

# C.R.E.A.M Finance 2nd PQ Review

Score: 83%

---

This is a [C.R.E.A.M Finance](#) Process Quality Review completed on 12 April 2021. It was performed using the Process Review process (version 0.6.2) and is documented [here](#). This is a revision of an earlier review (from Oct 2020) that is found [here](#). The review was performed by ShinkaRex of DeFiSafety. Check out our [Telegram](#).

The final score of the review is 83%, a strong pass. The breakdown of the scoring is in [Scoring Appendix](#). For our purposes, a pass is 70%.

## Summary of the Process

Very simply, the review looks for the following declarations from the developer's site. With these declarations, it is reasonable to trust the smart contracts.

- **Here are my smart contracts on the blockchain**
- **Here is the documentation that explains what my smart contracts do**
- **Here are the tests I ran to verify my smart contract**
- **Here are the audit(s) performed on my code by third party experts**

## Disclaimer

This report is for informational purposes only and does not constitute investment advice of any kind, nor does it constitute an offer to provide investment advisory or other services. Nothing in this report shall be considered a solicitation or offer to buy or sell any security, token, future, option or other financial instrument or to offer or provide any investment advice or service to any person in any jurisdiction. Nothing contained in this report constitutes investment advice or offers any opinion with respect to the suitability of any security, and the views expressed in this report should not be taken as advice to buy, sell or hold any security. The information in this report should not be relied upon for the purpose of investing. In preparing the information contained in this report, we have not taken into account

the investment needs, objectives and financial circumstances of any particular investor. This information has no regard to the specific investment objectives, financial situation and particular needs of any specific recipient of this information and investments discussed may not be suitable for all investors.

Any views expressed in this report by us were prepared based upon the information available to us at the time such views were written. The views expressed within this report are limited to DeFiSafety and the author and do not reflect those of any additional or third party and are strictly based upon DeFiSafety, its authors, interpretations and evaluation of relevant data. Changed or additional information could cause such views to change. All information is subject to possible correction. Information may quickly become unreliable for various reasons, including changes in market conditions or economic circumstances.

This completed report is copyright (c) DeFiSafety 2021. Permission is given to copy in whole, retaining this copyright label.

## Code and Team

This section looks at the code deployed on the Mainnet that gets reviewed and its corresponding software repository. The document explaining these questions is [here](#). This review will answer the questions;

1. Are the executing code addresses readily available? (Y/N)
2. Is the code actively being used? (%)
3. Is there a public software repository? (Y/N)
4. Is there a development history visible? (%)
5. Is the team public (not anonymous)? (Y/N)

### Are the executing code addresses readily available? (Y/N)

 Answer: Yes

They are available at website <https://docs.cream.finance/smart-contract-address> as indicated in the [Appendix](#).

## Is the code actively being used? (%)

✓ Answer: 100%

Activity is over 10 transactions a day on contract crEth lending, as indicated in the Appendix.

### Percentage Score Guidance

100%	More than 10 transactions a day
70%	More than 10 transactions a week
40%	More than 10 transactions a month
10%	Less than 10 transactions a month
0%	No activity

## Is there a public software repository? (Y/N)

✓ Answer: Yes

There is no direct link to the GitHub, but there is a GitHub link on the docs page. This leads to the GitHub for the documentation. But, from there you can find the contracts repo.

GitHub: <https://github.com/CreamFi/compound-protocol>

Is there a public software repository with the code at a minimum, but normally test and scripts also (Y/N). Even if the repo was created just to hold the files and has just 1 transaction, it gets a Yes. For teams with private repos, this answer is No.

## Is there a development history visible? (%)



Answer: 100%

With a 120 commits and 12 branches, this is a healthy repo.

This checks if the software repository demonstrates a strong steady history. This is normally demonstrated by commits, branches and releases in a software repository. A healthy history demonstrates a history of more than a month (at a minimum).

Guidance:

100%	Any one of 100+ commits, 10+branches
70%	Any one of 70+ commits, 7+branches
50%	Any one of 50+ commits, 5+branches
30%	Any one of 30+ commits, 3+branches
0%	Less than 2 branches or less than 10 commits

## Is the team public (not anonymous)? (Y/N)



Answer: Yes

Team page : <https://docs.cream.finance/about>

For a yes in this question the real names of some team members must be public on the website or other documentation. If the team is anonymous and then this question is a No.


## Documentation

This section looks at the software documentation. The document explaining these questions is [here](#).

Required questions are;

1. Is there a whitepaper? (Y/N)
2. Are the basic software functions documented? (Y/N)
3. Does the software function documentation fully (100%) cover the deployed contracts? (%)
4. Are there sufficiently detailed comments for all functions within the deployed contract code (%)
5. Is it possible to trace from software documentation to the implementation in codee (%)

## Is there a whitepaper? (Y/N)

 Answer: Yes

Location: <https://docs.cream.finance/>

There is a Gitbook which has excellent descriptions of the basic functions.

## Are the basic software functions documented? (Y/N)

 Answer: No

There is no real software documentation at all. They say that CREAM is a fork of Compound and this is indicated with some changes in the readme. However there are frequent software updates. It is unclear how much is still based on Compound. There were no references to Compound documentation and no software documentation done by CREAM.

### How to improve this score

Write the document based on the deployed code. For guidance, refer to the [SecurEth System Description Document](#).

## Does the software function documentation fully (100%)

## cover the deployed contracts? (%)

 Answer: 65%

First, this is a huge improvement to their previous review. Also what is documented, is documented well with implicit traceability. The commenting is very strong.

There is software documentation at <https://docs.cream.finance/developer/crtokens#introduction>. It documents the functions in CWrappedNativeDelegator.sol, which (according to my cursory inspection of the three contracts unique to CREAM (as in not in Compound). This seems to be the bulk of the unique code but it is tough to be sure as the differences and reasons for the changes are not well documented except in the readme briefly (which does not even mention CWrappedNativeDelegator.sol).

There is also an API, events list and full error codes, which is great.

Guidance:

- 100% All contracts and functions documented
- 80% Only the major functions documented
- 79-1% Estimate of the level of software documentation
- 0% No software documentation

### How to improve this score

This score can improve by adding content to the requirements document such that it comprehensively covers the requirements. For guidance, refer to the [SecurEth System Description Document](#) . Using tools that aid traceability detection will help.

## Are there sufficiently detailed comments for all functions within the deployed contract code (%)

✓ Answer: 90%

As per the code snippet below there are clear consistent NatSpec comments throughout. Of course, I don't know how much is Compound, how much is CREAM.

Code examples are in the [Appendix](#). As per the [SLOC](#), there is 70% commenting to code (CtC).

The Comments to Code (CtC) ratio is the primary metric for this score.

Guidance:

100%	CtC > 100	Useful comments consistently on all code
90-70%	CtC > 70	Useful comment on most code
60-20%	CtC > 20	Some useful commenting
0%	CtC < 20	No useful commenting

### How to improve this score

This score can improve by adding comments to the deployed code such that it comprehensively covers the code. For guidance, refer to the [SecurEth Software Requirements](#).

## Is it possible to trace from software documentation to the implementation in code (%)

i Answer: 60%

With no real software documentation, tracing from code documentation is impossible.

Guidance:

- 100% - Clear explicit traceability between code and documentation at a requirement level for all code
- 60% - Clear association between code and documents via non explicit traceability

40% - Documentation lists all the functions and describes their functions

0% - No connection between documentation and code

### How to improve this score

This score can improve by adding traceability from requirements to code such that it is clear where each requirement is coded. For reference, check the SecurEth guidelines on [traceability](#).

## Testing

This section looks at the software testing available. It is explained in this [document](#). This section answers the following questions;

1. Full test suite (Covers all the deployed code) (%)
2. Code coverage (Covers all the deployed lines of code, or explains misses) (%)
3. Scripts and instructions to run the tests (Y/N)
4. Packaged with the deployed code (Y/N)
5. Report of the results (%)
6. Formal Verification test done (%)
7. Stress Testing environment (%)

### Is there a Full test suite? (%)



Answer: 100%

With a test to code of 120%, this is a full test suite.

This score is guided by the Test to Code ratio (TtC). Generally a good test to code ratio is over 100%. However the reviewers best judgement is the final deciding factor.

Guidance:

100% TtC > 120% Both unit and system test visible

80% TtC > 80% Both unit and system test visible



40%      TtC < 80% Some tests visible  
0%        No tests obvious

## Code coverage (Covers all the deployed lines of code, or explains misses) (%)

 Answer: 50%

Despite having a test report, there is no indication of the code coverage resulting from the test run. Therefore a score of 50% is given.

Guidance:

100% - Documented full coverage

99-51% - Value of test coverage from documented results

50% - No indication of code coverage but clearly there is a reasonably complete set of tests


30% - Some tests evident but not complete

0% - No test for coverage seen

### How to improve this score

This score can improve by adding tests achieving full code coverage. A clear report and scripts in the software repository will guarantee a high score.

## Scripts and instructions to run the tests (Y/N)

 Answer: Yes

## Packaged with the deployed code (Y/N)

 Answer: Yes

## Report of the results (%)

 Answer: 70%

A CircleCI test report is available (<https://app.circleci.com/pipelines/github/CreamFi/compound-protocol>) and it indicates the tests running and passing, without comment or coverage or explanation. But this much better than many others.

Guidance:

100% - Detailed test report as described below

70% - GitHub Code coverage report visible

0% - No test report evident

### How to improve this score

Add a report with the results. The test scripts should generate the report or elements of it.

## Formal Verification test done (%)

 Answer: 0%

No evidence of formal verification is found.

## Stress Testing environment (%)

 Answer: 0%

No evidence of test networks is found.

# Audits



Answer: 70%

A single quick audit was performed by a top notch auditing firm; Trail of Bits. This was done in January and V2 of CREAM appears was deployed around Aug 2020.

Location: <https://github.com/trailofbits/publications/blob/master/reviews/CREAMSummary.pdf>

Guidance:

1. Multiple Audits performed before deployment and results public and implemented or not required (100%)
2. Single audit performed before deployment and results public and implemented or not required (90%)
3. Audit(s) performed after deployment and no changes required. Audit report is public. (70%)
4. No audit performed (20%)
5. Audit Performed after deployment, existence is public, report is not public and no improvements deployed OR smart contract address' not found, question 1 (0%)

## Appendices

### Author Details

The author of this review is Rex of DeFi Safety.

Email : [rex@defisafety.com](mailto:rex@defisafety.com) Twitter : @defisafety

I started with Ethereum just before the DAO and that was a wonderful education. It showed the importance of code quality. The second Parity hack also showed the importance of good process. Here my aviation background offers some value.

Aerospace knows how to make reliable code using quality processes.

I was coaxed to go to EthDenver 2018 and there I started [SecuEth.org](#) with Bryant and Roman. We created guidelines on good processes for blockchain code development. We got [EthFoundation funding](#) to assist in their development.

Process Quality Reviews are an extension of the SecurEth guidelines that will further increase the quality processes in Solidity and Vyper development.

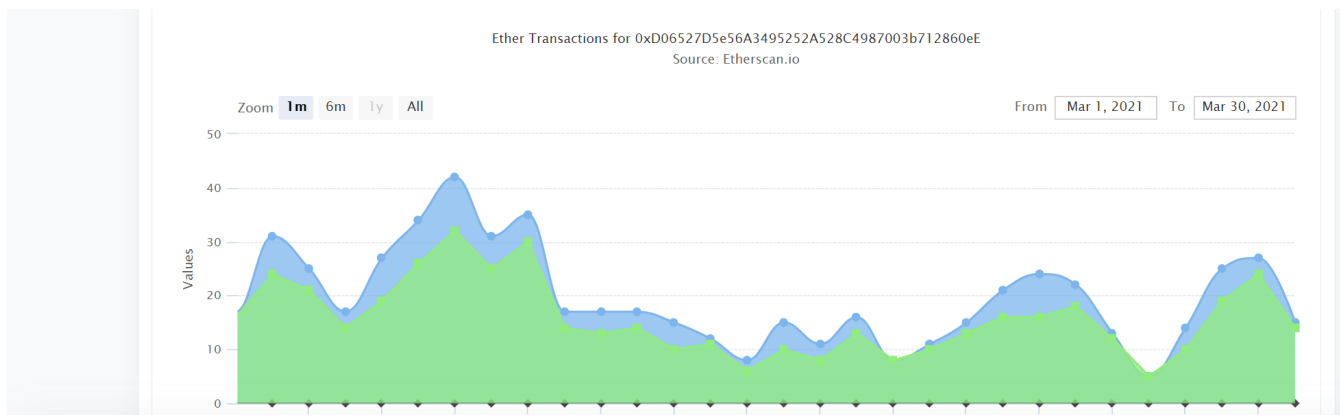
DeFiSafety is my full time gig and we are working on funding vehicles for a permanent staff.

## Scoring Appendix

PQ Audit Scoring Matrix (v0.6)	Total	CREAM	
	Points	Answer	Points
Total	240		200.25
<b>Code and Team</b>			<b>83%</b>
1. Are the executing code addresses readily available? (Y/N)	30	Y	30
2. Is the code actively being used? (%)	10	100%	10
3. Is there a public software repository? (Y/N)	5	Y	5
4. Is there a development history visible? (%)	5	100%	5
Is the team public (not anonymous)? (Y/N)	20	Y	20
<b>Code Documentation</b>			
1. Is there a whitepaper? (Y/N)	5	Y	5
2. Are the basic software functions documented? (Y/N)	10	Y	10
3. Does the software function documentation fully (100%) cover the deployed contracts? (%)	15	65%	9.75
4. Are there sufficiently detailed comments for all functions within the deployed contract code (%)	10	90%	9
5. Is it possible to trace from software documentation to the implementation in code (%)	5	60%	3
<b>Testing</b>			
1. Full test suite (Covers all the deployed code) (%)	20	100%	20
2. Code coverage (Covers all the deployed lines of code, or explains misses) (%)	5	50%	2.5
3. Scripts and instructions to run the tests? (Y/N)	5	Y	5
4. Packaged with the deployed code (Y/N)	5	Y	5
5. Report of the results (%)	10	70%	7
6. Formal Verification test done (%)	5	0%	0
7. Stress Testing environment (%)	5	0%	0
			5
<b>Audits</b>			
Audit done	70	70%	49
<b>Section Scoring</b>			
Code and Team	70	100%	
Documentation	45	82%	
Testing	55	72%	
Audits	70	70%	

## Executing Code Appendix





## Example Code Appendix

```

1      // closeFactorMantissa must not exceed this value
2      uint internal constant closeFactorMaxMantissa = 0.9e18; // 0.9
3
4      // No collateralFactorMantissa may exceed this value
5      uint internal constant collateralFactorMaxMantissa = 0.9e18; // 0.9
6
7      // liquidationIncentiveMantissa must be no less than this value
8      uint internal constant liquidationIncentiveMinMantissa = 1.0e18; // 1
9
10     // liquidationIncentiveMantissa must be no greater than this value
11     uint internal constant liquidationIncentiveMaxMantissa = 1.5e18; // 1
12
13     constructor() public {
14         admin = msg.sender;
15     }
16
17     /** Assets You Are In */
18
19     /**
20      * @notice Returns the assets an account has entered
21      * @param account The address of the account to pull assets for
22      * @return A dynamic list with the assets the account has entered
23      */
24     function getAssetsIn(address account) external view returns (CToken[]
25         CToken[] memory assetsIn = accountAssets[account];
26
27         return assetsIn;
28     }
29
30     /**
31      * @notice Returns whether the given account is entered in the given
32      * @param account The address of the account to check
33      * @param cToken The cToken to check
34      * @return True if the account is in the asset, otherwise false.
35      */
36     function checkMembership(address account, CToken cToken) external vie

```

```
37         return markets[address(cToken)].accountMembership[account];
38     }
39
40     /**
41     * @notice Add assets to be included in account liquidity calculation
42     * @param cTokens The list of addresses of the cToken markets to be e
43     * @return Success indicator for whether each corresponding market wa
44     */
45     function enterMarkets(address[] memory cTokens) public returns (uint[
46         uint len = cTokens.length;
47
48         uint[] memory results = new uint[](len);
49         for (uint i = 0; i < len; i++) {
50             CToken cToken = CToken(cTokens[i]);
51
52             results[i] = uint(addToMarketInternal(cToken, msg.sender));
53         }
54
55         return results;
56     }
57
58     /**
59     * @notice Add the market to the borrower's "assets in" for liquidity
60     * @param cToken The market to enter
61     * @param borrower The address of the account to modify
62     * @return Success indicator for whether the market was entered
63     */
64     function addToMarketInternal(CToken cToken, address borrower) interna
65         Market storage marketToJoin = markets[address(cToken)];
66
67         if (!marketToJoin.isListed) {
68             // market is not listed, cannot join
69             return Error.MARKET_NOT_LISTED;
70         }
71
72         if (marketToJoin.accountMembership[borrower] == true) {
73             // already joined
74             return Error.NO_ERROR;
75         }
76
77         if (accountAssets[borrower].length >= maxAssets) {
78             // no space, cannot join
79             return Error.TOO_MANY_ASSETS;
80         }
81
82         // survived the gauntlet, add to list
83         // NOTE: we store these somewhat redundantly as a significant opti
84         // this avoids having to iterate through the list for the most c
85         // that is, only when we need to perform liquidity checks
86         // and not whenever we want to check if an account is in a parti
87         marketToJoin.accountMembership[borrower] = true;
88         accountAssets[borrower].push(cToken);
```

```
89
90         emit MarketEntered(cToken, borrower);
91
92         return Error.NO_ERROR;
93     }
```

## SLOC Appendix

### Solidity Contracts

Language	Files	Lines	Blanks	Comments	Code	Complexity
Solidity	36	14463	2248	5026	7189	1351

Comments to Code  $5026 / 7189 = 70\%$

### Javascript Tests

Language	Files	Lines	Blanks	Comments	Code	Complexity
JavaScript	65	10900	1732	531	8637	244

Tests to Code  $8637 / 7189 = 120\%$