Quantstamp Contract Security Certificate

November 8th 2019 — Quantstamp Verified

RToken Ethereum Contracts

This smart contract audit was prepared by Quantstamp, the protocol for securing smart contracts.

Executive Summary

Type	ERC20 Reedemable Token Audit
Auditors	Sebastian Banescu, Senior Research Engineer Sung-Shine Lee, Research Engineer Kacper Bąk, Senior Research Engineer
Timeline	2019-10-16 through 2019-10-25
EVM	Byzantium
Languages	Solidity, Javascript
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	RToken Ethereum Contracts
Source Code	

Repository	Commit
rtoken-contracts	1d3c5df

Total Issues	9	(5 Resolved)
High Risk Issues	1	(1 Resolved)
Medium Risk Issues	1	(1 Resolved)
Low Risk Issues	2	(0 Resolved)
Informational Risk Issues	2	(1 Resolved)
Undetermined Risk Issues	3	(2 Resolved)

Overall Assessment

Quantstamp has audited the security of the RToken Ethereum Contracts and has identified a number of issues, several of which are of undetermined severity. The quality and security of this project is overall superior to what we commonly see. However, we do recommend that the issues of high down to low severity are addressed before this code is deployed in production. Some of the more concerning issues are the integer overflow and underflow that occur in several locations in the code, as well as the possible transfer to 0x0.

NOTE: For this security audit, Quantstamp has not audited the contracts located in the `compound/` folder. However, we do note that those contracts have been audited by other third parties and all indicated issues have been resolved.

Update: Quantstamp confirms that the most severe issues found during the audit have been fixed in the update of the code that we have reaudited.

Severity Categories	
︽ High	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
^ Medium	The issue puts a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or is reasonably likely to lead to moderate financial impact.
⌄ Low	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low-impact in view of the client's business circumstances.
○ Informational	The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.
- Undetermined	The impact of the issue is uncertain.

Changelog

- 2019-10-25 - Initial report
- 2019-10-31 - Updated based on commit `375cae3`

Quantstamp Audit Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Methodology

The Quantstamp auditing process follows a routine series of steps:

1. Code review that includes the following
   i. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract
   ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
2. Testing and automated analysis that includes the following:
   i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
   ii. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Toolset

The notes below outline the setup and steps performed in the process of this audit.

Setup

Tool Setup:

- Maian
- Truffle
- Ganache
- SolidityCoverage
- Oyente
- Mythril
- Truffle-Flattener
- Securify
- Slither

Steps taken to run the tools:

1. Installed Truffle: `npm install -g truffle`
2. Installed Ganache: `npm install -g ganache-cli`
3. Installed the solidity-coverage tool (within the project's root directory): `npm install --save-dev solidity-coverage`
4. Ran the coverage tool from the project's root directory: `./node_modules/.bin/solidity-coverage`
5. Flattened the source code using `truffle-flattener` to accommodate the auditing tools.
6. Installed the Mythril tool from Pypi: `pip3 install mythril`
7. Ran the Mythril tool on each contract: `myth -x path/to/contract`
8. Ran the Securify tool: `java -Xmx6048m -jar securify-0.1.jar -fs contract.sol`
9. Installed the Oyente tool from Docker: `docker pull luongnguyen/oyente`
10. Migrated files into Oyente (root directory): `docker run -v $(pwd):/tmp - it luongnguyen/oyente`
11. Ran the Oyente tool on each contract: `cd /oyente/oyente && python oyente.py /tmp/path/to/contract`
12. Cloned the MAIAN tool: `git clone --depth 1 https://github.com/MAIAN-tool/MAIAN.git maian`
13. Ran the MAIAN tool on each contract: `cd maian/tool/ && python3 maian.py -s path/to/contract contract.sol`
14. Installed the Slither tool: `pip install slither-analyzer`
15. Run Slither from the project directory `slither .`

Assessment

Findings

Integer Overflow / Underflow

Severity: *High*

Status: Fixed

Contract(s) affected: `RToken.sol`

Related Issue(s): SWC-101

Description: Integer overflow/underflow occur when an integer hits its bit-size limit. Every integer has a set range; when that range is passed, the value loops back around. A clock is a good analogy: at 11:59, the minute hand goes to 0, not 60, because 59 is the largest possible minute.
The following lines of code are potentially affected by integer overflow / underflow:

- `L540: savingAssetOrignalAmount += sOriginalCreated;`
- `L617: totalProportions += uint256(proportions[i]);`
- `L621: (uint256(proportions[i]) * uint256(PROPORTION_BASE)) /`
- `L667: if (rGross > (account.lDebt + account.rInterest)) {`
- `L668: return rGross - account.lDebt - account.rInterest;`
- `L705: (rAmount * hat.proportions[i]) / PROPORTION_BASE;`
- `L717: (sInternalAmount * hat.proportions[i]) / PROPORTION_BASE;`
- `L801: (rAmount * hat.proportions[i]) / PROPORTION_BASE;`
- `L814: (sInternalAmount * hat.proportions[i]) / PROPORTION_BASE;`

Recommendation: We recommend using the `SafeMath` library from OpenZeppelin for performing arithmetic operations.

Possible Transfer to 0x0 / Contract Address

Severity: *Medium*

Status: Fixed

Contract(s) affected: `RToken.sol`

Description: It is rarely desirable for tokens to be sent to the `0x0` address (intentional token burning is a notable exception) nor to the contract itself. However, these mistakes are often made due to human errors. Hence, it's often a good idea to prevent these mistakes from happening within the smart contract itself.
Neither `createHat` nor `createHatInternal` checks if one of the `recipients` addresses are equal to `0x0`. This allows a user to set one of the recipients to `0x0` by mistake, which would effectively lead to transferring tokens to `0x0` when `distributeLoans` is called.

Recommendation: Check that none of the recipients' addresses are `0x0` inside of the `createHatInternal` function.

Unlocked Pragma

Severity: *Low*

Contract(s) affected: `All smart contracts under the contracts/ directory`

Related Issue(s): SWC-103

Description: Every Solidity file specifies in the header a version number of the format `pragma solidity (^)0.5.*`. The caret (^) before the version number implies an unlocked pragma, meaning that the compiler will use the specified version *and above*, hence the term "unlocked." For consistency and to prevent unexpected behavior in the future, it is recommended to remove the caret to lock the file onto a specific Solidity version.
Because the use of `ABIEncoderV2`, the `pragma` should be locked above `0.5.10`, as there is a known bug in array storage: https://blog.ethereum.org/2019/06/25/solidity-storage-array-bugs/
There are currently no signed integer arrays that are initialized in a vulnerable way, hence the low severity of this issue. However, it is best to take precautions for future updates of the code.

Recommendation: For consistency and to prevent unexpected behavior in the future, it is recommended to remove the caret to lock the file onto a specific Solidity version. Furthermore, as a best practice we recommend using the most recent version of Solidity.

Update: The RToken team notified us that locking the pragma to specific version would make it impossible for other projects to integrate.

Allowance Double-Spend Exploit

Severity: *Low*

Contract(s) affected: `RToken.sol`

Related Issue(s): SWC-114

Description: As it presently is constructed, the contract is vulnerable to the allowance double-spend exploit, as with other ERC20 tokens.

Exploit Scenario: An example of an exploit goes as follows:

1. Alice allows Bob to transfer `N` amount of Alice's tokens (`N>0`) by calling the `approve()` method on `Token` smart contract (passing Bob's address and `N` as method arguments)
2. After some time, Alice decides to change from `N` to `M` (`M>0`) the number of Alice's tokens Bob is allowed to transfer, so she calls the `approve()` method again, this time passing Bob's address and `M` as method arguments
3. Bob notices Alice's second transaction before it was mined and quickly sends another transaction that calls the `transferFrom()` method to transfer `N` Alice's tokens somewhere
4. If Bob's transaction will be executed before Alice's transaction, then Bob will successfully transfer `N` Alice's tokens and will gain an ability to transfer another `M` tokens
5. Before Alice notices any irregularities, Bob calls `transferFrom()` method again, this time to transfer `M` Alice's tokens.

Recommendation: The exploit (as described above) is mitigated through use of functions that increase/decrease the allowance relative to its current value, such as `increaseAllowance` and `decreaseAllowance`. However, we did not identify the presence of these functions in the code.
Pending community agreement on an ERC standard that would protect against this exploit, we recommend that developers of applications dependent on `approve()` / `transferFrom()` should keep in mind that they have to set allowance to 0 first and verify if it was used before setting the new value. Teams who decide to wait for such a standard should make these recommendations to app developers who work with their token contract.

Update: The RToken team decided not to fix the issue since they see it as a low severity issue.

User can Lock Oneself Out

Severity: *Informational*

Contract(s) affected: `RToken.sol`

Description: In the functions `distributeLoans` and `recollectLoans` a user may accidentally lock oneself out if the number of `recipients` or the number of `proportions` is too high. Consequently, the for loops on lines 612, 619, 697 and 793 will require too much gas to finish and the transaction (function call) will fail for those accounts.

Recommendation: While given the limit of 50 recipients, this is unlikely to happen, we still recommend performing a gas analysis to find out the length of `recipients` and `proportions` for which gas consumption may be too high. Then, based on that value, the team should decide whether it makes sense to break down the `for`-loop or place a lower limit than 50 on the possible number of recipients and proportions.

Update: RToken acknowledged that this issue is known to them and is planning to update the code accordingly.

Unchecked Return Value

Severity: Informational

Status: Fixed

Contract(s) affected: `RToken.sol`

Description: Most functions will return a `true` or `false` value upon success. Some functions, like `send()`, are more crucial to check than others. It's important to ensure that every necessary function is checked. The following unchecked return values were identified:

- `RToken.changeAllocationStrategy(IAllocationStrategy)` ignores return value by external calls `token.transferFrom(msg.sender,address(this),totalSupply)`
- `RToken.changeAllocationStrategy(IAllocationStrategy)` ignores return value by external calls `token.approve(address(ias),totalSupply)`
- `RToken.mintInternal(uint256)` ignores return value by external calls `token.transferFrom(msg.sender,address(this),mintAmount)`
- `RToken.mintInternal(uint256)` ignores return value by external calls `token.approve(address(ias),mintAmount)`
- `RToken.redeemInternal(address,uint256)` ignores return value by external calls `token.transfer(redeemTo,redeemAmount)`
- `RToken.estimateAndRecollectLoans(address,uint256)` ignores return value by external calls `ias.accrueInterest()`
- `RToken.sameHatTransfer(address,address,uint256)` ignores return value by external calls `ias.accrueInterest()`
- `RToken.payInterestInternal(address)` ignores return value by external calls `ias.accrueInterest()`

Recommendation: We recommend checking the return value and taking a decision to proceed or not based on the return value.

Source Account can be Recipient

Severity: Undetermined

Contract(s) affected: `RToken.sol`

Description: While the code prevents from the obvious inheritance case where the recipient is the original owner (`SELF_HAT_ID`) on `L478`, it seems that the `src` can hand craft a hat that makes him-/her-self a recipient. This way it would be automatically inherited by the recipient.

Recommendation: In case it is undesired that users make themselves hat recipients of the tokens they transfer, then it is recommended to prevent this from happening by adding an additional check. Otherwise, the documentation should reflect this possibility explicitly.

Update: The developers have notified Quantstamp that they do not consider this to be an issue.

Possible Confusion between `savingAssetConversionRate` and `10**18`

Severity: Undetermined

Status: Fixed

Contract(s) affected: `RToken.sol`

Description: On several lines of code (`L342`, `L386`, `L770`), the constant `10**18` is used for certain computations. For example:

```
340: totalSavingsAmount += savingAssetOrignalAmount
341:             .mul(ias.exchangeRateStored())
342:             .div(10**18);
```

Similar computations are being done on other lines of code using `savingAssetConversionRate`instead of the constant `10**18`. For example:

```
368: stats.totalSavings = statsStored.totalInternalSavings
369:             .mul(ias.exchangeRateStored())
370:             .div(savingAssetConversionRate);
```

Since the initial value used to initialize `savingAssetConversionRate` is `10**18`, it is unclear whether the constant `10**18` is being misused or not. A similar finding was observed for the same constant `10**18`and the call to `ias.exchangeRateStored()` (see `L770` versus `L750`).

Recommendation: We recommend reviewing the semantics of the computations performed using the constant `10**18`and replace it with either `savingAssetConversionRate`, `ias.exchangeRateStored()` or a named constant.

Use of Experimental Features in Production

Severity: Undetermined

Status: Fixed

Contract(s) affected: `RToken.sol`, `IRToken.sol`, `RTokenStorage.sol`

Description: Some smart contracts rely on the experimental feature `ABIEncoderV2`, whose use is discouraged in production. Some bugs associated with it have already been reported (see https://blog.ethereum.org/2019/03/26/solidity-optimizer-and-abiencoderv2-bug/ and https://blog.ethereum.org/2019/06/25/solidity-storage-array-bugs/), but have been fixed on release `0.5.10` onwards.

Recommendation: Migrate the contracts to any version higher or equal to `0.5.10`.

Automated Analyses

Maian

Maian was not able to analyze this code base.

Oyente

Oyente reported no issues for this code base.

Mythril

Mythril reported no issues for this code base.

Securify

Security indicated `LockedEther`warnings for all the smart contracts. We classified these issues as false positives.

Slither

Slither has indicated several reentrancy issues for functions that perform calls to the `IAllocationStrategy` instance `ias` and then update state variables. However, we have classified these issues as false positives.

Adherence to Specification

The technical specification of this project is of a good quality. We have identified the following open issues/questions regarding the implementation that we want to signal to the development team:

- `RToken.sol` on line 135, why is it necessary to pay interests to `dst`? Why doesn't it happen in `transfer()`, `transferAll()`, and `transferFromI()` as well? **Update:** FIXED
- The technical specification mentions that the administrator can change the contract's hat, but also says that "contracts can generally change the hat once". However, the administrator can change the contract's hat any number of times. Either the documentation needs to be adjusted to indicate this capability of the administrator or the implementation should be adjusted. **Update:** The team informed us the hat inheritance rule currently is permanently applied to a contract, hence the language was used there "generally be changed once".
- `RToken.sol` on line 506: It is not clear where the rounding error comes from. This is not described in the specification. **Update:** The RToken team acknowledged that the issue would be explained in the future versions of the documentation.
- `RToken.sol` on line 390: The "allocationStrategy" can be changed by the owner of the contract and all the investment is automatically invested into the new allocation strategy. Since it is possible that the "allocationStrategy" is malicious (detrimental for the user), it requires some trust from the user. Moreover, on line 416, the owner can change the had for any contract address for any number of times. **Update:** Response from RToken: The RToken contract is governance agnostic. Although, the official rDAIs will have Aragon DAO App/Agent as admin, centralized largely but with a governance.
- `RToken.sol`: If `accounts[src].hatID != 0`, `accounts[dst].hatID == 0`, and `accounts[src].hatID != SELF_HAT_ID`, then `sameHat` is false on `L473`. In this case, `L479` will be executed, making them logically the "same hat". However, the variable `sameHat` remains false, and will go into the `if`-clause at `L491`. Is this the expected behaviour? **Update:** The RToken team confirmed that the logic is correct and has simplified the logic to prevent future confusion.

Code Documentation

The project is well documented with code comments. However, some functions do not describe their parameters or return values. It is recommended that each function describe at least its parameters and return value.

Adherence to Best Practices

We recommend addressing the following best practice issues:

- Functions `recollectLoans` (L780) and `distributeLoans` (L684) in `RToken.sol` share very similar code and are very long and complex functions, which makes them hard to maintain. It is recommended to extract common code into functions and call the new functions in each of the 2 aforementioned functions. **Update:** The RToken Team decided not to fix as it is hard to reuse the code.
- Call/reuse `getHatByID()`inside of the `getHatByAddress` function on line 261 of `RToken.sol`, in order to avoid code clones. **Update:** FIXED
- Similarly to the previous issue:
  - `transferAll()` could reuse `transfer()`on line 125.
  - `transferAllFrom()` should probably reuse `transferFrom()`
  - `minWithNewHat()` could reuse `minWithSelectedHat()`
  - `redeemAll()` could reuse `redeem()`
  - `redeemAndTranferAll()` could reuse `redeemAndTransfer()`
  - **Update:** the RToken team updated the code so that the aforementioned functions use `transferInternal()` / `mintInternal()` / `redeemInternal()` instead.

- Explicitly declare the value as a named constant. Avoid using unnamed constants such as:
  - `10**18`on lines 47, 342, 386, 410, 545 and 770 of `RToken.sol`. **Update:** FIXED
  - `uint256(int256(-1))` on line 55; reuse `SELF_HAT_ID` defined on line 28 instead. **Update:** FIXED
  - `uint256(-1)` on lines 454, 486; note that `uint256(-1)`is equal to `uint256(int256(-1))`. **Update:** FIXED

- The calls to `payInterestInternal(src)` on lines 124 and 134 are unnecessary, because the subsequently called `transferInternal` function also calls `payInterestInternal(src)`with the same parameter value. **Update:** The team informed us that this design was intentional in order to pay all interest at once and drain the address balance to 0. However, as an improvement they removed `payInterest()` from `transferInternal()`.
- `RToken.sol L548-549`: `Mint` event is not standard in ERC20. The way the events are currently emitted might confuse some tracker that follows OpenZeppelin. To be specific, there is no `Mint` event in ERC20, usually people emit a `Transfer` event when mint happens. However, the `Transfer` event is from `0x0` to the minter address. L549 would also be confusing to a tracker that only looks at the events, as there are no tokens given to the contract, and the contract sends token to the `msg.sender`. Suggestion: change the event emissions of `Mint` and `Transfer` to just `Transfer(0x0, msg.sender, mintAmount);` **Update:** FIXED
- `RToken.sol L588-589`: Similarly for the `redeemInternal` method, the burn should become `Transfer(msg.sender, 0x0, redeemAmount)` **Update:** FIXED
- `RToken.sol L878`: It seems like this method mints tokens, therefore, it is recommended to follow the same convention described previously and make the `Transfer` event from `0x0` to the minter address. **Update:** FIXED
- `RToken.sol L892`: The 4th parameter of the `LoansTransferred` event is hard-coded to `true`. Since this is the single instance where this event is used in this code base, it can be removed. **Update:** FIXED
- `RToken L41`: To avoid confusion regarding where the `initialize()` function is defined, we recommend using `super.initialize()`. **Update:** FIXED
- The statement on lines 645 and 648 can be brought before the `if`-testament on `L640` and the `else`-clause could be then removed. **Update:** FIXED
- `CompoundAllocationStrategy.sol L38, L51`: There are outstanding TODO comments, which we recommend be addressed or removed. **Update:** The team informed us that they are leaving this issue as unresolved for now; they are using `revert()` / `require()` as a remedy.
- `CompoundAllocationStrategy.sol L42 - 44, L55-57`: The `else` branch is commented out and indicates that this case is undesired. If there control flow should never enter the `else` branch, then use `require` or `assert` and the `if`-clause would no longer be necessary. **Update:** FIXED

Test Results

Test Suite Results

All tests in the provided test suite are passing.

```
  Contract: CToken
admin is 0x79f2FCc50EC9f1C5Ef5CB0B6b0BE976AA87c0F17
bingeBorrower is 0x6922725d089F2280741715BDB0c0861e37E5ca3d
customer1 is 0xBE6424C8C3D871733EA1Ea41d7deCEd7CFe0b6BF
ERC20Mintable.new: started
ERC20Mintable.new: done, gas used 1498981, gas price 0.000000001 Gwei
token.mint 1000 -> customer1: started
token.mint 1000 -> customer1: done, gas used 66909, gas price 0.000000001 Gwei
ComptrollerMock.new: started
ComptrollerMock.new: done, gas used 4956159, gas price 0.000000001 Gwei
InterestRateModelMock.new: started
InterestRateModelMock.new: done, gas used 304892, gas price 0.000000001 Gwei
```

```
CErc20.new: started
CErc20.new: done, gas used 9116329, gas price 0.000000001 Gwei
token.approve 100 by customer1: started
token.approve 100 by customer1: done, gas used 45741, gas price 0.000000001 Gwei
cToken.mint 100 to customer1: started
cToken.mint 100 to customer1: done, gas used 144026, gas price 0.000000001 Gwei
cToken.redeem 100: started
cToken.redeem 100: done, gas used 107476, gas price 0.000000001 Gwei
Before binge borrowing: 1 cToken = 0.10000 Token
cToken.borrow 10 to bingeBorrower: started
cToken.borrow 10 to bingeBorrower: done, gas used 165160, gas price 0.000000001 Gwei
Wait for 100 blocks...
cToken.accrueInterest: started
cToken.accrueInterest: done, gas used 55499, gas price 0.000000001 Gwei
After binge borrowing: 1 cToken = 0.10000 Token
cToken.redeem 100: started
cToken.redeem 100: done, gas used 107476, gas price 0.000000001 Gwei
token redeemed 920.00011
cToken.redeemUnderlying 10: started
cToken.redeemUnderlying 10: done, gas used 108111, gas price 0.000000001 Gwei
     ✓ #1 cToken basic operations (6089ms)

  Contract: RToken
admin is 0x79f2FCc50EC9f1C5Ef5CB0B6b0BE976AA87c0F17
bingeBorrower is 0x6922725d089F2280741715BDB0c0861e37E5ca3d
customer1 is 0xBE6424C8C3D871733EA1Ea41d7deCEd7CFe0b6BF
customer2 is 0x05055D4B2190f4ABBAA5Ef8376F8A4c0BeA8A85E
customer3 is 0x7e7FdDFf2e91E9D049e892927f7e67DC25e4f11b
ERC20Mintable.new: started
ERC20Mintable.new: done, gas used 1498981, gas price 0.000000001 Gwei
token.mint 1000 -> customer1: started
token.mint 1000 -> customer1: done, gas used 66909, gas price 0.000000001 Gwei
ComptrollerMock.new: started
ComptrollerMock.new: done, gas used 4956159, gas price 0.000000001 Gwei
InterestRateModelMock.new: started
InterestRateModelMock.new: done, gas used 304892, gas price 0.000000001 Gwei
CErc20.new: started
CErc20.new: done, gas used 9116329, gas price 0.000000001 Gwei
CompoundAllocationStrategy.new: started
CompoundAllocationStrategy.new: done, gas used 2067660, gas price 0.000000001 Gwei
RToken.new: started
RToken.new: done, gas used 18913795, gas price 0.000000001 Gwei
Proxy.new: started
Proxy.new: done, gas used 451298, gas price 0.000000001 Gwei
compoundAS.transferOwnership: started
compoundAS.transferOwnership: done, gas used 30444, gas price 0.000000001 Gwei
     ✓ #0 initial test condition (109ms)
ERC20Mintable.new: started
ERC20Mintable.new: done, gas used 1498981, gas price 0.000000001 Gwei
token.mint 1000 -> customer1: started
token.mint 1000 -> customer1: done, gas used 66909, gas price 0.000000001 Gwei
ComptrollerMock.new: started
ComptrollerMock.new: done, gas used 4956159, gas price 0.000000001 Gwei
InterestRateModelMock.new: started
InterestRateModelMock.new: done, gas used 304892, gas price 0.000000001 Gwei
CErc20.new: started
CErc20.new: done, gas used 9116329, gas price 0.000000001 Gwei
CompoundAllocationStrategy.new: started
CompoundAllocationStrategy.new: done, gas used 2067660, gas price 0.000000001 Gwei
RToken.new: started
RToken.new: done, gas used 18913795, gas price 0.000000001 Gwei
Proxy.new: started
Proxy.new: done, gas used 451298, gas price 0.000000001 Gwei
compoundAS.transferOwnership: started
compoundAS.transferOwnership: done, gas used 30444, gas price 0.000000001 Gwei
token.approve 100 by customer1: started
token.approve 100 by customer1: done, gas used 45741, gas price 0.000000001 Gwei
rToken.mint 100 to customer1: started
rToken.mint 100 to customer1: done, gas used 446203, gas price 0.000000001 Gwei
customer1 tokenBalance 100.00000 expected 100.00000
customer1 receivedLoan 100.00000 expected 100.00000
customer1 receivedSavings 100.00000 expected 100.00000
customer1 interestPayable 0.00000 expected 0.00000
customer1 cumulativeInterest 0.00000 expected 0.00000
Before binge borrowing: 1 cToken = 0.10000 Token
cToken.borrow 10 to bingeBorrower: started
cToken.borrow 10 to bingeBorrower: done, gas used 165160, gas price 0.000000001 Gwei
Wait for 100 blocks...
cToken.accrueInterest: started
cToken.accrueInterest: done, gas used 55499, gas price 0.000000001 Gwei
After binge borrowing: 1 cToken = 0.10000 Token
customer1 tokenBalance 100.00000 expected 100.00000
customer1 receivedLoan 100.00000 expected 100.00000
customer1 receivedSavings 100.00100 expected 100.00100
customer1 interestPayable 0.00100 expected 0.00100
customer1 cumulativeInterest 0.00000 expected 0.00000
rToken.redeem 10 to customer1: started
rToken.redeem 10 to customer1: done, gas used 324103, gas price 0.000000001 Gwei
customer1 tokenBalance 90.00101 expected 90.00101
customer1 receivedLoan 90.00000 expected 90.00000
customer1 receivedSavings 90.00101 expected 90.00101
customer1 interestPayable 0.00000 expected 0.00000
customer1 cumulativeInterest 0.00101 expected 0.00101
rToken.payInterest to customer1: started
rToken.payInterest to customer1: done, gas used 106802, gas price 0.000000001 Gwei
customer1 tokenBalance 90.00102 expected 90.00102
customer1 receivedLoan 90.00000 expected 90.00000
customer1 receivedSavings 90.00102 expected 90.00102
customer1 interestPayable 0.00000 expected 0.00000
customer1 cumulativeInterest 0.00102 expected 0.00102
rToken.payInterest to customer1 again: started
rToken.payInterest to customer1 again: done, gas used 106802, gas price 0.000000001 Gwei
customer1 tokenBalance 90.00103 expected 90.00103
```

```
customer1 receivedLoan 90.00000 expected 90.00000
customer1 receivedSavings 90.00103 expected 90.00103
customer1 interestPayable 0.00000 expected 0.00000
customer1 cumulativeInterest 0.00103 expected 0.00103
rToken.redeem 2 of customer1 to customer3: started
rToken.redeem 2 of customer1 to customer3: done, gas used 310822, gas price 0.000000001 Gwei
    ✓ #2 rToken normal operations with zero hatter (15974ms)
ERC20Mintable.new: started
ERC20Mintable.new: done, gas used 1498981, gas price 0.000000001 Gwei
token.mint 1000 -> customer1: started
token.mint 1000 -> customer1: done, gas used 66909, gas price 0.000000001 Gwei
ComptrollerMock.new: started
ComptrollerMock.new: done, gas used 4956159, gas price 0.000000001 Gwei
InterestRateModelMock.new: started
InterestRateModelMock.new: done, gas used 304892, gas price 0.000000001 Gwei
CErc20.new: started
CErc20.new: done, gas used 9116329, gas price 0.000000001 Gwei
CompoundAllocationStrategy.new: started
CompoundAllocationStrategy.new: done, gas used 2067660, gas price 0.000000001 Gwei
RToken.new: started
RToken.new: done, gas used 18913795, gas price 0.000000001 Gwei
Proxy.new: started
Proxy.new: done, gas used 451298, gas price 0.000000001 Gwei
compoundAS.transferOwnership: started
compoundAS.transferOwnership: done, gas used 30444, gas price 0.000000001 Gwei
token.approve 100 by customer1: started
token.approve 100 by customer1: done, gas used 45741, gas price 0.000000001 Gwei
rToken.mint 100 to customer1 with a hat benefiting admin(90%) and customer2(10%): started
rToken.mint 100 to customer1 with a hat benefiting admin(90%) and customer2(10%): done, gas used 762773, gas
price 0.000000001 Gwei
customer1 tokenBalance 100.00000 expected 100.00000
customer1 receivedLoan 0.00000 expected 0.00000
customer1 receivedSavings 0.00000 expected 0.00000
customer1 interestPayable 0.00000 expected 0.00000
customer1 cumulativeInterest 0.00000 expected 0.00000
admin tokenBalance 0.00000 expected 0.00000
admin receivedLoan 90.00000 expected 90.00000
admin receivedSavings 90.00000 expected 90.00000
admin interestPayable 0.00000 expected 0.00000
admin cumulativeInterest 0.00000 expected 0.00000
customer2 tokenBalance 0.00000 expected 0.00000
customer2 receivedLoan 10.00000 expected 10.00000
customer2 receivedSavings 10.00000 expected 10.00000
customer2 interestPayable 0.00000 expected 0.00000
customer2 cumulativeInterest 0.00000 expected 0.00000
Before binge borrowing: 1 cToken = 0.10000 Token
cToken.borrow 10 to bingeBorrower: started
cToken.borrow 10 to bingeBorrower: done, gas used 165160, gas price 0.000000001 Gwei
Wait for 100 blocks...
cToken.accrueInterest: started
cToken.accrueInterest: done, gas used 55499, gas price 0.000000001 Gwei
After binge borrowing: 1 cToken = 0.10000 Token
customer1 tokenBalance 100.00000 expected 100.00000
customer1 receivedLoan 0.00000 expected 0.00000
customer1 receivedSavings 0.00000 expected 0.00000
customer1 interestPayable 0.00000 expected 0.00000
customer1 cumulativeInterest 0.00000 expected 0.00000
admin tokenBalance 0.00000 expected 0.00000
admin receivedLoan 90.00000 expected 90.00000
admin receivedSavings 90.00090 expected 90.00090
admin interestPayable 0.00090 expected 0.00090
admin cumulativeInterest 0.00000 expected 0.00000
customer2 tokenBalance 0.00000 expected 0.00000
customer2 receivedLoan 10.00000 expected 10.00000
customer2 receivedSavings 10.00010 expected 10.00010
customer2 interestPayable 0.00010 expected 0.00010
customer2 cumulativeInterest 0.00000 expected 0.00000
rToken.redeem 10 to customer1: started
rToken.redeem 10 to customer1: done, gas used 322371, gas price 0.000000001 Gwei
customer1 tokenBalance 90.00000 expected 90.00000
customer1 receivedLoan 0.00000 expected 0.00000
customer1 receivedSavings 0.00000 expected 0.00000
customer1 interestPayable 0.00000 expected 0.00000
customer1 cumulativeInterest 0.00000 expected 0.00000
admin tokenBalance 0.00000 expected 0.00000
admin receivedLoan 81.00000 expected 81.00000
admin receivedSavings 81.00091 expected 81.00091
admin interestPayable 0.00091 expected 0.00091
admin cumulativeInterest 0.00000 expected 0.00000
customer2 tokenBalance 0.00000 expected 0.00000
customer2 receivedLoan 9.00000 expected 9.00000
customer2 receivedSavings 9.00010 expected 9.00010
customer2 interestPayable 0.00010 expected 0.00010
customer2 cumulativeInterest 0.00000 expected 0.00000
rToken.transfer 10 customer1 -> customer3: started
rToken.transfer 10 customer1 -> customer3: done, gas used 459706, gas price 0.000000001 Gwei
customer1 tokenBalance 80.00000 expected 80.00000
customer1 receivedLoan 0.00000 expected 0.00000
customer1 receivedSavings 0.00000 expected 0.00000
customer1 interestPayable 0.00000 expected 0.00000
customer1 cumulativeInterest 0.00000 expected 0.00000
admin tokenBalance 0.00000 expected 0.00000
admin receivedLoan 81.00000 expected 81.00000
admin receivedSavings 81.00092 expected 81.00092
admin interestPayable 0.00092 expected 0.00092
admin cumulativeInterest 0.00000 expected 0.00000
customer2 tokenBalance 0.00000 expected 0.00000
customer2 receivedLoan 9.00000 expected 9.00000
customer2 receivedSavings 9.00010 expected 9.00010
customer2 interestPayable 0.00010 expected 0.00010
customer2 cumulativeInterest 0.00000 expected 0.00000
customer3 tokenBalance 10.00000 expected 10.00000
customer3 receivedLoan 0.00000 expected 0.00000
```

```
customer3 receivedSavings 0.00000 expected 0.00000
customer3 interestPayable 0.00000 expected 0.00000
customer3 cumulativeInterest 0.00000 expected 0.00000
rToken.payInterest to admin: started
rToken.payInterest to admin: done, gas used 151802, gas price 0.000000001 Gwei
customer1 tokenBalance 80.00000 expected 80.00000
customer1 receivedLoan 0.00000 expected 0.00000
customer1 receivedSavings 0.00000 expected 0.00000
customer1 interestPayable 0.00000 expected 0.00000
customer1 cumulativeInterest 0.00000 expected 0.00000
admin tokenBalance 0.00093 expected 0.00093
admin receivedLoan 81.00000 expected 81.00000
admin receivedSavings 81.00093 expected 81.00093
admin interestPayable 0.00000 expected 0.00000
admin cumulativeInterest 0.00093 expected 0.00093
customer2 tokenBalance 0.00000 expected 0.00000
customer2 receivedLoan 9.00000 expected 9.00000
customer2 receivedSavings 9.00010 expected 9.00010
customer2 interestPayable 0.00010 expected 0.00010
customer2 cumulativeInterest 0.00000 expected 0.00000
customer3 tokenBalance 10.00000 expected 10.00000
customer3 receivedLoan 0.00000 expected 0.00000
customer3 receivedSavings 0.00000 expected 0.00000
customer3 interestPayable 0.00000 expected 0.00000
customer3 cumulativeInterest 0.00000 expected 0.00000
Wait for 100 blocks...
cToken.accrueInterest: started
cToken.accrueInterest: done, gas used 55499, gas price 0.000000001 Gwei
customer1 tokenBalance 80.00000 expected 80.00000
customer1 receivedLoan 0.00000 expected 0.00000
customer1 receivedSavings 0.00000 expected 0.00000
customer1 interestPayable 0.00000 expected 0.00000
customer1 cumulativeInterest 0.00000 expected 0.00000
admin tokenBalance 0.00093 expected 0.00093
admin receivedLoan 81.00000 expected 81.00000
admin receivedSavings 81.00183 expected 81.00183
admin interestPayable 0.00090 expected 0.00090
admin cumulativeInterest 0.00093 expected 0.00093
customer2 tokenBalance 0.00000 expected 0.00000
customer2 receivedLoan 9.00000 expected 9.00000
customer2 receivedSavings 9.00020 expected 9.00020
customer2 interestPayable 0.00020 expected 0.00020
customer2 cumulativeInterest 0.00000 expected 0.00000
customer3 tokenBalance 10.00000 expected 10.00000
customer3 receivedLoan 0.00000 expected 0.00000
customer3 receivedSavings 0.00000 expected 0.00000
customer3 interestPayable 0.00000 expected 0.00000
customer3 cumulativeInterest 0.00000 expected 0.00000
     ✓ #3 rToken normal operations with hat (21595ms)
ERC20Mintable.new: started
ERC20Mintable.new: done, gas used 1498981, gas price 0.000000001 Gwei
token.mint 1000 -> customer1: started
token.mint 1000 -> customer1: done, gas used 66909, gas price 0.000000001 Gwei
ComptrollerMock.new: started
ComptrollerMock.new: done, gas used 4956159, gas price 0.000000001 Gwei
InterestRateModelMock.new: started
InterestRateModelMock.new: done, gas used 304892, gas price 0.000000001 Gwei
CErc20.new: started
CErc20.new: done, gas used 9116329, gas price 0.000000001 Gwei
CompoundAllocationStrategy.new: started
CompoundAllocationStrategy.new: done, gas used 2067660, gas price 0.000000001 Gwei
RToken.new: started
RToken.new: done, gas used 18913795, gas price 0.000000001 Gwei
Proxy.new: started
Proxy.new: done, gas used 451298, gas price 0.000000001 Gwei
compoundAS.transferOwnership: started
compoundAS.transferOwnership: done, gas used 30444, gas price 0.000000001 Gwei
token.approve 100 by customer1: started
token.approve 100 by customer1: done, gas used 45741, gas price 0.000000001 Gwei
rToken.mint 10 to customer1: started
rToken.mint 10 to customer1: done, gas used 461139, gas price 0.000000001 Gwei
customer1 tokenBalance 10.00000 expected 10.00000
customer1 receivedLoan 10.00000 expected 10.00000
customer1 receivedSavings 10.00000 expected 10.00000
customer1 interestPayable 0.00000 expected 0.00000
customer1 cumulativeInterest 0.00000 expected 0.00000
rToken.mint 5 to customer1: started
rToken.mint 5 to customer1: done, gas used 315477, gas price 0.000000001 Gwei
customer1 tokenBalance 15.00000 expected 15.00000
customer1 receivedLoan 15.00000 expected 15.00000
customer1 receivedSavings 15.00000 expected 15.00000
customer1 interestPayable 0.00000 expected 0.00000
customer1 cumulativeInterest 0.00000 expected 0.00000
Before binge borrowing: 1 cToken = 0.10000 Token
cToken.borrow 10 to bingeBorrower: started
cToken.borrow 10 to bingeBorrower: done, gas used 165160, gas price 0.000000001 Gwei
Wait for 100 blocks...
cToken.accrueInterest: started
cToken.accrueInterest: done, gas used 55489, gas price 0.000000001 Gwei
After binge borrowing: 1 cToken = 0.10007 Token
customer1 tokenBalance 15.00000 expected 15.00000
customer1 receivedLoan 15.00000 expected 15.00000
customer1 receivedSavings 15.01000 expected 15.01000
customer1 interestPayable 0.01000 expected 0.01000
customer1 cumulativeInterest 0.00000 expected 0.00000
     ✓ #4 rToken mint multiple times (6692ms)
ERC20Mintable.new: started
ERC20Mintable.new: done, gas used 1498981, gas price 0.000000001 Gwei
token.mint 1000 -> customer1: started
token.mint 1000 -> customer1: done, gas used 66909, gas price 0.000000001 Gwei
ComptrollerMock.new: started
ComptrollerMock.new: done, gas used 4956159, gas price 0.000000001 Gwei
InterestRateModelMock.new: started
InterestRateModelMock.new: done, gas used 304892, gas price 0.000000001 Gwei
CErc20.new: started
CErc20.new: done, gas used 9116329, gas price 0.000000001 Gwei
CompoundAllocationStrategy.new: started
CompoundAllocationStrategy.new: done, gas used 2067660, gas price 0.000000001 Gwei
RToken.new: started
RToken.new: done, gas used 18913795, gas price 0.000000001 Gwei
Proxy.new: started
Proxy.new: done, gas used 451298, gas price 0.000000001 Gwei
compoundAS.transferOwnership: started
compoundAS.transferOwnership: done, gas used 30444, gas price 0.000000001 Gwei
token.approve 100 by customer1: started
token.approve 100 by customer1: done, gas used 45741, gas price 0.000000001 Gwei
rToken.mint 200 to customer1 with a hat benefiting customer1(10%) and customer2(90%): started
rToken.mint 200 to customer1 with a hat benefiting customer1(10%) and customer2(90%): done, gas used 762832, gas
price 0.000000001 Gwei
rToken.transfer 190 customer1 -> customer2: started
rToken.transfer 190 customer1 -> customer2: done, gas used 459888, gas price 0.000000001 Gwei
customer1 tokenBalance 100.00000 expected 100.00000
customer1 receivedLoan 20.00000 expected 20.00000
customer1 receivedSavings 20.00000 expected 20.00000
customer1 interestPayable 0.00000 expected 0.00000
customer1 cumulativeInterest 0.00000 expected 0.00000
customer2 tokenBalance 100.00000 expected 100.00000
customer2 receivedLoan 180.00000 expected 180.00000
customer2 receivedSavings 180.00000 expected 180.00000
customer2 interestPayable 0.00000 expected 0.00000
customer2 cumulativeInterest 0.00000 expected 0.00000
Before binge borrowing: 1 cToken = 0.10000 Token
cToken.borrow 10 to bingeBorrower: started
cToken.borrow 10 to bingeBorrower: done, gas used 165160, gas price 0.000000001 Gwei
Wait for 100 blocks...
cToken.accrueInterest: started
cToken.accrueInterest: done, gas used 55499, gas price 0.000000001 Gwei
After binge borrowing: 1 cToken = 0.10000 Token
customer1 tokenBalance 100.00000 expected 100.00000
customer1 receivedLoan 20.00000 expected 20.00000
customer1 receivedSavings 20.00010 expected 20.00010
customer1 interestPayable 0.00010 expected 0.00010
customer1 cumulativeInterest 0.00000 expected 0.00000
customer2 tokenBalance 100.00000 expected 100.00000
customer2 receivedLoan 180.00000 expected 180.00000
customer2 receivedSavings 180.00090 expected 180.00090
customer2 interestPayable 0.00090 expected 0.00090
customer2 cumulativeInterest 0.00000 expected 0.00000
rToken.payInterest to customer2: started
rToken.payInterest to customer2: done, gas used 136802, gas price 0.000000001 Gwei
customer1 tokenBalance 100.00000 expected 100.00000
customer1 receivedLoan 20.00000 expected 20.00000
customer1 receivedSavings 20.00010 expected 20.00010
customer1 interestPayable 0.00010 expected 0.00010
customer1 cumulativeInterest 0.00000 expected 0.00000
customer2 tokenBalance 100.00091 expected 100.00091
customer2 receivedLoan 180.00000 expected 180.00000
customer2 receivedSavings 180.00091 expected 180.00091
customer2 interestPayable 0.00000 expected 0.00000
customer2 cumulativeInterest 0.00091 expected 0.00091
rToken.redeem maximum to customer2: started
rToken.redeem maximum to customer2: done, gas used 338646, gas price 0.000000001 Gwei
     ✓ #5 rToken redeem all including paid interests (14288ms)
ERC20Mintable.new: started
ERC20Mintable.new: done, gas used 1498981, gas price 0.000000001 Gwei
token.mint 1000 -> customer1: started
token.mint 1000 -> customer1: done, gas used 66909, gas price 0.000000001 Gwei
ComptrollerMock.new: started
ComptrollerMock.new: done, gas used 4956159, gas price 0.000000001 Gwei
InterestRateModelMock.new: started
InterestRateModelMock.new: done, gas used 304892, gas price 0.000000001 Gwei
CErc20.new: started
CErc20.new: done, gas used 9116329, gas price 0.000000001 Gwei
CompoundAllocationStrategy.new: started
CompoundAllocationStrategy.new: done, gas used 2067660, gas price 0.000000001 Gwei
RToken.new: started
RToken.new: done, gas used 18913795, gas price 0.000000001 Gwei
Proxy.new: started
Proxy.new: done, gas used 451298, gas price 0.000000001 Gwei
compoundAS.transferOwnership: started
compoundAS.transferOwnership: done, gas used 30444, gas price 0.000000001 Gwei
rToken.createHat for customer1 benefiting admin and customer3 10/90: started
rToken.createHat for customer1 benefiting admin and customer3 10/90: done, gas used 233016, gas price
0.000000001 Gwei
rToken.createHat for customer2 benefiting admin and customer4 20/80: started
rToken.createHat for customer2 benefiting admin and customer4 20/80: done, gas used 233016, gas price
0.000000001 Gwei
token.approve 100 by customer1: started
token.approve 100 by customer1: done, gas used 45741, gas price 0.000000001 Gwei
rToken.mint 100 to customer1: started
rToken.mint 100 to customer1: done, gas used 559228, gas price 0.000000001 Gwei
admin tokenBalance 0.00000 expected 0.00000
admin receivedLoan 20.00000 expected 20.00000
admin receivedSavings 20.00000 expected 20.00000
admin interestPayable 0.00000 expected 0.00000
admin cumulativeInterest 0.00000 expected 0.00000
customer1 tokenBalance 200.00000 expected 200.00000
customer1 receivedLoan 0.00000 expected 0.00000
customer1 receivedSavings 0.00000 expected 0.00000
customer1 interestPayable 0.00000 expected 0.00000
customer1 cumulativeInterest 0.00000 expected 0.00000
customer2 tokenBalance 0.00000 expected 0.00000
customer2 receivedLoan 0.00000 expected 0.00000
customer2 receivedSavings 0.00000 expected 0.00000
customer2 interestPayable 0.00000 expected 0.00000
customer2 cumulativeInterest 0.00000 expected 0.00000
```

```
customer3 tokenBalance 0.00000 expected 0.00000
customer3 receivedLoan 180.00000 expected 180.00000
customer3 receivedSavings 180.00000 expected 180.00000
customer3 interestPayable 0.00000 expected 0.00000
customer3 cumulativeInterest 0.00000 expected 0.00000
customer4 tokenBalance 0.00000 expected 0.00000
customer4 receivedLoan 0.00000 expected 0.00000
customer4 receivedSavings 0.00000 expected 0.00000
customer4 interestPayable 0.00000 expected 0.00000
customer4 cumulativeInterest 0.00000 expected 0.00000
rToken.transfer 100 from customer1 to customer 2: started
rToken.transfer 100 from customer1 to customer 2: done, gas used 484799, gas price 0.000000001 Gwei
admin tokenBalance 0.00000 expected 0.00000
admin receivedLoan 30.00000 expected 30.00000
admin receivedSavings 30.00000 expected 30.00000
admin interestPayable 0.00000 expected 0.00000
admin cumulativeInterest 0.00000 expected 0.00000
customer1 tokenBalance 100.00000 expected 100.00000
customer1 receivedLoan 0.00000 expected 0.00000
customer1 receivedSavings 0.00000 expected 0.00000
customer1 interestPayable 0.00000 expected 0.00000
customer1 cumulativeInterest 0.00000 expected 0.00000
customer2 tokenBalance 100.00000 expected 100.00000
customer2 receivedLoan 0.00000 expected 0.00000
customer2 receivedSavings 0.00000 expected 0.00000
customer2 interestPayable 0.00000 expected 0.00000
customer2 cumulativeInterest 0.00000 expected 0.00000
customer3 tokenBalance 0.00000 expected 0.00000
customer3 receivedLoan 90.00000 expected 90.00000
customer3 receivedSavings 90.00000 expected 90.00000
customer3 interestPayable 0.00000 expected 0.00000
customer3 cumulativeInterest 0.00000 expected 0.00000
customer4 tokenBalance 0.00000 expected 0.00000
customer4 receivedLoan 80.00000 expected 80.00000
customer4 receivedSavings 80.00000 expected 80.00000
customer4 interestPayable 0.00000 expected 0.00000
customer4 cumulativeInterest 0.00000 expected 0.00000
Before binge borrowing: 1 cToken = 0.10000 Token
cToken.borrow 10 to bingeBorrower: started
cToken.borrow 10 to bingeBorrower: done, gas used 165160, gas price 0.000000001 Gwei
Wait for 100 blocks...
cToken.accrueInterest: started
cToken.accrueInterest: done, gas used 55499, gas price 0.000000001 Gwei
After binge borrowing: 1 cToken = 0.10000 Token
admin tokenBalance 0.00000 expected 0.00000
admin receivedLoan 30.00000 expected 30.00000
admin receivedSavings 30.00015 expected 30.00015
admin interestPayable 0.00015 expected 0.00015
admin cumulativeInterest 0.00000 expected 0.00000
customer1 tokenBalance 100.00000 expected 100.00000
customer1 receivedLoan 0.00000 expected 0.00000
customer1 receivedSavings 0.00000 expected 0.00000
customer1 interestPayable 0.00000 expected 0.00000
customer1 cumulativeInterest 0.00000 expected 0.00000
customer2 tokenBalance 100.00000 expected 100.00000
customer2 receivedLoan 0.00000 expected 0.00000
customer2 receivedSavings 0.00000 expected 0.00000
customer2 interestPayable 0.00000 expected 0.00000
customer2 cumulativeInterest 0.00000 expected 0.00000
customer3 tokenBalance 0.00000 expected 0.00000
customer3 receivedLoan 90.00000 expected 90.00000
customer3 receivedSavings 90.00045 expected 90.00045
customer3 interestPayable 0.00045 expected 0.00045
customer3 cumulativeInterest 0.00000 expected 0.00000
customer4 tokenBalance 0.00000 expected 0.00000
customer4 receivedLoan 80.00000 expected 80.00000
customer4 receivedSavings 80.00040 expected 80.00040
customer4 interestPayable 0.00040 expected 0.00040
customer4 cumulativeInterest 0.00000 expected 0.00000
     ✓ #6 transfer and switch hats (14077ms)
ERC20Mintable.new: started
ERC20Mintable.new: done, gas used 1498981, gas price 0.000000001 Gwei
token.mint 1000 -> customer1: started
token.mint 1000 -> customer1: done, gas used 66909, gas price 0.000000001 Gwei
ComptrollerMock.new: started
ComptrollerMock.new: done, gas used 4956159, gas price 0.000000001 Gwei
InterestRateModelMock.new: started
InterestRateModelMock.new: done, gas used 304892, gas price 0.000000001 Gwei
CErc20.new: started
CErc20.new: done, gas used 9116329, gas price 0.000000001 Gwei
CompoundAllocationStrategy.new: started
CompoundAllocationStrategy.new: done, gas used 2067660, gas price 0.000000001 Gwei
RToken.new: started
RToken.new: done, gas used 18913795, gas price 0.000000001 Gwei
Proxy.new: started
Proxy.new: done, gas used 451298, gas price 0.000000001 Gwei
compoundAS.transferOwnership: started
compoundAS.transferOwnership: done, gas used 30444, gas price 0.000000001 Gwei
token.transfer 100 from customer 1 to customer 2: started
token.transfer 100 from customer 1 to customer 2: done, gas used 51616, gas price 0.000000001 Gwei
token.approve 100 by customer2: started
token.approve 100 by customer2: done, gas used 45741, gas price 0.000000001 Gwei
rToken.mint 100 to customer2: started
rToken.mint 100 to customer2: done, gas used 431203, gas price 0.000000001 Gwei
token.approve 100 by customer1: started
token.approve 100 by customer1: done, gas used 45741, gas price 0.000000001 Gwei
rToken.mint 100 to customer1: started
rToken.mint 100 to customer1: done, gas used 345541, gas price 0.000000001 Gwei
Before binge borrowing: 1 cToken = 0.10000 Token
cToken.borrow 10 to bingeBorrower: started
cToken.borrow 10 to bingeBorrower: done, gas used 165160, gas price 0.000000001 Gwei
Wait for 100 blocks...
cToken.accrueInterest: started
cToken.accrueInterest: done, gas used 55499, gas price 0.000000001 Gwei
After binge borrowing: 1 cToken = 0.10000 Token
rToken.payInterest to customer1: started
rToken.payInterest to customer1: done, gas used 136802, gas price 0.000000001 Gwei
customer1 tokenBalance 100.00051 expected 100.00051
customer1 receivedLoan 100.00000 expected 100.00000
customer1 receivedSavings 100.00051 expected 100.00051
customer1 interestPayable 0.00000 expected 0.00000
customer1 cumulativeInterest 0.00051 expected 0.00051
rToken.redeem all to customer1: started
rToken.redeem all to customer1: done, gas used 284541, gas price 0.000000001 Gwei
customer1 tokenBalance 0.0000 expected 0.0000
customer1 receivedLoan 0.0000 expected 0.0000
customer1 receivedSavings 0.0000 expected 0.0000
customer1 interestPayable 0.0000 expected 0.0000
customer1 cumulativeInterest 0.0005 expected 0.0005
token.approve 100 by customer1: started
token.approve 100 by customer1: done, gas used 45741, gas price 0.000000001 Gwei
rToken.mint 100 to customer1: started
rToken.mint 100 to customer1: done, gas used 341251, gas price 0.000000001 Gwei
Wait for 100 blocks...
cToken.accrueInterest: started
cToken.accrueInterest: done, gas used 55499, gas price 0.000000001 Gwei
rToken.payInterest to customer1: started
rToken.payInterest to customer1: done, gas used 106802, gas price 0.000000001 Gwei
customer1 tokenBalance 100.00051 expected 100.00051
customer1 receivedLoan 100.00000 expected 100.00000
customer1 receivedSavings 100.00051 expected 100.00051
customer1 interestPayable 0.00000 expected 0.00000
customer1 cumulativeInterest 0.00102 expected 0.00102
rToken.transfer all from customer1 to customer2: started
rToken.transfer all from customer1 to customer2: done, gas used 341702, gas price 0.000000001 Gwei
customer1 tokenBalance 0.0000 expected 0.0000
customer1 receivedLoan 0.0000 expected 0.0000
customer1 receivedSavings 0.0000 expected 0.0000
customer1 interestPayable 0.0000 expected 0.0000
customer1 cumulativeInterest 0.0010 expected 0.0010
token.approve 100 by customer1: started
token.approve 100 by customer1: done, gas used 45741, gas price 0.000000001 Gwei
rToken.mint 100 to customer1: started
rToken.mint 100 to customer1: done, gas used 341251, gas price 0.000000001 Gwei
Wait for 100 blocks...
cToken.accrueInterest: started
cToken.accrueInterest: done, gas used 55499, gas price 0.000000001 Gwei
rToken.payInterest to customer1: started
rToken.payInterest to customer1: done, gas used 106802, gas price 0.000000001 Gwei
customer1 tokenBalance 100.0003 expected 100.0003
customer1 receivedLoan 100.0000 expected 100.0000
customer1 receivedSavings 100.0003 expected 100.0003
customer1 interestPayable 0.0000 expected 0.0000
customer1 cumulativeInterest 0.0014 expected 0.0014
     ✓ #7 rToken redeem all including paid interest for zero hatter (17867ms)
ERC20Mintable.new: started
ERC20Mintable.new: done, gas used 1498981, gas price 0.000000001 Gwei
token.mint 1000 -> customer1: started
token.mint 1000 -> customer1: done, gas used 66909, gas price 0.000000001 Gwei
ComptrollerMock.new: started
ComptrollerMock.new: done, gas used 4956159, gas price 0.000000001 Gwei
InterestRateModelMock.new: started
InterestRateModelMock.new: done, gas used 304892, gas price 0.000000001 Gwei
CErc20.new: started
CErc20.new: done, gas used 9116329, gas price 0.000000001 Gwei
CompoundAllocationStrategy.new: started
CompoundAllocationStrategy.new: done, gas used 2067660, gas price 0.000000001 Gwei
RToken.new: started
RToken.new: done, gas used 18913795, gas price 0.000000001 Gwei
Proxy.new: started
Proxy.new: done, gas used 451298, gas price 0.000000001 Gwei
compoundAS.transferOwnership: started
compoundAS.transferOwnership: done, gas used 30444, gas price 0.000000001 Gwei
     ✓ #8 special hats (89ms)
ERC20Mintable.new: started
ERC20Mintable.new: done, gas used 1498981, gas price 0.000000001 Gwei
token.mint 1000 -> customer1: started
token.mint 1000 -> customer1: done, gas used 66909, gas price 0.000000001 Gwei
ComptrollerMock.new: started
ComptrollerMock.new: done, gas used 4956159, gas price 0.000000001 Gwei
InterestRateModelMock.new: started
InterestRateModelMock.new: done, gas used 304892, gas price 0.000000001 Gwei
CErc20.new: started
CErc20.new: done, gas used 9116265, gas price 0.000000001 Gwei
CompoundAllocationStrategy.new: started
CompoundAllocationStrategy.new: done, gas used 2067660, gas price 0.000000001 Gwei
RToken.new: started
RToken.new: done, gas used 18913795, gas price 0.000000001 Gwei
Proxy.new: started
Proxy.new: done, gas used 451298, gas price 0.000000001 Gwei
compoundAS.transferOwnership: started
compoundAS.transferOwnership: done, gas used 30444, gas price 0.000000001 Gwei
token.approve 100 by customer1: started
token.approve 100 by customer1: done, gas used 45741, gas price 0.000000001 Gwei
rToken.mintWithSelectedHat 100 to customer1 with the self hat: started
rToken.mintWithSelectedHat 100 to customer1 with the self hat: done, gas used 497542, gas price 0.000000001 Gwei
customer1 tokenBalance 100.00000 expected 100.00000
customer1 receivedLoan 100.00000 expected 100.00000
customer1 receivedSavings 100.00000 expected 100.00000
customer1 interestPayable 0.00000 expected 0.00000
customer1 cumulativeInterest 0.00000 expected 0.00000
rToken.transfer all from customer1 to customer2: started
rToken.transfer all from customer1 to customer2: done, gas used 347890, gas price 0.000000001 Gwei
customer1 tokenBalance 80.00000 expected 80.00000
customer1 receivedLoan 80.00000 expected 80.00000
customer1 receivedSavings 80.00000 expected 80.00000
```

```
customer1 interestPayable 0.00000 expected 0.00000
customer1 cumulativeInterest 0.00000 expected 0.00000
customer2 tokenBalance 20.00000 expected 20.00000
customer2 receivedLoan 20.00000 expected 20.00000
customer2 receivedSavings 20.00000 expected 20.00000
customer2 interestPayable 0.00000 expected 0.00000
customer2 cumulativeInterest 0.00000 expected 0.00000
rToken changeHat for customer3 with selfhat: started
rToken changeHat for customer3 with selfhat: done, gas used 115909, gas price 0.000000001 Gwei
rToken.transfer all from customer1 to customer3: started
rToken.transfer all from customer1 to customer3: done, gas used 314971, gas price 0.000000001 Gwei
customer1 tokenBalance 60.00000 expected 60.00000
customer1 receivedLoan 60.00000 expected 60.00000
customer1 receivedSavings 60.00000 expected 60.00000
customer1 interestPayable 0.00000 expected 0.00000
customer1 cumulativeInterest 0.00000 expected 0.00000
customer3 tokenBalance 20.00000 expected 20.00000
customer3 receivedLoan 20.00000 expected 20.00000
customer3 receivedSavings 20.00000 expected 20.00000
customer3 interestPayable 0.00000 expected 0.00000
customer3 cumulativeInterest 0.00000 expected 0.00000
Before binge borrowing: 1 cToken = 0.10000 Token
cToken.borrow 10 to bingeBorrower: started
cToken.borrow 10 to bingeBorrower: done, gas used 165160, gas price 0.000000001 Gwei
Wait for 100 blocks...
cToken.accrueInterest: started
cToken.accrueInterest: done, gas used 55499, gas price 0.000000001 Gwei
After binge borrowing: 1 cToken = 0.10000 Token
customer1 tokenBalance 60.00000 expected 60.00000
customer1 receivedLoan 60.00000 expected 60.00000
customer1 receivedSavings 60.00060 expected 60.00060
customer1 interestPayable 0.00060 expected 0.00060
customer1 cumulativeInterest 0.00000 expected 0.00000
customer2 tokenBalance 20.00000 expected 20.00000
customer2 receivedLoan 20.00000 expected 20.00000
customer2 receivedSavings 20.00020 expected 20.00020
customer2 interestPayable 0.00020 expected 0.00020
customer2 cumulativeInterest 0.00000 expected 0.00000
customer3 tokenBalance 20.00000 expected 20.00000
customer3 receivedLoan 20.00000 expected 20.00000
customer3 receivedSavings 20.00020 expected 20.00020
customer3 interestPayable 0.00020 expected 0.00020
customer3 cumulativeInterest 0.00000 expected 0.00000
     ✓ #9 rToken operations with self hatter (10161ms)
ERC20Mintable.new: started
ERC20Mintable.new: done, gas used 1498981, gas price 0.000000001 Gwei
token.mint 1000 -> customer1: started
token.mint 1000 -> customer1: done, gas used 66909, gas price 0.000000001 Gwei
ComptrollerMock.new: started
ComptrollerMock.new: done, gas used 4956159, gas price 0.000000001 Gwei
InterestRateModelMock.new: started
InterestRateModelMock.new: done, gas used 304892, gas price 0.000000001 Gwei
CErc20.new: started
CErc20.new: done, gas used 9116329, gas price 0.000000001 Gwei
CompoundAllocationStrategy.new: started
CompoundAllocationStrategy.new: done, gas used 2067660, gas price 0.000000001 Gwei
RToken.new: started
RToken.new: done, gas used 18913795, gas price 0.000000001 Gwei
Proxy.new: started
Proxy.new: done, gas used 451298, gas price 0.000000001 Gwei
compoundAS.transferOwnership: started
compoundAS.transferOwnership: done, gas used 30380, gas price 0.000000001 Gwei
token.approve 100 by customer1: started
token.approve 100 by customer1: done, gas used 45677, gas price 0.000000001 Gwei
rToken.mintWithSelectedHat 100 to customer1 with the self hat: started
rToken.mintWithSelectedHat 100 to customer1 with the self hat: done, gas used 497542, gas price 0.000000001 Gwei
redeemUnderlying by admin: started
     ✓ #10 CompoundAs ownership protection (1616ms)
ERC20Mintable.new: started
ERC20Mintable.new: done, gas used 1498981, gas price 0.000000001 Gwei
token.mint 1000 -> customer1: started
token.mint 1000 -> customer1: done, gas used 66909, gas price 0.000000001 Gwei
ComptrollerMock.new: started
ComptrollerMock.new: done, gas used 4956159, gas price 0.000000001 Gwei
InterestRateModelMock.new: started
InterestRateModelMock.new: done, gas used 304892, gas price 0.000000001 Gwei
CErc20.new: started
CErc20.new: done, gas used 9116329, gas price 0.000000001 Gwei
CompoundAllocationStrategy.new: started
CompoundAllocationStrategy.new: done, gas used 2067660, gas price 0.000000001 Gwei
RToken.new: started
RToken.new: done, gas used 18913795, gas price 0.000000001 Gwei
Proxy.new: started
Proxy.new: done, gas used 451298, gas price 0.000000001 Gwei
compoundAS.transferOwnership: started
compoundAS.transferOwnership: done, gas used 30444, gas price 0.000000001 Gwei
token.approve 100 by customer1: started
token.approve 100 by customer1: done, gas used 45741, gas price 0.000000001 Gwei
rToken.mintWithSelectedHat 100 to customer1 with the self hat: started
rToken.mintWithSelectedHat 100 to customer1 with the self hat: done, gas used 497542, gas price 0.000000001 Gwei
Before binge borrowing: 1 cToken = 0.10000 Token
cToken.borrow 10 to bingeBorrower: started
cToken.borrow 10 to bingeBorrower: done, gas used 165160, gas price 0.000000001 Gwei
Wait for 100 blocks...
cToken.accrueInterest: started
cToken.accrueInterest: done, gas used 55499, gas price 0.000000001 Gwei
After binge borrowing: 1 cToken = 0.10000 Token
rToken.transferAll from customer1 to customer2: started
rToken.transferAll from customer1 to customer2: done, gas used 318616, gas price 0.000000001 Gwei
customer1 tokenBalance 0.00000 expected 0.00000
customer1 receivedLoan 0.00000 expected 0.00000
customer1 receivedSavings 0.00000 expected 0.00000
customer1 interestPayable 0.00000 expected 0.00000
```

```
customer1 cumulativeInterest 0.00101 expected 0.00101
customer2 tokenBalance 100.00101 expected 100.00101
customer2 receivedLoan 100.00101 expected 100.00101
customer2 receivedSavings 100.00101 expected 100.00101
customer2 interestPayable 0.00000 expected 0.00000
customer2 cumulativeInterest 0.00000 expected 0.00000
     ✓ #11 transferAll (6243ms)
ERC20Mintable.new: started
ERC20Mintable.new: done, gas used 1498981, gas price 0.000000001 Gwei
token.mint 1000 -> customer1: started
token.mint 1000 -> customer1: done, gas used 66909, gas price 0.000000001 Gwei
ComptrollerMock.new: started
ComptrollerMock.new: done, gas used 4956159, gas price 0.000000001 Gwei
InterestRateModelMock.new: started
InterestRateModelMock.new: done, gas used 304892, gas price 0.000000001 Gwei
CErc20.new: started
CErc20.new: done, gas used 9116265, gas price 0.000000001 Gwei
CompoundAllocationStrategy.new: started
CompoundAllocationStrategy.new: done, gas used 2067660, gas price 0.000000001 Gwei
RToken.new: started
RToken.new: done, gas used 18913795, gas price 0.000000001 Gwei
Proxy.new: started
Proxy.new: done, gas used 451234, gas price 0.000000001 Gwei
compoundAS.transferOwnership: started
compoundAS.transferOwnership: done, gas used 30444, gas price 0.000000001 Gwei
token.approve 100 by customer1: started
token.approve 100 by customer1: done, gas used 45741, gas price 0.000000001 Gwei
rToken.mintWithSelectedHat 100 to customer1 with the self hat: started
rToken.mintWithSelectedHat 100 to customer1 with the self hat: done, gas used 497542, gas price 0.000000001 Gwei
token.transfer 100 from customer 1 to customer 2: started
token.transfer 100 from customer 1 to customer 2: done, gas used 51616, gas price 0.000000001 Gwei
token.approve 100 by customer2: started
token.approve 100 by customer2: done, gas used 45741, gas price 0.000000001 Gwei
rToken.mintWithSelectedHat 100 to customer2 with the self hat: started
rToken.mintWithSelectedHat 100 to customer2 with the self hat: done, gas used 366880, gas price 0.000000001 Gwei
Before binge borrowing: 1 cToken = 0.10000 Token
cToken.borrow 10 to bingeBorrower: started
cToken.borrow 10 to bingeBorrower: done, gas used 165160, gas price 0.000000001 Gwei
Wait for 100 blocks...
cToken.accrueInterest: started
cToken.accrueInterest: done, gas used 55499, gas price 0.000000001 Gwei
After binge borrowing: 1 cToken = 0.10000 Token
rToken.redeemAll for customer1: started
rToken.redeemAll for customer1: done, gas used 268822, gas price 0.000000001 Gwei
customer1 tokenBalance 0.00000 expected 0.00000
customer1 receivedLoan 0.00000 expected 0.00000
customer1 receivedSavings 0.00000 expected 0.00000
customer1 interestPayable 0.00000 expected 0.00000
customer1 cumulativeInterest 0.00051 expected 0.00051
     ✓ #12 redeemAll (7541ms)
ERC20Mintable.new: started
ERC20Mintable.new: done, gas used 1498981, gas price 0.000000001 Gwei
token.mint 1000 -> customer1: started
token.mint 1000 -> customer1: done, gas used 66909, gas price 0.000000001 Gwei
ComptrollerMock.new: started
ComptrollerMock.new: done, gas used 4956159, gas price 0.000000001 Gwei
InterestRateModelMock.new: started
InterestRateModelMock.new: done, gas used 304892, gas price 0.000000001 Gwei
CErc20.new: started
CErc20.new: done, gas used 9116329, gas price 0.000000001 Gwei
CompoundAllocationStrategy.new: started
CompoundAllocationStrategy.new: done, gas used 2067660, gas price 0.000000001 Gwei
RToken.new: started
RToken.new: done, gas used 18913795, gas price 0.000000001 Gwei
Proxy.new: started
Proxy.new: done, gas used 451298, gas price 0.000000001 Gwei
compoundAS.transferOwnership: started
compoundAS.transferOwnership: done, gas used 30444, gas price 0.000000001 Gwei
token.approve 100 by customer1: started
token.approve 100 by customer1: done, gas used 45741, gas price 0.000000001 Gwei
rToken.mintWithSelectedHat 100 to customer1 with the self hat: started
rToken.mintWithSelectedHat 100 to customer1 with the self hat: done, gas used 497542, gas price 0.000000001 Gwei
token.approve customer 2 by customer1: started
token.approve customer 2 by customer1: done, gas used 47788, gas price 0.000000001 Gwei
rToken transferFrom customer1 -> customer3 by customer2 more than approved: started
rToken transferFrom customer1 -> customer3 by customer2 all approved: started
rToken transferFrom customer1 -> customer3 by customer2 all approved: done, gas used 340041, gas price
0.000000001 Gwei
token.approve customer 2 by customer1: started
token.approve customer 2 by customer1: done, gas used 47852, gas price 0.000000001 Gwei
Before binge borrowing: 1 cToken = 0.10000 Token
cToken.borrow 10 to bingeBorrower: started
cToken.borrow 10 to bingeBorrower: done, gas used 165160, gas price 0.000000001 Gwei
Wait for 100 blocks...
cToken.accrueInterest: started
cToken.accrueInterest: done, gas used 55499, gas price 0.000000001 Gwei
After binge borrowing: 1 cToken = 0.10000 Token
rToken transferAllFrom customer1 -> customer3 by customer2: started
rToken transferAllFrom customer1 -> customer3 by customer2: done, gas used 306499, gas price 0.000000001 Gwei
customer1 tokenBalance 0.00000 expected 0.00000
customer1 receivedLoan 0.00000 expected 0.00000
customer1 receivedSavings 0.00000 expected 0.00000
customer1 interestPayable 0.00000 expected 0.00000
customer1 cumulativeInterest 0.00051 expected 0.00051
customer3 tokenBalance 100.00101 expected 100.00101
customer3 receivedLoan 100.00051 expected 100.00051
customer3 receivedSavings 100.00101 expected 100.00101
customer3 interestPayable 0.00000 expected 0.00000
customer3 cumulativeInterest 0.00051 expected 0.00051
     ✓ #13 approve & transferFrom & transferAllFrom (10455ms)
ERC20Mintable.new: started
ERC20Mintable.new: done, gas used 1498981, gas price 0.000000001 Gwei
token.mint 1000 -> customer1: started
token.mint 1000 -> customer1: done, gas used 66909, gas price 0.000000001 Gwei
ComptrollerMock.new: started
ComptrollerMock.new: done, gas used 4956159, gas price 0.000000001 Gwei
InterestRateModelMock.new: started
InterestRateModelMock.new: done, gas used 304892, gas price 0.000000001 Gwei
CErc20.new: started
CErc20.new: done, gas used 9116329, gas price 0.000000001 Gwei
CompoundAllocationStrategy.new: started
CompoundAllocationStrategy.new: done, gas used 2067660, gas price 0.000000001 Gwei
RToken.new: started
RToken.new: done, gas used 18913795, gas price 0.000000001 Gwei
Proxy.new: started
Proxy.new: done, gas used 451234, gas price 0.000000001 Gwei
compoundAS.transferOwnership: started
compoundAS.transferOwnership: done, gas used 30444, gas price 0.000000001 Gwei
token.approve 100 by customer1: started
token.approve 100 by customer1: done, gas used 45741, gas price 0.000000001 Gwei
rToken.mintWithSelectedHat 100 to customer1 with the self hat: started
rToken.mintWithSelectedHat 100 to customer1 with the self hat: done, gas used 497542, gas price 0.000000001 Gwei
token.transfer 100 from customer 1 to customer 2: started
token.transfer 100 from customer 1 to customer 2: done, gas used 51616, gas price 0.000000001 Gwei
token.approve 100 by customer2: started
token.approve 100 by customer2: done, gas used 45741, gas price 0.000000001 Gwei
rToken.mintWithSelectedHat 100 to customer2 with the self hat: started
rToken.mintWithSelectedHat 100 to customer2 with the self hat: done, gas used 366880, gas price 0.000000001 Gwei
Before binge borrowing: 1 cToken = 0.10000 Token
cToken.borrow 10 to bingeBorrower: started
cToken.borrow 10 to bingeBorrower: done, gas used 165160, gas price 0.000000001 Gwei
Wait for 100 blocks...
cToken.accrueInterest: started
cToken.accrueInterest: done, gas used 55499, gas price 0.000000001 Gwei
After binge borrowing: 1 cToken = 0.10000 Token
rToken.redeemAndTransferAll for customer1: started
rToken.redeemAndTransferAll for customer1: done, gas used 285483, gas price 0.000000001 Gwei
customer1 tokenBalance 0.00000 expected 0.00000
customer1 receivedLoan 0.00000 expected 0.00000
customer1 receivedSavings 0.00000 expected 0.00000
customer1 interestPayable 0.00000 expected 0.00000
customer1 cumulativeInterest 0.00051 expected 0.00051
    ✓ #14 redeemAndTransferAll (7367ms)
ERC20Mintable.new: started
ERC20Mintable.new: done, gas used 1498981, gas price 0.000000001 Gwei
token.mint 1000 -> customer1: started
token.mint 1000 -> customer1: done, gas used 66909, gas price 0.000000001 Gwei
ComptrollerMock.new: started
ComptrollerMock.new: done, gas used 4956159, gas price 0.000000001 Gwei
InterestRateModelMock.new: started
InterestRateModelMock.new: done, gas used 304892, gas price 0.000000001 Gwei
CErc20.new: started
CErc20.new: done, gas used 9116329, gas price 0.000000001 Gwei
CompoundAllocationStrategy.new: started
CompoundAllocationStrategy.new: done, gas used 2067660, gas price 0.000000001 Gwei
RToken.new: started
RToken.new: done, gas used 18913795, gas price 0.000000001 Gwei
Proxy.new: started
Proxy.new: done, gas used 451298, gas price 0.000000001 Gwei
compoundAS.transferOwnership: started
compoundAS.transferOwnership: done, gas used 30444, gas price 0.000000001 Gwei
RToken.new: started
RToken.new: done, gas used 18913795, gas price 0.000000001 Gwei
rToken.updateCode: started
rToken.updateCode: done, gas used 33315, gas price 0.000000001 Gwei
token.approve 100 by customer1: started
token.approve 100 by customer1: done, gas used 45741, gas price 0.000000001 Gwei
rToken.mint 100 to customer1: started
rToken.mint 100 to customer1: done, gas used 446203, gas price 0.000000001 Gwei
customer1 tokenBalance 100.00000 expected 100.00000
customer1 receivedLoan 100.00000 expected 100.00000
customer1 receivedSavings 100.00000 expected 100.00000
customer1 interestPayable 0.00000 expected 0.00000
customer1 cumulativeInterest 0.00000 expected 0.00000
Before binge borrowing: 1 cToken = 0.10000 Token
cToken.borrow 10 to bingeBorrower: started
cToken.borrow 10 to bingeBorrower: done, gas used 165160, gas price 0.000000001 Gwei
Wait for 100 blocks...
cToken.accrueInterest: started
cToken.accrueInterest: done, gas used 55499, gas price 0.000000001 Gwei
After binge borrowing: 1 cToken = 0.10000 Token
customer1 tokenBalance 100.00000 expected 100.00000
customer1 receivedLoan 100.00000 expected 100.00000
customer1 receivedSavings 100.00100 expected 100.00100
customer1 interestPayable 0.00100 expected 0.00100
customer1 cumulativeInterest 0.00000 expected 0.00000
rToken.redeem 10 to customer1: started
rToken.redeem 10 to customer1: done, gas used 324103, gas price 0.000000001 Gwei
customer1 tokenBalance 90.00101 expected 90.00101
customer1 receivedLoan 90.00000 expected 90.00000
customer1 receivedSavings 90.00101 expected 90.00101
customer1 interestPayable 0.00000 expected 0.00000
customer1 cumulativeInterest 0.00101 expected 0.00101
rToken.payInterest to customer1: started
rToken.payInterest to customer1: done, gas used 106802, gas price 0.000000001 Gwei
customer1 tokenBalance 90.00102 expected 90.00102
customer1 receivedLoan 90.00000 expected 90.00000
customer1 receivedSavings 90.00102 expected 90.00102
customer1 interestPayable 0.00000 expected 0.00000
customer1 cumulativeInterest 0.00102 expected 0.00102
rToken.payInterest to customer1 again: started
rToken.payInterest to customer1 again: done, gas used 106802, gas price 0.000000001 Gwei
customer1 tokenBalance 90.00103 expected 90.00103
customer1 receivedLoan 90.00000 expected 90.00000
customer1 receivedSavings 90.00103 expected 90.00103
customer1 interestPayable 0.00000 expected 0.00000
```

```
customer1 cumulativeInterest 0.00103 expected 0.00103
rToken.redeem 2 of customer1 to customer3: started
rToken.redeem 2 of customer1 to customer3: done, gas used 310822, gas price 0.000000001 Gwei
    ✓ #15 upgrade contract (13260ms)
ERC20Mintable.new: started
ERC20Mintable.new: done, gas used 1498981, gas price 0.000000001 Gwei
token.mint 1000 -> customer1: started
token.mint 1000 -> customer1: done, gas used 66909, gas price 0.000000001 Gwei
ComptrollerMock.new: started
ComptrollerMock.new: done, gas used 4956159, gas price 0.000000001 Gwei
InterestRateModelMock.new: started
InterestRateModelMock.new: done, gas used 304892, gas price 0.000000001 Gwei
CErc20.new: started
CErc20.new: done, gas used 9116329, gas price 0.000000001 Gwei
CompoundAllocationStrategy.new: started
CompoundAllocationStrategy.new: done, gas used 2067660, gas price 0.000000001 Gwei
RToken.new: started
RToken.new: done, gas used 18913795, gas price 0.000000001 Gwei
Proxy.new: started
Proxy.new: done, gas used 451298, gas price 0.000000001 Gwei
compoundAS.transferOwnership: started
compoundAS.transferOwnership: done, gas used 30380, gas price 0.000000001 Gwei
rTokenLogic.initialize first time: started
rTokenLogic.initialize first time: done, gas used 236294, gas price 0.000000001 Gwei
rTokenLogic.initialize second time: started
rTokenLogic.updateCode from non-owner: started
rToken.initialize (original): started
rToken.updateCode (original): started
RToken.new: started
RToken.new: done, gas used 18913795, gas price 0.000000001 Gwei
rToken.updateCode: started
rToken.updateCode: done, gas used 33315, gas price 0.000000001 Gwei
rTokenLogic.initialize first time: started
rTokenLogic.initialize first time: done, gas used 236294, gas price 0.000000001 Gwei
rTokenLogic.initialize second time: started
rTokenLogic.updateCode from non-owner: started
rToken.initialize (original): started
rToken.updateCode (original): started
    ✓ #16 proxy security (1567ms)
ERC20Mintable.new: started
ERC20Mintable.new: done, gas used 1498981, gas price 0.000000001 Gwei
token.mint 1000 -> customer1: started
token.mint 1000 -> customer1: done, gas used 66909, gas price 0.000000001 Gwei
ComptrollerMock.new: started
ComptrollerMock.new: done, gas used 4956159, gas price 0.000000001 Gwei
InterestRateModelMock.new: started
InterestRateModelMock.new: done, gas used 304892, gas price 0.000000001 Gwei
CErc20.new: started
CErc20.new: done, gas used 9116329, gas price 0.000000001 Gwei
CompoundAllocationStrategy.new: started
CompoundAllocationStrategy.new: done, gas used 2067660, gas price 0.000000001 Gwei
RToken.new: started
RToken.new: done, gas used 18913795, gas price 0.000000001 Gwei
Proxy.new: started
Proxy.new: done, gas used 451298, gas price 0.000000001 Gwei
compoundAS.transferOwnership: started
compoundAS.transferOwnership: done, gas used 30444, gas price 0.000000001 Gwei
token.approve 100 by customer1: started
token.approve 100 by customer1: done, gas used 45741, gas price 0.000000001 Gwei
rToken.mint 100 to customer1: started
rToken.mint 100 to customer1: done, gas used 446203, gas price 0.000000001 Gwei
customer1 tokenBalance 100.00000 expected 100.00000
customer1 receivedLoan 100.00000 expected 100.00000
customer1 receivedSavings 100.00000 expected 100.00000
customer1 interestPayable 0.00000 expected 0.00000
customer1 cumulativeInterest 0.00000 expected 0.00000
Before binge borrowing: 1 cToken = 0.10000 Token
cToken.borrow 10 to bingeBorrower: started
cToken.borrow 10 to bingeBorrower: done, gas used 165160, gas price 0.000000001 Gwei
Wait for 100 blocks...
cToken.accrueInterest: started
cToken.accrueInterest: done, gas used 55499, gas price 0.000000001 Gwei
After binge borrowing: 1 cToken = 0.10000 Token
customer1 tokenBalance 100.00000 expected 100.00000
customer1 receivedLoan 100.00000 expected 100.00000
customer1 receivedSavings 100.00100 expected 100.00100
customer1 interestPayable 0.00100 expected 0.00100
customer1 cumulativeInterest 0.00000 expected 0.00000
rToken.redeem 10 to customer1: started
rToken.redeem 10 to customer1: done, gas used 324103, gas price 0.000000001 Gwei
customer1 tokenBalance 90.00101 expected 90.00101
customer1 receivedLoan 90.00000 expected 90.00000
customer1 receivedSavings 90.00101 expected 90.00101
customer1 interestPayable 0.00000 expected 0.00000
customer1 cumulativeInterest 0.00101 expected 0.00101
rToken.payInterest to customer1: started
rToken.payInterest to customer1: done, gas used 106802, gas price 0.000000001 Gwei
customer1 tokenBalance 90.00102 expected 90.00102
customer1 receivedLoan 90.00000 expected 90.00000
customer1 receivedSavings 90.00102 expected 90.00102
customer1 interestPayable 0.00000 expected 0.00000
customer1 cumulativeInterest 0.00102 expected 0.00102
RToken.new: started
RToken.new: done, gas used 18913795, gas price 0.000000001 Gwei
rToken.updateCode: started
rToken.updateCode: done, gas used 33315, gas price 0.000000001 Gwei
rToken.payInterest to customer1 again: started
rToken.payInterest to customer1 again: done, gas used 106802, gas price 0.000000001 Gwei
customer1 tokenBalance 90.00105 expected 90.00105
customer1 receivedLoan 90.00000 expected 90.00000
customer1 receivedSavings 90.00105 expected 90.00105
customer1 interestPayable 0.00000 expected 0.00000
customer1 cumulativeInterest 0.00105 expected 0.00105
```

```
rToken.redeem 2 of customer1 to customer3: started
rToken.redeem 2 of customer1 to customer3: done, gas used 310822, gas price 0.000000001 Gwei
     ✓ #17 storage continuity during upgrade (10169ms)
ERC20Mintable.new: started
ERC20Mintable.new: done, gas used 1498981, gas price 0.000000001 Gwei
token.mint 1000 -> customer1: started
token.mint 1000 -> customer1: done, gas used 66909, gas price 0.000000001 Gwei
ComptrollerMock.new: started
ComptrollerMock.new: done, gas used 4956159, gas price 0.000000001 Gwei
InterestRateModelMock.new: started
InterestRateModelMock.new: done, gas used 304892, gas price 0.000000001 Gwei
CErc20.new: started
CErc20.new: done, gas used 9116329, gas price 0.000000001 Gwei
CompoundAllocationStrategy.new: started
CompoundAllocationStrategy.new: done, gas used 2067660, gas price 0.000000001 Gwei
RToken.new: started
RToken.new: done, gas used 18913795, gas price 0.000000001 Gwei
Proxy.new: started
Proxy.new: done, gas used 451298, gas price 0.000000001 Gwei
compoundAS.transferOwnership: started
compoundAS.transferOwnership: done, gas used 30444, gas price 0.000000001 Gwei
token.approve 100 by customer1: started
token.approve 100 by customer1: done, gas used 45741, gas price 0.000000001 Gwei
rToken.mint 100 to customer1 with a hat benefiting admin(90%) and customer2(10%): started
rToken.mint 100 to customer1 with a hat benefiting admin(90%) and customer2(10%): done, gas used 762773, gas
price 0.000000001 Gwei
rToken.changeHatFor by customer1: started
rToken.changeHatFor by customer1: started
rToken.changeHatFor by customer1: done, gas used 60907, gas price 0.000000001 Gwei
rToken.changeHatFor by customer1: started
     ✓ #18 admin.changeHatFor (2709ms)
ERC20Mintable.new: started
ERC20Mintable.new: done, gas used 1498981, gas price 0.000000001 Gwei
token.mint 1000 -> customer1: started
token.mint 1000 -> customer1: done, gas used 66909, gas price 0.000000001 Gwei
ComptrollerMock.new: started
ComptrollerMock.new: done, gas used 4956159, gas price 0.000000001 Gwei
InterestRateModelMock.new: started
InterestRateModelMock.new: done, gas used 304892, gas price 0.000000001 Gwei
CErc20.new: started
CErc20.new: done, gas used 9116329, gas price 0.000000001 Gwei
CompoundAllocationStrategy.new: started
CompoundAllocationStrategy.new: done, gas used 2067660, gas price 0.000000001 Gwei
RToken.new: started
RToken.new: done, gas used 18913795, gas price 0.000000001 Gwei
Proxy.new: started
Proxy.new: done, gas used 451298, gas price 0.000000001 Gwei
compoundAS.transferOwnership: started
compoundAS.transferOwnership: done, gas used 30444, gas price 0.000000001 Gwei
token.approve 100 by customer1: started
token.approve 100 by customer1: done, gas used 45741, gas price 0.000000001 Gwei
rToken.mint 100 to customer1 with a sombreror: started
rToken.mint 100 to customer1 with a sombreror: done, gas used 6239427, gas price 0.000000001 Gwei
rToken.transfer 10 customer1 -> customer2: started
rToken.transfer 10 customer1 -> customer2: done, gas used 5447338, gas price 0.000000001 Gwei
rToken.transfer 10 customer1 -> customer2 again: started
rToken.transfer 10 customer1 -> customer2 again: done, gas used 299856, gas price 0.000000001 Gwei
Same hat transfer tx cost 299856
rToken.createHat by bigger sombrero: started
token.transfer 100 from customer 1 to customer 2: started
token.transfer 100 from customer 1 to customer 2: done, gas used 51616, gas price 0.000000001 Gwei
token.approve 100 by customer3: started
token.approve 100 by customer3: done, gas used 45741, gas price 0.000000001 Gwei
rToken.mint 100 to customer3 with a smaller sombrero: started
rToken.mint 100 to customer3 with a smaller sombrero: done, gas used 4803765, gas price 0.000000001 Gwei
     ✓ #19 Max hat numbers & same hat optimization (39028ms)
ERC20Mintable.new: started
ERC20Mintable.new: done, gas used 1498981, gas price 0.000000001 Gwei
token.mint 1000 -> customer1: started
token.mint 1000 -> customer1: done, gas used 66909, gas price 0.000000001 Gwei
ComptrollerMock.new: started
ComptrollerMock.new: done, gas used 4956159, gas price 0.000000001 Gwei
InterestRateModelMock.new: started
InterestRateModelMock.new: done, gas used 304892, gas price 0.000000001 Gwei
CErc20.new: started
CErc20.new: done, gas used 9116329, gas price 0.000000001 Gwei
CompoundAllocationStrategy.new: started
CompoundAllocationStrategy.new: done, gas used 2067660, gas price 0.000000001 Gwei
RToken.new: started
RToken.new: done, gas used 18913795, gas price 0.000000001 Gwei
Proxy.new: started
Proxy.new: done, gas used 451298, gas price 0.000000001 Gwei
compoundAS.transferOwnership: started
compoundAS.transferOwnership: done, gas used 30444, gas price 0.000000001 Gwei
     ✓ #20 Change hat with invalid hat ID should fail (95ms)
ERC20Mintable.new: started
ERC20Mintable.new: done, gas used 1498981, gas price 0.000000001 Gwei
token.mint 1000 -> customer1: started
token.mint 1000 -> customer1: done, gas used 66909, gas price 0.000000001 Gwei
ComptrollerMock.new: started
ComptrollerMock.new: done, gas used 4956159, gas price 0.000000001 Gwei
InterestRateModelMock.new: started
InterestRateModelMock.new: done, gas used 304892, gas price 0.000000001 Gwei
CErc20.new: started
CErc20.new: done, gas used 9116265, gas price 0.000000001 Gwei
CompoundAllocationStrategy.new: started
CompoundAllocationStrategy.new: done, gas used 2067660, gas price 0.000000001 Gwei
RToken.new: started
RToken.new: done, gas used 18913795, gas price 0.000000001 Gwei
Proxy.new: started
Proxy.new: done, gas used 451298, gas price 0.000000001 Gwei
compoundAS.transferOwnership: started
compoundAS.transferOwnership: done, gas used 30444, gas price 0.000000001 Gwei
```

```
     ✓ #21 Hat should not have 0x0 recipient (173ms)
ERC20Mintable.new: started
ERC20Mintable.new: done, gas used 1498981, gas price 0.000000001 Gwei
token.mint 1000 -> customer1: started
token.mint 1000 -> customer1: done, gas used 66909, gas price 0.000000001 Gwei
ComptrollerMock.new: started
ComptrollerMock.new: done, gas used 4956159, gas price 0.000000001 Gwei
InterestRateModelMock.new: started
InterestRateModelMock.new: done, gas used 304892, gas price 0.000000001 Gwei
CErc20.new: started
CErc20.new: done, gas used 9116329, gas price 0.000000001 Gwei
CompoundAllocationStrategy.new: started
CompoundAllocationStrategy.new: done, gas used 2067660, gas price 0.000000001 Gwei
RToken.new: started
RToken.new: done, gas used 18913795, gas price 0.000000001 Gwei
Proxy.new: started
Proxy.new: done, gas used 451298, gas price 0.000000001 Gwei
compoundAS.transferOwnership: started
compoundAS.transferOwnership: done, gas used 30444, gas price 0.000000001 Gwei
ComptrollerMock.new: started
ComptrollerMock.new: done, gas used 4956159, gas price 0.000000001 Gwei
InterestRateModelMock.new: started
InterestRateModelMock.new: done, gas used 304892, gas price 0.000000001 Gwei
CErc20.new: started
CErc20.new: done, gas used 9116265, gas price 0.000000001 Gwei
CompoundAllocationStrategy.new: started
CompoundAllocationStrategy.new: done, gas used 2067660, gas price 0.000000001 Gwei
ComptrollerMock.new: started
ComptrollerMock.new: done, gas used 4956159, gas price 0.000000001 Gwei
InterestRateModelMock.new: started
InterestRateModelMock.new: done, gas used 304892, gas price 0.000000001 Gwei
CErc20.new: started
CErc20.new: done, gas used 9116329, gas price 0.000000001 Gwei
CompoundAllocationStrategy.new: started
CompoundAllocationStrategy.new: done, gas used 2067660, gas price 0.000000001 Gwei
compoundAS2.transferOwnership: started
compoundAS2.transferOwnership: done, gas used 30444, gas price 0.000000001 Gwei
compoundAS3.transferOwnership: started
compoundAS3.transferOwnership: done, gas used 30444, gas price 0.000000001 Gwei
token.approve 100 by customer1: started
token.approve 100 by customer1: done, gas used 45741, gas price 0.000000001 Gwei
rToken.mint 100 to customer1: started
rToken.mint 100 to customer1: done, gas used 446203, gas price 0.000000001 Gwei
change allocation strategy: started
change allocation strategy: done, gas used 305019, gas price 0.000000001 Gwei
change allocation strategy: started
change allocation strategy: done, gas used 305019, gas price 0.000000001 Gwei
rToken.redeem 10 to customer1: started
rToken.redeem 10 to customer1: done, gas used 147060, gas price 0.000000001 Gwei
     ✓ #22 Change allocation strategy multiple times (10070ms)

  Contract: RTokenStorage
RTokenStorageLayoutTester.new: started
RTokenStorageLayoutTester.new: done, gas used 20715043, gas price 0.000000001 Gwei
     ✓ #0 validate immutable storage layout (1113ms)


  25 passing (5m)
```

Code Coverage

The branch coverage is very low. As a best practice, we recommend improving the coverage to reach at least 80% for each contract in each category. In our opinion, shows that you not only check the happy code paths, but also the undesired code paths.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	95.28	73.33	89.04	95.36	
CompoundAllocationStrategy.sol	100	50	100	100	
IAllocationStrategy.sol	100	100	100	100	
IRToken.sol	100	100	100	100	
IRTokenAdmin.sol	100	100	100	100	
LibraryLock.sol	100	50	100	100	
Ownable.sol	36.36	50	42.86	41.67	… 58,65,69,70
Proxiable.sol	100	50	100	100	
Proxy.sol	100	50	100	100	
RToken.sol	97.1	80	92.45	97.09	… 602,667,671
RTokenStorage.sol	100	100	100	100	
RTokenStructs.sol	100	100	100	100	
ReentrancyGuard.sol	100	50	100	100	
contracts/test/	52.75	25.68	47.62	58.25	
ComptrollerMock.sol	35.82	17.54	38.89	34.85	… 159,160,161
InterestRateModelMock.sol	100	100	100	100	
RinkebyDaiFaucet.sol	100	100	100	100	
StorageLayout.sol	100	50	100	100	
TestBundle.sol	100	100	100	100	
All files	**85.82**	**43.7**	**79.79**	**86.38**	

Appendix

File Signatures

The following are the SHA-256 hashes of the audited contracts and/or test files. A smart contract or file with a different SHA-256 hash has been modified, intentionally or otherwise, after the audit. You are cautioned that a different SHA-256 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the audit.

Contracts

```
233a8bfaf41b7387d9a7686468b48d78007d962eb4ad177d8d28faa9f7a1bfe3 ./contracts/RTokenStorage.sol

9f1bc1f6baffbad577a93f0a0fbdc37374427d88fa8e7d44dbf459959f5d806e ./contracts/Proxiable.sol

ba19487144ee367b25c5b9e466fe957dbad1ff6eb106a4c0cae7e2333f78548d ./contracts/IAllocationStrategy.sol

0c9a743fce84803bed48ec054afb4053bf6f9d27bc96415c055522596c84a427 ./contracts/RTokenStructs.sol

847dcb160caa6cff695e63f6e2615e560cac0d009b2dfaf3210e70cfc565c6d3 ./contracts/Proxy.sol

8aa5b1bf84ee78ceb4f39bc0ffd51dc52f18752050bbe62d32d0ad4406cfcd6f ./contracts/ReentrancyGuard.sol

a74c7e196b0901570a168001f078eaf1a880594d0b30bd06215620b23f0b3d2e ./contracts/IRTokenAdmin.sol

409756cb1cb84472aa9675ccca31d3c4bfeca564b57ce9835c6a3a09449f9060 ./contracts/CompoundAllocationStrategy.sol

58dc74ff69dbd6e7cd429cf33406c031606b002915219dff0d30ecfa6bb9cf80 ./contracts/IRToken.sol

8409c210b46915dd34a73fc4e1ffec432818afcae55fd908cbc78cdb16553189 ./contracts/RToken.sol

4f3a34b5894241ea4fd35db6f3038fbd4a23b0747f7115cfd8b53acddb9e9d33 ./contracts/Ownable.sol

2e169692eed6eac3cf884addbdbca4471f2b42c5ab304f4ec8ee57cee6c2a3ed ./contracts/LibraryLock.sol

de850b5bfc1a1d10e75b8513f6709fd8c9442b2d8b1d0aabd1c5585a1e7e7cda ./contracts/test/StorageLayout.sol

7fdd7af6c008cbdd72932c6160a5589feb23c3ea835f2bb1bf59a50e592977d3 ./contracts/test/TestBundle.sol

ce2045518e4221238f0576e53bcbf0661d09b3bd4ec6ddb9970441b2370ccdc3 ./contracts/test/ComptrollerMock.sol

ea5ba01b3fc42af9f2ed01d8d82d4aeae3b97606fd7a751be2d8ecf7ea538270 ./contracts/test/RinkebyDaiFaucet.sol

432c861f02919096b29527b42a90d110d61b1a77c76d77d815cdf66ef07e424f ./contracts/test/InterestRateModelMock.sol
```

Tests

```
4c1f623f4023c53c35add93b39d18779fd2c339593d585345b5fa098e9cbe486 ./test/.eslintrc.js

823e759eb716c4f788acc36b5d6921e34bb0b5feddf0dc830fcc0b85b58cb5bc ./test/package.test.js

48554089723fb9030c5c3ac015f4c311e7e1ff0878829fa52d8d2d46743442d1 ./test/RToken.test.js

56f12e2febff3222061645832da594d6df36a9cf356d1b06c1a604df9290f1e3 ./test/StorageLayout.test.js

4e8b46645c79a06c03e12b1817c3917e1c8fa885a8e4fdd78e55660bff643a75 ./test/CToken.test.js

b1f853f9c0c2c81f222c1e81e127111bebb6f8c3183cabc359ac5a9528fc40ee ./test/all.js
```

About Quantstamp

Quantstamp is a Y Combinator-backed company that helps to secure smart contracts at scale using computer-aided reasoning tools, with a mission to help boost adoption of this exponentially growing technology.

Quantstamp's team boasts decades of combined experience in formal verification, static analysis, and software verification. Collectively, our individuals have over 500 Google scholar citations and numerous published papers. In its mission to proliferate development and adoption of blockchain applications, Quantstamp is also developing a new protocol for smart contract verification to help smart contract developers and projects worldwide to perform cost-effective smart contract security audits.

To date, Quantstamp has helped to secure hundreds of millions of dollars of transaction value in smart contracts and has assisted dozens of blockchain projects globally with its white glove security auditing services. As an evangelist of the blockchain ecosystem, Quantstamp assists core infrastructure projects and leading community initiatives such as the Ethereum Community Fund to expedite the adoption of blockchain technology.

Finally, Quantstamp's dedication to research and development in the form of collaborations with leading academic institutions such as National University of Singapore and MIT (Massachusetts Institute of Technology) reflects Quantstamp's commitment to enable world-class smart contract innovation.