Compound Finance Process Quality Review

Score: 93%

Overview

This is a Compound Finance Process Quality Review completed on July 19th 2021. It was performed using the Process Review process (version 0.7.3) and is documented here. The review was performed by Nic of DeFiSafety. Check out our Telegram.

The final score of the review is 93%, a 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**
- **Here are the admin controls and strategies**

Disclaimer

Chain

This section indicates the blockchain used by this protocol.

> ✓ **Chain: Ethereum**

Guidance:

Ethereum
Binance Smart Chain
Polygon

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 following questions:

1) Are the executing code addresses readily available? (%)
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)

1) Are the executing code addresses readily available? (%)

> ✓ Answer: 100%

They are available at website https://compound.finance/docs#networks, as indicated in the Appendix.

Guidance:

100%	Clearly labelled and on website, docs or repo, quick to find
70%	Clearly labelled and on website, docs or repo but takes a bit of looking
40%	Addresses in mainnet.json, in discord or sub graph, etc
20%	Address found but labeling not clear or easy to find
0%	Executing addresses could not be found

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

> ✓ Answer: 100%

Activity is 130 transactions a day on contract *Unitroller.sol*, as indicated in the Appendix.

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

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

> ✓ Answer: Yes

GitHub: https://github.com/compound-finance/compound-protocol.

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

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

> ✓ Answer: 100%

With 57 commits and 23 branches, this is a healthy software repository.

This metric 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 30 commits

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

> ✓ Answer: Yes

Public team info is available at https://compound.finance/about.

For a **"Yes"** in this question, the real names of some team members must be public on the website or other documentation (LinkedIn, etc). If the team is anonymous, then this question is a **"No"**.

Documentation

This section looks at the software documentation. The document explaining these questions is here.

Required questions are;

6) Is there a whitepaper? (Y/N)
7) Are the basic software functions documented? (Y/N)
8) Does the software function documentation fully (100%) cover the deployed contracts? (%)
9) Are there sufficiently detailed comments for all functions within the deployed contract code (%)
10) Is it possible to trace from software documentation to the implementation in code (%)

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

> ✓ Answer: Yes

Location: https://compound.finance/documents/Compound.Whitepaper.pdf.

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

> ✓ Answer: Yes

The Compound Finance software functions (code) are documented at https://github.com/compound-finance/compound-protocol/blob/master/docs/CompoundProtocol.pdf.

8) Does the software function documentation fully (100%) cover the deployed contracts? (%)

Answer: 100%

The Compound Finance software functions (code) are all methodically described, one-by-one, at https://github.com/compound-finance/compound-protocol/blob/master/docs/CompoundProtocol.pdf, and https://compound.finance/docs/compound-js#introduction

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

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

Answer: 64%

Code examples are in the Appendix. As per the SLOC, there is 64% commenting to code (CtC).

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

Note: CtC was calculated only with the code that was authored by Compound Finance. Third-party contracts from protocols such as OpenZeppelin or the EIP interfaces were not factored into the CtC calculation.

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.

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

Answer: 100%

There is clear and explicit traceability between the Compound software functions and their implementation to their source code at https://compound.finance/docs/compound-js#introduction.

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

Testing

This section looks at the software testing available. It is explained in this document. This section answers the following questions;

11) Full test suite (Covers all the deployed code) (%)
12) Code coverage (Covers all the deployed lines of code, or explains misses) (%)
13) Scripts and instructions to run the tests (Y/N)
14) Report of the results (%)
15) Formal Verification test done (%)
16) Stress Testing environment (%)

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

✓ Answer: 90%

Code examples are in the Appendix. As per the SLOC, there is 116% testing to code (TtC).

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

How to improve this score:

This score can improved by adding tests to fully cover the code. Document what is covered by traceability or test results in the software repository.

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

ⓘ Answer: 52%

According to codecov, Compound Finance has a code coverage value of 52% from their documented results.

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 improved by adding tests that achieve full code coverage. A clear report and scripts in the software repository will guarantee a high score.

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

> ✓ Answer: Yes

Scrips/Instructions location: Scripts can be found here, and instructions to run tests can be found at the bottom of here.

14) Report of the results (%)

> ⓘ Answer: 70%

The Compound Finance codecov report constitutes a basic GitHub code coverage report.

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.

15) Formal Verification test done (%)

> ✓ Answer: 100%

Compound finance was formally verified by Certora.

16) Stress Testing environment (%)

> ✓ Answer: 100%

There is evidence of Compound Finance test-net smart contract usage at https://compound.finance/docs, and the addresses are published at https://compound.finance/docs#networks

Security

This section looks at the 3rd party software audits done. It is explained in this document. This section answers the following questions;

17) Did 3rd Party audits take place? (%)
18) Is the bounty value acceptably high?

17) Did 3rd Party audits take place? (%)

Answer: 100%

Compound has had consistent audits through their development as documented on their site. They have audits from two top level audit organizations. The audits are public and they have implemented findings in order to improve their code.

Guidance:

100%	Multiple Audits performed before deployment and results public and implemented or not required
90%	Single audit performed before deployment and results public and implemented or not required
70%	Audit(s) performed after deployment and no changes required. Audit report is public
50%	Audit(s) performed after deployment and changes needed but not implemented
20%	No audit performed
0%	Audit Performed after deployment, existence is public, report is not public and no improvements deployed OR smart contract address' not found, question

Deduct 25% if code is in a private repo and no note from auditors that audit is applicable to deployed code

18) Is the bounty value acceptably high (%)

Answer: 60%

Compound's Bug Bounty program offers rewards up to 150k.

Guidance:

100%	Bounty is 10% TVL or at least $1M AND active program (see below)
90%	Bounty is 5% TVL or at least 500k AND active program
80%	Bounty is 5% TVL or at least 500k
70%	Bounty is 100k or over AND active program
60%	Bounty is 100k or over
50%	Bounty is 50k or over AND active program
40%	Bounty is 50k or over
20%	Bug bounty program bounty is less than 50k
0%	No bug bounty program offered

An active program means that a third party (such as Immunefi) is actively driving hackers to the site. An inactive program would be static mentions on the docs.

Access Controls

This section covers the documentation of special access controls for a DeFi protocol. The admin access controls are the contracts that allow updating contracts or coefficients in the protocol. Since these contracts can allow the protocol admins to "change the rules", complete disclosure of capabilities is vital for user's transparency. It is explained in this document. The questions this section asks are as follow;

19) Can a user clearly and quickly find the status of the admin controls?
20) Is the information clear and complete?
21) Is the information in non-technical terms that pertain to the investments?
22) Is there Pause Control documentation including records of tests?

19) Can a user clearly and quickly find the status of the access

controls (%)

> Answer: 100%

Compound Finance's admin access controls are readily found at https://compound.finance/docs/governance.

Guidance:

100%	Clearly labelled and on website, docs or repo, quick to find
70%	Clearly labelled and on website, docs or repo but takes a bit of looking
40%	Access control docs in multiple places and not well labelled
20%	Access control docs in multiple places and not labelled
0%	Admin Control information could not be found

20) Is the information clear and complete (%)

> Answer: 90%

a) Contracts are clearly labelled as upgradeable through Compound's voting/implementation structure.

b) Compound outlines Defined Roles within their voting and delegation structure that are found in the Governance section of their documentation.

c) Capabilities for change in the Compound contracts can be found at https://compound.finance/docs/governance#propose.

Guidance:

All the contracts are immutable -- 100% OR

a) All contracts are clearly labelled as upgradeable (or not) -- 30% AND
b) The type of ownership is clearly indicated (OnlyOwner / MultiSig / Defined Roles) -- 30% AND
c) The capabilities for change in the contracts are described -- 30%

How to improve this score:

Create a document that covers the items described above. An example is enclosed.

21) Is the information in non-technical terms that pertain to the investments (%)

> Answer: 90%

Compound Finance explains their governance and security aspects in mostly non-technical terms. There is a lot of written code, but they are also all explained in plain english.

Guidance:

100%	All the contracts are immutable
90%	Description relates to investments safety and updates in clear, complete non-software l language
30%	Description all in software specific language
0%	No admin control information could not be found

How to improve this score:

Create a document that covers the items described above in plain language that investors can understand. An example is enclosed.

22) Is there Pause Control documentation including records of tests (%)

Answer: 80%

Compound Finance's Pause Guardian function is documented here, and the latest test documented was from February 2020 here.

Note: They would get 100% had they done and published another Pause Guardian test in the last 3 months.

Guidance:

100%	All the contracts are immutable or no pause control needed and this is explained
OR	
100%	Pause control(s) are clearly documented and there is records of at least one test within 3 months
80%	Pause control(s) explained clearly but no evidence of regular tests
40%	Pause controls mentioned with no detail on capability or tests
0%	Pause control not documented or explained

How to improve this score:

Create a document that covers the items described above in plain language that investors can understand. An example is enclosed.

Appendices

Author Details

The author of this review is Rex of DeFi Safety.

Email : rex@defisafety.com Twitter : @defisafety

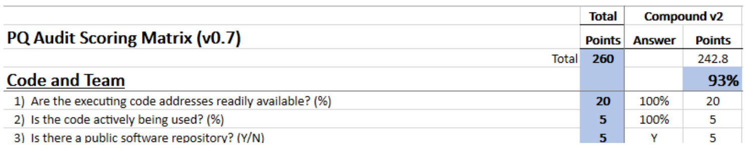
I started with Ethereum just before the DAO and that was a wonderful education. It showed the importance of code quality. The second Parity hack also showed the importance of good process. Here my aviation background offers some value. Aerospace knows how to make reliable code using quality processes.

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.

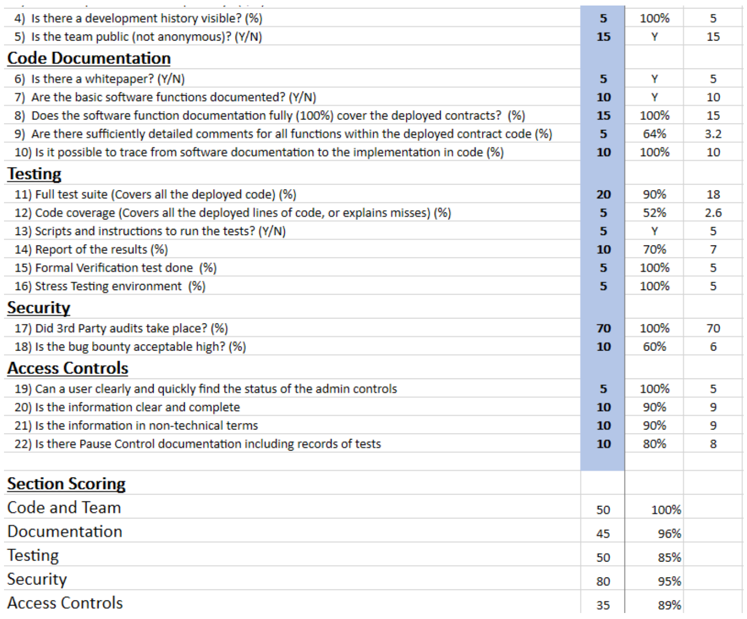
Process Quality Reviews are an extension of the SecurEth guidelines that will further increase the quality processes in Solidity and Vyper development.

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

Scoring Appendix

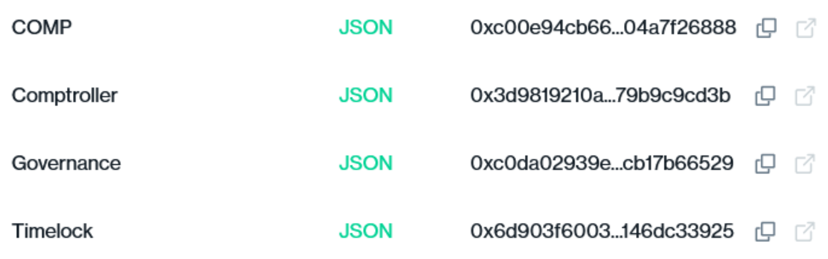


	Total	Compound v2	
PQ Audit Scoring Matrix (v0.7)	Points	Answer	Points
Total	260		242.8
Code and Team			93%
1) Are the executing code addresses readily available? (%)	20	100%	20
2) Is the code actively being used? (%)	5	100%	5
3) Is there a public software repository? (Y/N)	5	Y	5



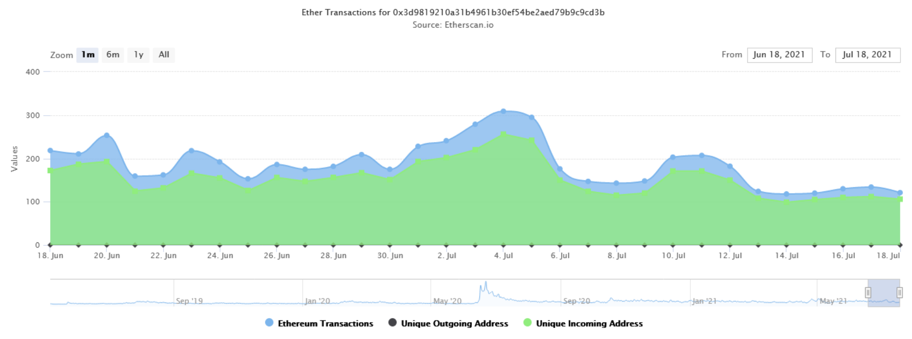
4) Is there a development history visible? (%)	**5**	100%	5
5) Is the team public (not anonymous)? (Y/N)	**15**	Y	15
Code Documentation			
6) Is there a whitepaper? (Y/N)	**5**	Y	5
7) Are the basic software functions documented? (Y/N)	**10**	Y	10
8) Does the software function documentation fully (100%) cover the deployed contracts? (%)	**15**	100%	15
9) Are there sufficiently detailed comments for all functions within the deployed contract code (%)	**5**	64%	3.2
10) Is it possible to trace from software documentation to the implementation in code (%)	**10**	100%	10
Testing			
11) Full test suite (Covers all the deployed code) (%)	**20**	90%	18
12) Code coverage (Covers all the deployed lines of code, or explains misses) (%)	**5**	52%	2.6
13) Scripts and instructions to run the tests? (Y/N)	**5**	Y	5
14) Report of the results (%)	**10**	70%	7
15) Formal Verification test done (%)	**5**	100%	5
16) Stress Testing environment (%)	**5**	100%	5
Security			
17) Did 3rd Party audits take place? (%)	**70**	100%	70
18) Is the bug bounty acceptable high? (%)	**10**	60%	6
Access Controls			
19) Can a user clearly and quickly find the status of the admin controls	**5**	100%	5
20) Is the information clear and complete	**10**	90%	9
21) Is the information in non-technical terms	**10**	90%	9
22) Is there Pause Control documentation including records of tests	**10**	80%	8
Section Scoring			
Code and Team	50	100%	
Documentation	45	96%	
Testing	50	85%	
Security	80	95%	
Access Controls	35	89%	

Executing Code Appendix



COMP	JSON	0xc00e94cb66...04a7f26888
Comptroller	JSON	0x3d9819210a...79b9c9cd3b
Governance	JSON	0xc0da02939e...cb17b66529
Timelock	JSON	0x6d903f6003...146dc33925

Code Used Appendix



Example Code Appendix

```
contract Unitroller is UnitrollerAdminStorage, ComptrollerErrorReporter {

    /**
      * @notice Emitted when pendingComptrollerImplementation is changed
      */
    event NewPendingImplementation(address oldPendingImplementation, address newPen

    /**
      * @notice Emitted when pendingComptrollerImplementation is accepted, which me
      */
    event NewImplementation(address oldImplementation, address newImplementation);

    /**
      * @notice Emitted when pendingAdmin is changed
      */
    event NewPendingAdmin(address oldPendingAdmin, address newPendingAdmin);

    /**
      * @notice Emitted when pendingAdmin is accepted, which means admin is updated
      */
    event NewAdmin(address oldAdmin, address newAdmin);

    constructor() public {
        // Set admin to caller
        admin = msg.sender;
    }

    /*** Admin Functions ***/
    function _setPendingImplementation(address newPendingImplementation) public ret

        if (msg.sender != admin) {
            return fail(Error.UNAUTHORIZED, FailureInfo.SET_PENDING_IMPLEMENTATION_
        }

        address oldPendingImplementation = pendingComptrollerImplementation;

        pendingComptrollerImplementation = newPendingImplementation;

        emit NewPendingImplementation(oldPendingImplementation, pendingComptrollerI

        return uint(Error.NO_ERROR);
    }

    /**
    * @notice Accepts new implementation of comptroller. msg.sender must be pending
    * @dev Admin function for new implementation to accept it's role as implementat
    * @return uint 0=success, otherwise a failure (see ErrorReporter.sol for detail
    */
    function _acceptImplementation() public returns (uint) {
        // Check caller is pendingImplementation and pendingImplementation ≠ addres
        if (msg.sender != pendingComptrollerImplementation || pendingComptrollerImp
            return fail(Error.UNAUTHORIZED, FailureInfo.ACCEPT_PENDING_IMPLEMENTATI
        }

        // Save current values for inclusion in log
        address oldImplementation = comptrollerImplementation;
        address oldPendingImplementation = pendingComptrollerImplementation;

        comptrollerImplementation = pendingComptrollerImplementation;

        pendingComptrollerImplementation = address(0);

        emit NewImplementation(oldImplementation, comptrollerImplementation);
        emit NewPendingImplementation(oldPendingImplementation, pendingComptrollerI

        return uint(Error.NO_ERROR);
    }


    /**
      * @notice Begins transfer of admin rights. The newPendingAdmin must call `_ac
      * @dev Admin function to begin change of admin. The newPendingAdmin must call
      * @param newPendingAdmin New pending admin.
      * @return uint 0=success, otherwise a failure (see ErrorReporter.sol for deta
      */
    function _setPendingAdmin(address newPendingAdmin) public returns (uint) {
        // Check caller = admin
        if (msg.sender != admin) {
            return fail(Error.UNAUTHORIZED, FailureInfo.SET_PENDING_ADMIN_OWNER_CHE
        }

        // Save current value, if any, for inclusion in log
        address oldPendingAdmin = pendingAdmin;

        // Store pendingAdmin with value newPendingAdmin
        pendingAdmin = newPendingAdmin;

        // Emit NewPendingAdmin(oldPendingAdmin, newPendingAdmin)
        emit NewPendingAdmin(oldPendingAdmin, newPendingAdmin);

        return uint(Error.NO_ERROR);
    }

    /**
      * @notice Accepts transfer of admin rights. msg.sender must be pendingAdmin
      * @dev Admin function for pending admin to accept role and update admin
      * @return uint 0=success, otherwise a failure (see ErrorReporter.sol for deta
      */
    function _acceptAdmin() public returns (uint) {
        // Check caller is pendingAdmin and pendingAdmin ≠ address(0)
        if (msg.sender != pendingAdmin || msg.sender == address(0)) {
            return fail(Error.UNAUTHORIZED, FailureInfo.ACCEPT_ADMIN_PENDING_ADMIN_
        }

        // Save current values for inclusion in log
        address oldAdmin = admin;
        address oldPendingAdmin = pendingAdmin;

        // Store admin with value pendingAdmin
        admin = pendingAdmin;

        // Clear the pending value
        pendingAdmin = address(0);

        emit NewAdmin(oldAdmin, admin);
        emit NewPendingAdmin(oldPendingAdmin, pendingAdmin);

        return uint(Error.NO_ERROR);
    }

    /**
     * @dev Delegates execution to an implementation contract.
     * It returns to the external caller whatever the implementation returns
     * or forwards reverts.
     */
    function () payable external {
        // delegate all other functions to current implementation
        (bool success, ) = comptrollerImplementation.delegatecall(msg.data);

        assembly {
              let free_mem_ptr := mload(0x40)
              returndatacopy(free_mem_ptr, 0, returndatasize)

              switch success
              case 0 { revert(free_mem_ptr, returndatasize) }
              default { return(free_mem_ptr, returndatasize) }
        }
    }
}
```

SLOC Appendix

Solidity Contracts

Language	Files	Lines	Blanks	Comments	Code	Complexity
Solidity	40	16410	2638	5356	8416	1772

Comments to Code 5356/8416 = 64%

Javascript Tests

Language	Files	Lines	Blanks	Comments	Code	Complexity
JavaScript	79	12294	1921	604	9769	394

Tests to Code 8416/9769 = 116%