717d9dc277a756b6061fbc24033beb28e119ed6.

The development team explained this is intentional. The hooks are left in case new `CTokens` want to implement it in the future. Documentation was added in source code in order to make this more evident.

RLD-007	Function state mutability can be restricted to pure

Total Risk	Impact	Location
Info	None	ComptrollerG*.sol
Fixed	**Likelihood**	
✘	None	

Description

The Solidity compiler emits a warning indicating the state mutability for the `getCompAddress` functions in all the `Comptroller` contracts should be restricted to pure, instead of view, to indicate the function does not read from the contract state:

```
/**
 * @notice Return the address of the COMP token
 * @return The address of COMP
 */
function getCompAddress() public view returns (address) {
    return 0xc00e94Cb662C3520282E6f5717214004A7f26888;
}
```

Recommendation

Declare the functions as pure instead of view.

RLD-008 Incorrect ecrecover return value checks

Total Risk	Impact	Location
Low	Low	Governance/RLEN.sol Governance/GovernorAlpha.sol
Fixed	Likelihood	
✘	Medium	

Description

In the RSK blockchain implementation, `ecrecover`'s precompiled contract return value in error scenarios is the value `0xdcc703c0E500B653Ca82273B7BFAd8045D85a470` in contrast with Ethereum's implementation which returns the 0 address for errors.

The code inherited from the Compound implementation was not adapted as needed for the RSK network.

The `castVoteBySig` function in the `GovernorAlpha` smart contract checks the return value is not address 0:

```
function castVoteBySig(uint proposalId, bool support, uint8 v, bytes32 r, bytes32 s) public {
    bytes32 domainSeparator = keccak256(abi.encode(DOMAIN_TYPEHASH, keccak256(bytes(name)),
            getChainId(), address(this)));
    bytes32 structHash = keccak256(abi.encode(BALLOT_TYPEHASH, proposalId, support));
    bytes32 digest = keccak256(abi.encodePacked("\x19\x01", domainSeparator, structHash));
    address signatory = ecrecover(digest, v, r, s);
    require(signatory != address(0), "GovernorAlpha::castVoteBySig: invalid signature");
    return _castVote(signatory, proposalId, support);
}
```

Also, in the `delegateBySig` function in the `RLEN` smart contract:

```
function delegateBySig(address delegatee, uint nonce, uint expiry, uint8 v, bytes32 r,
            bytes32 s) public {
    bytes32 domainSeparator = keccak256(abi.encode(DOMAIN_TYPEHASH, keccak256(bytes(name)),
            getChainId(), address(this)));
    bytes32 structHash = keccak256(abi.encode(DELEGATION_TYPEHASH, delegatee, nonce,
            expiry));
    bytes32 digest = keccak256(abi.encodePacked("\x19\x01", domainSeparator, structHash));
    address signatory = ecrecover(digest, v, r, s);
    require(signatory != address(0), "RLEN::delegateBySig: invalid signature");
    require(nonce == nonces[signatory]++, "RLEN::delegateBySig: invalid nonce");
    require(now <= expiry, "RLEN::delegateBySig: signature expired");
    return _delegate(signatory, delegatee);
}
```

As a consequence, it is possible to cast a vote or delegate voting rights using an invalid signature.

Recommendation

Coinspect recommends checking for `ecrecover` RSK specific error return value in order to prevent future mistakes; for an example check RSKAddrValidator.sol.
It is worth noting however, this code which is part of the governance model in Compound, will not be used in production, at least initially, in the rLending platform, as the development team has stated and thus this finding is considered a low risk issue. However, it is worth fixing it in case the code is used in the future.

9. Disclaimer

The information presented in this document is provided as is and without warranty. Vulnerability assessments are a “point in time” analysis and as such it is possible that something in the environment could have changed since the tests reflected in this report were run. This report should not be considered a perfect representation of the risks threatening the analysed system, networks, and applications.