Alpha Homora Process Quality Review

Score: 79%

This is a Alpha Homora Process Quality Review completed on 16 December 2020. It was performed using the Process Review process (version 0.6.1) and is documented here. The review was performed by ShinkaRex of DeFiSafety. Check out our Telegram.

The final score of the review is 79%, a good pass. The breakdown of the scoring is in Scoring Appendix.

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.

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

The Executing Code Addresses are available in the Alpha Finance Lab Discord channel, in the #Development-support channel, as indicated in the Appendix.

How to improve this score

Make the Ethereum addresses of the smart contract utilized by your application available on either your website or your GitHub (in the README for instance). Ensure the addresses is up to date. This is a very important question wrt to the final score.

Is the code actively being used? (%)

> Answer: 100%

Activity is *251 transactions a day on contract Bank.sol*, 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

GitHub: https://github.com/AlphaFinanceLab/alphahomora

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 103 commits and 7 branches, this is a well maintained GitHub Repository.

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

How to improve this score

Continue to test and perform other verification activities after deployment, including routine maintenance updating to new releases of testing and deployment tools. A public development history indicates clearly to the public the level of continued investment and activity by the developers on the application. This gives a level of security and faith in the application.

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

> Answer: Yes

The name of the lead developer is public and can be found on her twitter.

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://alphafinancelab.gitbook.io/alpha-homora/what-is-alpha-homora

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

> Answer: Yes

Some of the software functions are documented in the paramaters page in their documentation.

Location: https://alphafinancelab.gitbook.io/alpha-homora/key-parameters

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

> Answer: 40%

Key functions are defined in the documentation, but most functions remain undefined. There is no specific contract-by-contract documentation.

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: 80%

The comments in the code are detailed.

Code examples are in the Appendix. As per the SLOC, there is 71% 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 (%)

Answer: 0%

The documentation lists the some major functions but this does not facilitate traceability..

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 TtC ratio of 170%, there is clearly 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%

There is no indication of testing code coverage, but there is clearly a reasonable set of tests present.

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: No

There is no indication of scripts or installation to run the testing on their github, or their documentation.

How to improve this score

Add the scripts to the repository and ensure they work. Ask an outsider to create the environment and run the tests. Improve the scripts and docs based on their feedback.

Packaged with the deployed code (Y/N)

Answer: Yes

The tests are packaged with the deployed code in the primary github repository.

Report of the results (%)

Answer: 0%

There is no apparent report of the results available in their github, their documentation, or their audit.

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%

There's no indication of formal verification testing having been done.

Stress Testing environment (%)

Answer: 0%

There are no published kovan or ropsten testnet addresses, therefore no way to verify if any stress testing has been preformed.

Audits

> Answer: 90%

Alpha Homora was launched October 8th.

PeckShield Conducted an informal security audit on October 5th.

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 Caliburn Consulting.

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.

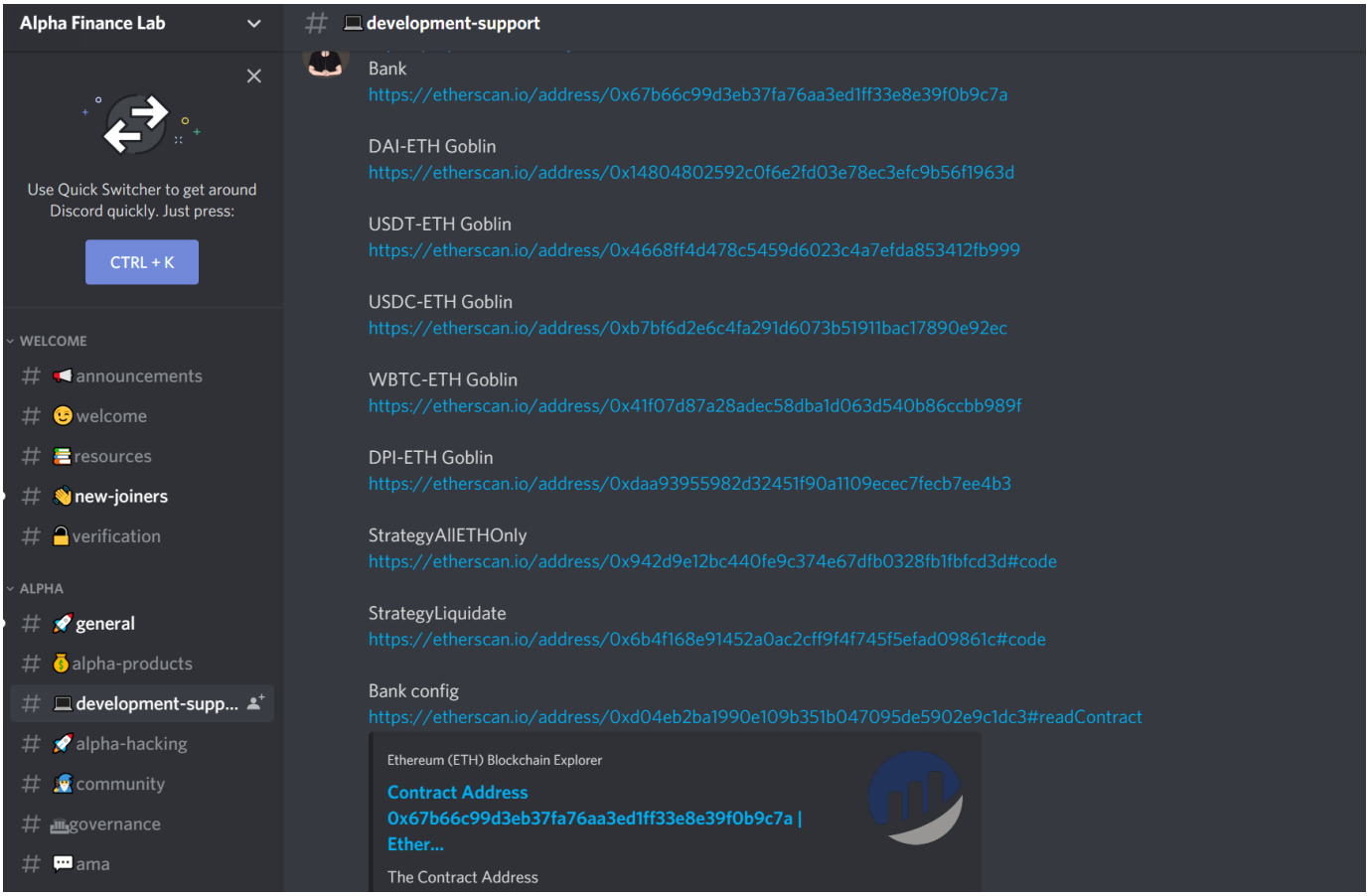
Process Quality Reviews are an extension of the SecurEth guidelines that will further increase the quality processes in Solidity and Vyper development.

Career wise I am a business development manager for an avionics supplier.

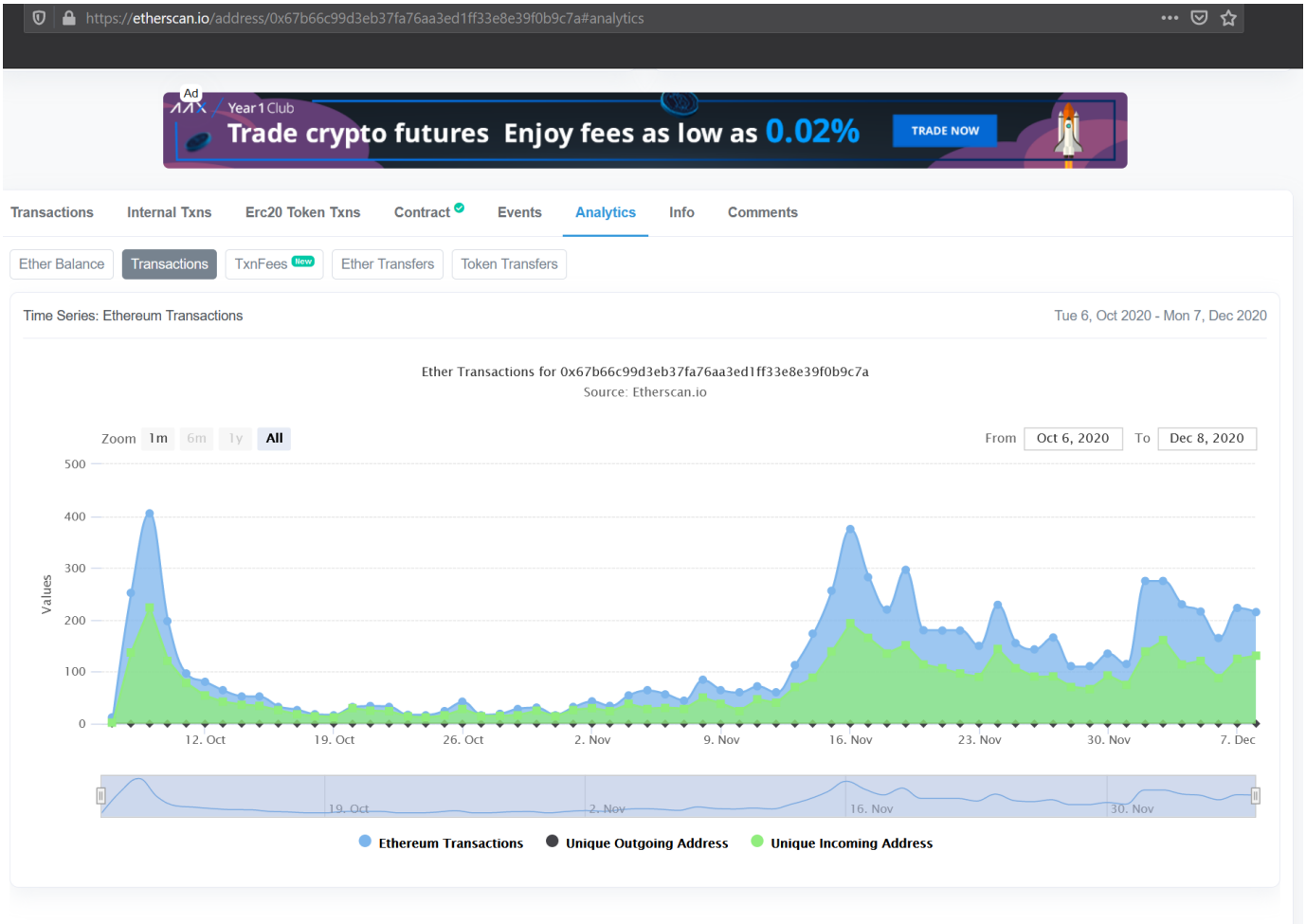
Scoring Appendix

	Total	Alpha Homera	
PQ Audit Scoring Matrix (v0.6)	**Points**	**Answer**	**Points**
Total	240		189.5
Code and Team			**79%**
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
Code Documentation			
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	40%	6
4. Are there sufficiently detailed comments for all functions within the deployed contract code (%)	10	80%	8
5. Is it possible to trace from software documentation to the implementation in code (%)	5	0%	0
Testing			
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	N	0
4. Packaged with the deployed code (Y/N)	5	Y	5
5. Report of the results (%)	10	0%	0
6. Formal Verification test done (%)	5	0%	0
7. Stress Testing environment (%)	5	0%	0
Audits			
Audit done	70	90%	63
Section Scoring			
Code and Team	70	100%	
Documentation	45	64%	
Testing	55	50%	
Audits	70	90%	

Executing Code Appendix



Code Used Appendix



Example Code Appendix

```
        // 4. Check and update position debt.
        uint256 lessDebt = Math.min(debt, Math.min(back, maxReturn));
        debt = debt.sub(lessDebt);
        if (debt > 0) {
            require(debt >= config.minDebtSize(), "too small debt size");
            uint256 health = Goblin(goblin).health(id);
            uint256 workFactor = config.workFactor(goblin, debt);
            require(health.mul(workFactor) >= debt.mul(10000), "bad work fa
            _addDebt(id, debt);
        }
        // 5. Return excess ETH back.
        if (back > lessDebt) SafeToken.safeTransferETH(msg.sender, back - le
    }

    /// @dev Kill the given to the position. Liquidate it immediately if ki
    /// @param id The position ID to be killed.
    function kill(uint256 id) external onlyEOA accrue(0) nonReentrant {
        // 1. Verify that the position is eligible for liquidation.
        Position storage pos = positions[id];
        require(pos.debtShare > 0, "no debt");
        uint256 debt = _removeDebt(id);
        uint256 health = Goblin(pos.goblin).health(id);
        uint256 killFactor = config.killFactor(pos.goblin, debt);
        require(health.mul(killFactor) < debt.mul(10000), "can't liquidate"
        // 2. Perform liquidation and compute the amount of ETH received.
        uint256 beforeETH = address(this).balance;
        Goblin(pos.goblin).liquidate(id);
        uint256 back = address(this).balance.sub(beforeETH);
        uint256 prize = back.mul(config.getKillBps()).div(10000);
        uint256 rest = back.sub(prize);
        // 3. Clear position debt and return funds to liquidator and positi
        if (prize > 0) SafeToken.safeTransferETH(msg.sender, prize);
        uint256 left = rest > debt ? rest - debt : 0;
        if (left > 0) SafeToken.safeTransferETH(pos.owner, left);
        emit Kill(id, msg.sender, prize, left);
    }

    /// @dev Internal function to add the given debt value to the given pos
    function _addDebt(uint256 id, uint256 debtVal) internal {
        Position storage pos = positions[id];
        uint256 debtShare = debtValToShare(debtVal);
        pos.debtShare = pos.debtShare.add(debtShare);
        glbDebtShare = glbDebtShare.add(debtShare);
        glbDebtVal = glbDebtVal.add(debtVal);
        emit AddDebt(id, debtShare);
    }

    /// @dev Internal function to clear the debt of the given position. Retu
    function _removeDebt(uint256 id) internal returns (uint256) {
        Position storage pos = positions[id];
```

```
51          uint256 debtShare = pos.debtShare;
52          if (debtShare > 0) {
53              uint256 debtVal = debtShareToVal(debtShare);
54              pos.debtShare = 0;
55              glbDebtShare = glbDebtShare.sub(debtShare);
56              glbDebtVal = glbDebtVal.sub(debtVal);
57              emit RemoveDebt(id, debtShare);
58              return debtVal;
59          } else {
60              return 0;
61          }
62      }
63
64      /// @dev Update bank configuration to a new address. Must only be calle
65      /// @param _config The new configurator address.
66      function updateConfig(BankConfig _config) external onlyOwner {
67          config = _config;
68      }
69
70      /// @dev Withdraw ETH reserve for underwater positions to the given addr
71      /// @param to The address to transfer ETH to.
72      /// @param value The number of ETH tokens to withdraw. Must not exceed
73      function withdrawReserve(address to, uint256 value) external onlyOwner r
74          reservePool = reservePool.sub(value);
75          SafeToken.safeTransferETH(to, value);
76      }
77
78      /// @dev Reduce ETH reserve, effectively giving them to the depositors.
79      /// @param value The number of ETH reserve to reduce.
80      function reduceReserve(uint256 value) external onlyOwner {
81          reservePool = reservePool.sub(value);
82      }
83
84      /// @dev Recover ERC20 tokens that were accidentally sent to this smart
85      /// @param token The token contract. Can be anything. This contract shou
86      /// @param to The address to send the tokens to.
87      /// @param value The number of tokens to transfer to `to`.
88      function recover(address token, address to, uint256 value) external only
89          token.safeTransfer(to, value);
90      }
91
92      /// @dev Fallback function to accept ETH. Goblins will send ETH back the
93      function() external payable {}
94  }
```

SLOC Appendix

Solidity Contracts

Language	Files	Lines	Blanks	Comments	Code	Complexity
Solidity	21	2482	233	467	1782	168

Comments to Code 1782/ 2482 = 71%

Javascript Tests

Language	Files	Lines	Blanks	Comments	Code	Complexity
JavaScript	5	1084	81	114	889	0
TypeScript	4	2783	269	372	2142	73
Total	9	3908	354	515	3039	73

Tests to Code 3039 / 1782 = 170%