Ampleforth Protocol Process Quality Review

Score: 93%

This is a Ampleforth Protocol Process Quality Review completed on 13 October 2020. It was performed using the Process Review process (version 0.5) and is documented here. The review was performed by ShinkaRex of Caliburn Consulting. Check out our Telegram.

The final score of the review is 93%, an awesome score. 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.

1. **Here is my smart contract on the blockchain**
2. **You can see it matches a software repository used to develop the code**
3. **Here is the documentation that explains what my smart contract does**
4. **Here are the tests I ran to verify my smart contract**
5. **Here are the audit(s) performed to review 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, 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. 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.

Executing Code Verification

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 address(s) readily available? (Y/N)
2. Is the code actively being used? (%)
3. Are the Contract(s) Verified/Verifiable? (Y/N)
4. Does the code match a tagged version in the code hosting platform? (%)
5. Is the software repository healthy? (%)

Are the executing code address(s) readily available? (Y/N)

Answer: Yes

Went to the bottom of the website homepage and click the GitHub icon. Addresses are in the README of the ufragments repository.

They are available at Address 0x6FB00a180781E75F87E2B690Af0196bAa77C7e7C as indicated in the Appendix. This review only covers the contract Orchestrator.

Is the code actively being used? (%)

> Answer: 70%

Activity is around 5 transactions a day, 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

Are the Contract(s) Verified/Verifiable? (Y/N)

> Answer: Yes

0x6FB00a180781E75F87E2B690Af0196bAa77C7e7C is the Etherscan verified contract address.

Does the code match a tagged version on a code hosting platform? (%)

> Answer: 100%

The code matched the latest release (1.04), except mediamoracle.sol which is in the market-oracle repo. Everything matches.

Guidance:

100% All code matches and Repository was clearly labelled
60 % All code matches but no labelled repository. Repository was found manually
30% Almost all code does match perfectly and repository was found manually
0% Most matching Code could not be found

GitHub address : https://github.com/ampleforth

Deployed contracts in the following file;

AMPL_Deployed.rar AMPL_Deployed.rar - 28KB

Matching Repository: https://github.com/ampleforth/uFragments/releases/tag/v1.0.4

How to improve this score

Ensure there is a clearly labelled repository holding all the contracts, documentation and tests for the deployed code. Ensure an appropriately labeled tag exists corresponding to deployment dates. Release tags are clearly communicated.

Is development software repository healthy? (%)

Answer: 100%

With 233 commits, 9 branches and 7 releases, this is a healthy repo.

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 application requirements documented? (Y/N)
3. Do the requirements 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 software requirements to the implementation in code (%)

Is there a whitepaper? (Y/N)

> Answer: Yes

Location: https://www.ampleforth.org/papers/

Are the basic application requirements documented? (Y/N)

> Answer: Yes

Location: https://www.ampleforth.org/basics/ https://www.ampleforth.org/technology/

How to improve this score

Write the document based on the deployed code. For guidance, refer to the SecurEth System Description Document.

Do the requirements fully (100%) cover the deployed

contracts? (%)

Answer: 50%

There is significant documentation on the logic underlying the AMPL currency. It explains the underlying logic, how it works in practice and how it works economically. However this excellent documentation never really refers to the contract software. For this reason, I score a 50% will be given.

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%

Very good commenting throughout, including at least some NatSpec parameters.

Code examples are in the Appendix. As per the SLOC, there is 72% commenting to code ratio.

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 requirements to the implementation in code (%)

⚠ Answer: 0%

The excellent documentation at no time refers to the code. This means that it is not possible to trace from the requirements to the code.

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%

Both unit and simulation tests are evident.

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

Answer: 96%

Coveralls coverage is listed as 96% No indication of what the misses are or why but that is just because I am not allowed access (which makes sense).

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

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

Answer: Yes

Testing instructions are with the uFragments repo readme.

Packaged with the deployed code (Y/N)

Answer: Yes

Report of the results (%)

> Answer: 100%

The coveralls score indicates 96%. They do not allow access to the details which would indicate the misses.

But, when reading the Quantstamp Audit report I found the first real test report in my time of building reviews (yipee ;) It indicates each test, the result and details. The only thing missing is a specific description of where and why theyre is missed coverage, but I am not complaining.

Formal Verification test done (%)

> Answer: 100%

Formal Verification was used in one of the later Certik audits; https://github.com/ampleforth/ampleforth-audits/blob/master/token-geyser/v1.0.0/CertiK_Verification_Report.pdf

Stress Testing environment (%)

> Answer: 100%

The addresses of the Rinkby testnets are in the uFragments repo readme. Some are recently active.

Audits

Ampleforth has a sequence of audits by top notch auditors. 3 independent audits on the core protocol; Trail of Bits, Quantstamp and Slow Mist. Another audit by Certik on V 1.0.2 and another Certik audit on the Orchestra abd geyser functions. All audits took place before deployment and had their corrections incorporated.

> Answer: 100%

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.

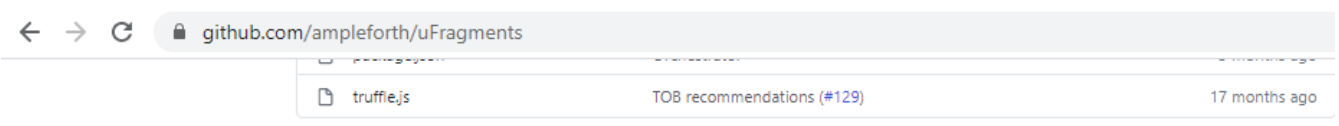
Process Quality Reviews are an extension of the SecurEth guidelines that will further increase the quality processes in Solidity and Vyper development.

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

Scoring Appendix

	Total	Ampleforth	
PQ Audit Scoring Matrix (v0.4 and 0.5)	**Points**	**Answer**	**Points**
Total	240		214.8
Executing Code Verification			**90%**
1. Is the executing code address(s) readily available? (Y/N)	30	Y	30
2. Is the code actively being used? (%)	5	70%	3.5
3. Are the Contract(s) Verified/Verifiable? (Y/N)	5	Y	5
4. Does the code match a tagged version on a code hosting platform? (%)	20	80%	16
5. Is development software repository healthy? (%)	10	100%	10
Code Documentation			
1. Is there a whitepaper? (Y/N)	5	Y	5
2. Are the basic application requirements documented? (Y/N)	10	Y	10
3. Do the requirements fully (100%) cover the deployed contracts? (%)	15	50%	7.5
4. Are there sufficiently detailed comments for all functions within the deployed contract code (%)	10	80%	8
5. Is it possible to trace requirements 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	96%	4.8
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	100%	10
6. Formal Verification test done (%)	5	0%	0
7. Stress Testing environment (%)	5	100%	5
Audits			
Audit done	70	100%	70
Section Scoring			
Executing Code Verification	70	92%	
Documentation	45	68%	
Testing	55	91%	
Audits	70	100%	

Executing Code Appendix



README.md

Ampleforth

build passing coverage 96%

Ampleforth (code name uFragments) is a decentralized elastic supply protocol. It maintains a stable unit price by adjusting supply directly to and from wallet holders. You can read the whitepaper for the motivation and a complete description of the protocol.

This repository is a collection of smart contracts that implement the Ampleforth protocol on the Ethereum blockchain.

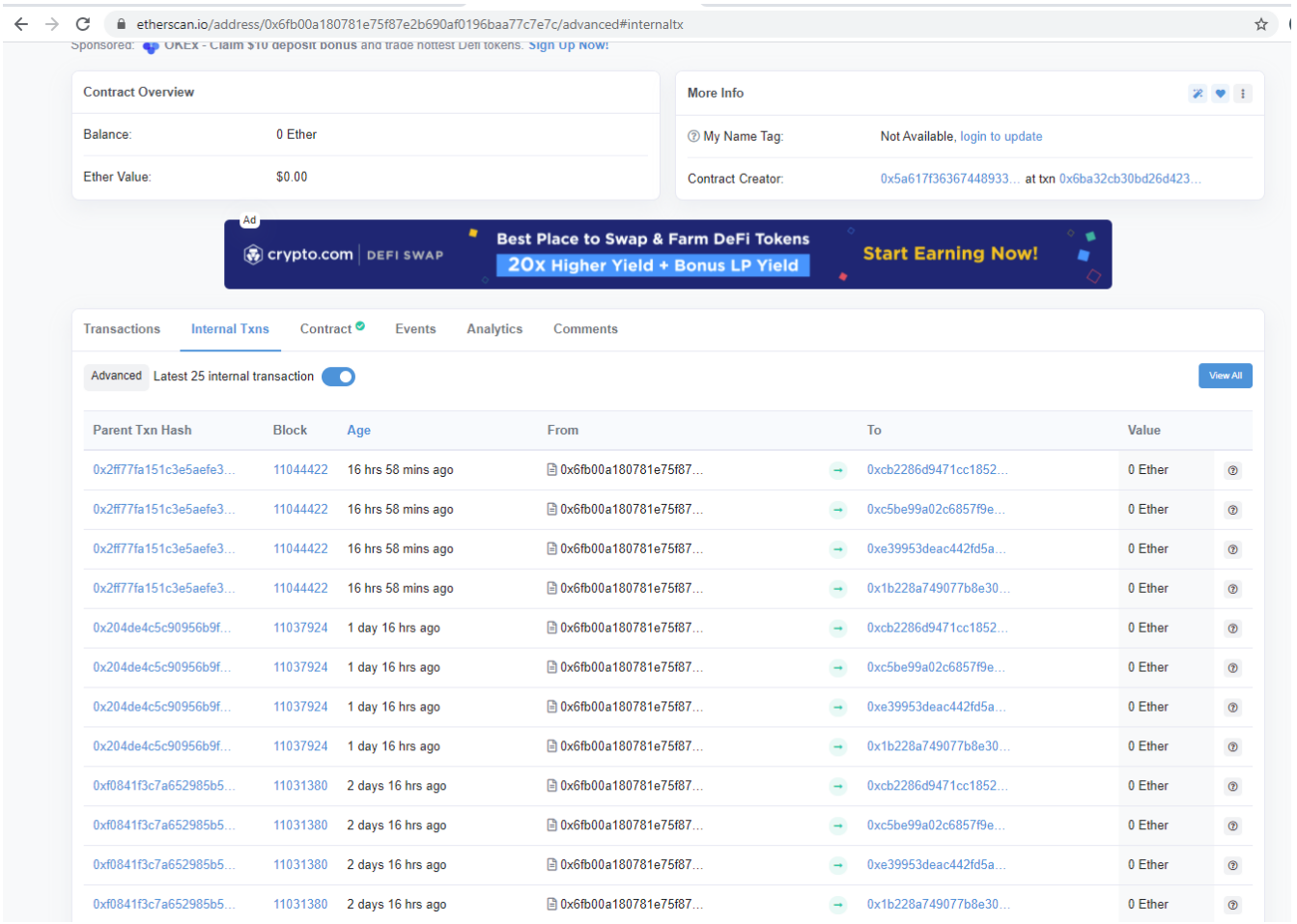
The official mainnet addresses are:

- ERC-20 Token: 0xD46bA6D942050d489DBd938a2C909A5d5039A161
- Supply Policy: 0x1B228a749077b8e307C5856cE62Ef35d96Dca2ea
- Orchestrator: 0x6fb00a180781e75f87e2b690af0196baa77c7e7c
- Market Oracle: 0x99c9775e076fdf99388c029550155032ba2d8914
- CPI Oracle: 0xa759f960dd59a1ad32c995ecabe802a0c35f244f

Table of Contents

- Install
- Testing
- Testnets
- Contribute
- License

Code Used Appendix



Example Code Appendix

```
/**
 * @title Orchestrator
 * @notice The orchestrator is the main entry point for rebase operations
 * actions with external consumers.
 */
contract Orchestrator is Ownable {

    struct Transaction {
        bool enabled;
        address destination;
        bytes data;
    }

    event TransactionFailed(address indexed destination, uint index, byte

    // Stable ordering is not guaranteed.
    Transaction[] public transactions;

    UFragmentsPolicy public policy;

    /**
     * @param policy_ Address of the UFragments policy.
     */
    constructor(address policy_) public {
        Ownable.initialize(msg.sender);
        policy = UFragmentsPolicy(policy_);
    }

    /**
     * @notice Main entry point to initiate a rebase operation.
     *         The Orchestrator calls rebase on the policy and notifies do
     *         Contracts are guarded from calling, to avoid flash loan att
     *         providers.
     *         If a transaction in the transaction list reverts, it is sw
     *         transactions are executed.
     */
    function rebase()
        external
    {
        require(msg.sender == tx.origin);  // solhint-disable-line avoid-

        policy.rebase();

        for (uint i = 0; i < transactions.length; i++) {
            Transaction storage t = transactions[i];
            if (t.enabled) {
                bool result =
```

```
48                      externalCall(t.destination, t.data);
49                  if (!result) {
50                      emit TransactionFailed(t.destination, i, t.data);
51                      revert("Transaction Failed");
52                  }
53              }
54          }
55      }
56
```

SLOC Appendix

Solidity Contracts

Language	Files	Lines	Blanks	Comments	Code	Complexity
Solidity	4	3428	553	1203	1672	180

Comments to Code 1203 / 1672 = 72%

Javascript Tests

Language	Files	Lines	Blanks	Comments	Code	Complexity
JavaScript	8	2288	342	59	1887	8

Tests to Code 1887/ 1672 = 129%