CORE (Cvault.finance) Process Quality Review

Score 66%

This is a Process Quality Review of CORE (Cvault Finance) completed on 2 November 2020. It was performed using the Process Review process (version 0.5) and is documented here. The review was performed by SentientPlant of Caliburn Consulting. Check out our Telegram.

The final score of the review is 66%, a 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.

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

They are available at Address https://github.com/cVault-finance/CORE-v1 as indicated in the Appendix. This review only covers the contract AdminUpgradableProxy.sol, which is a proxy for CoreVault.sol.

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 address 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 40 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

0xf7cA8F55c54CbB6d0965BC6D65C43aDC500Bc591 is the Etherscan verified contract address.

How to improve this score

Ensure that the deployed code is verified as described in this article for Etherscan or ETHPM. Improving this score may require redeployment.

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

✓ Answer: 100%

Code matching was an easy process.

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/cVault-finance/CORE-v2

Deployed contracts in the following file;

deployed_forutbe.rar deployed_forutbe.rar - 12KB

Matching Repository: https://github.com/cVault-finance/CORE-v2/tree/master/src/contracts

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: 50%

With 64 commits and 1 branch, this is a semi-healthy repository.

How to improve this score

Ensure there is a clearly labelled repository holding all the contracts, documentation and tests for the deployed code. Continue to test and perform other verification activities after deployment, including routine maintenance updating to new releases of testing and deployment tools.

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://medium.com/@CORE_Vault/introducing-core-fef3e1b77d12

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

✓ Answer: Yes

https://help.cvault.finance/faqs/faq provides an overview of how the formulas used in the app.

Do the requirements fully (100%) cover the deployed contracts? (%)

⚠ Answer: 0%

There are no application function documentation, and therefore the requirements do not cover any of the contracts.

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: 20%

There are limited comments within the code. High SLOC is probably due to code drawings they include in their contracts.

Code examples are in the Appendix. As per the SLOC, there is 90% commenting to code.

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%

Since there is no formal process documentation, there is no connection between the documentation and 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%

There is four test files, but the files are quite comprehensive. There are a total of 4 tests that are evident. The first 2 tests are packaged in the https://github.com/cVault-finance/CORE-v1/tree/master/test directory. The other 2 tests are packaged in the https://github.com/cVault-finance/CORE-v2/tree/master/src/test directory.

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

ⓘ Answer: 50%

No results available so 50% by default.

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 apparent indication to demonstrate how a user could run these tests themselves.

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 testing suite is packaged with the deployed code.

How to improve this score

Improving this score requires redeployment of the code, with the tests. This score gives credit to those who test their code before deployment and release them together. If a developer adds tests after deployment they can gain full points for all test elements except this one.

Report of the results (%)

> ⚠ Answer: 0%

There is no apparent report of any testing results.

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 is no indication of any formal verification testing.

Stress Testing environment (%)

> ⚠ Answer: 0%

Audits

> ✓ Answer: 70%

An audit by The Acadia Group was completed October 8th.

Cvault.Finance was released September 18th.

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

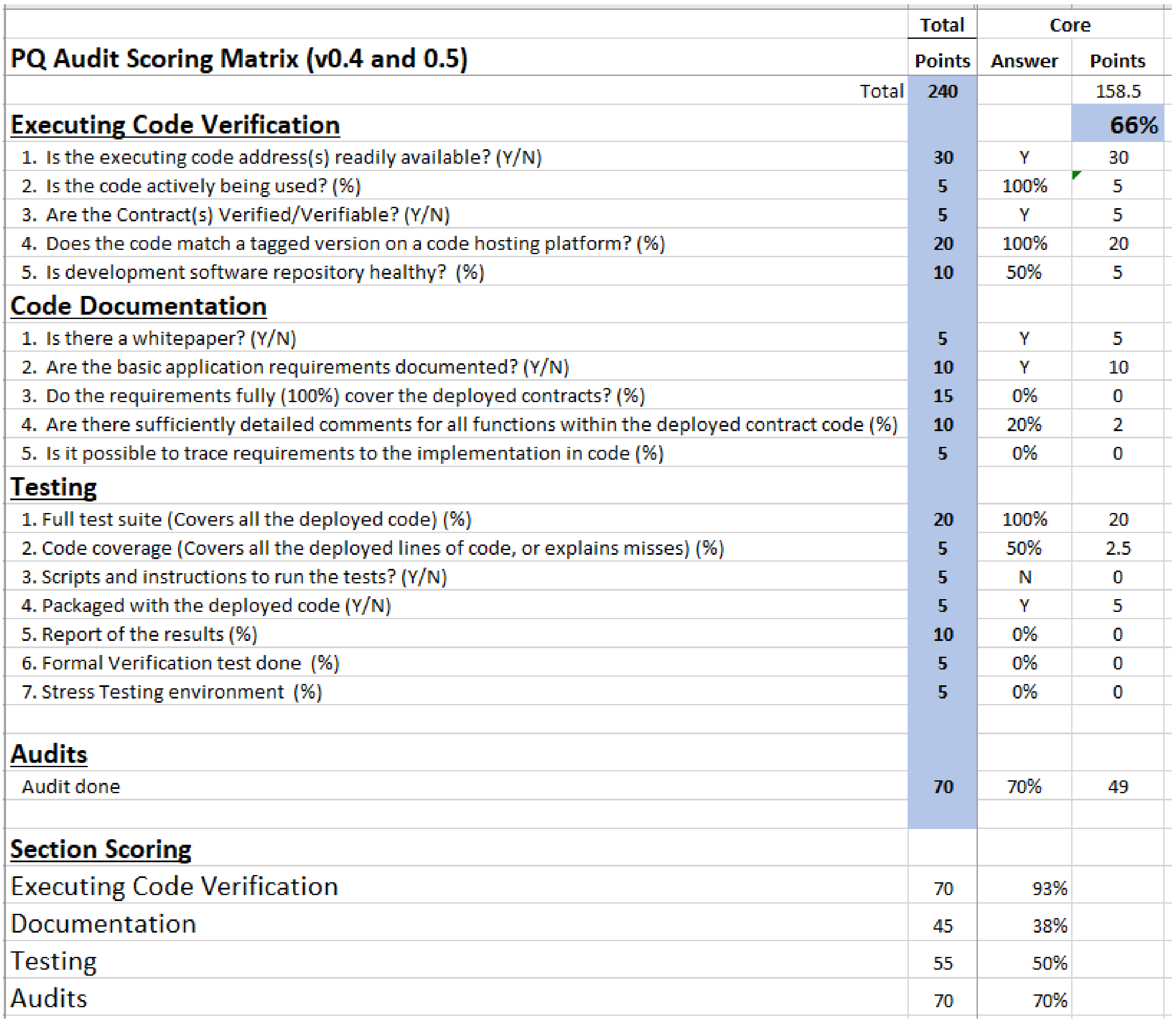
I started with Ethereum just before the DAO and that was a wonderful education. It showed the importance of code quality. The second Parity hack also showed the importance of good process. Here my aviation background offers some value. Aerospace knows how to make reliable code using quality processes.

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.

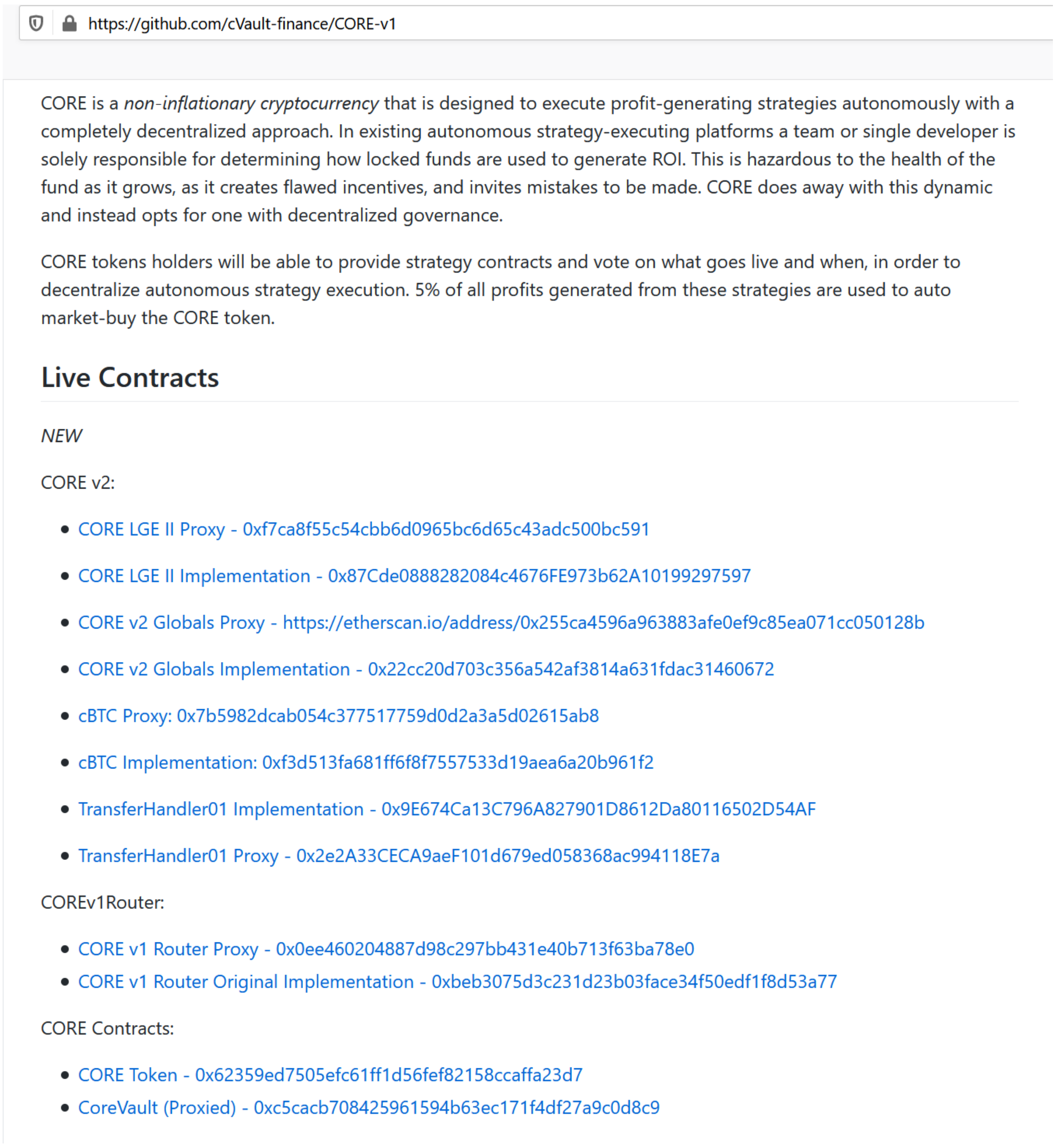
Career wise I am a business development manager for an avionics supplier.

Scoring Appendix



	Total	Core	
PQ Audit Scoring Matrix (v0.4 and 0.5)	**Points**	**Answer**	**Points**
Total	**240**		158.5
Executing Code Verification			**66%**
1. Is the executing code address(s) readily available? (Y/N)	**30**	Y	30
2. Is the code actively being used? (%)	**5**	100%	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**	100%	20
5. Is development software repository healthy? (%)	**10**	50%	5
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**	0%	0
4. Are there sufficiently detailed comments for all functions within the deployed contract code (%)	**10**	20%	2
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**	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**	70%	49
Section Scoring			
Executing Code Verification	70	93%	
Documentation	45	38%	
Testing	55	50%	
Audits	70	70%	

Executing Code Appendix



https://github.com/cVault-finance/CORE-v1

CORE is a *non-inflationary cryptocurrency* that is designed to execute profit-generating strategies autonomously with a completely decentralized approach. In existing autonomous strategy-executing platforms a team or single developer is solely responsible for determining how locked funds are used to generate ROI. This is hazardous to the health of the fund as it grows, as it creates flawed incentives, and invites mistakes to be made. CORE does away with this dynamic and instead opts for one with decentralized governance.

CORE tokens holders will be able to provide strategy contracts and vote on what goes live and when, in order to decentralize autonomous strategy execution. 5% of all profits generated from these strategies are used to auto market-buy the CORE token.

Live Contracts

NEW

CORE v2:

- CORE LGE II Proxy - 0xf7ca8f55c54cbb6d0965bc6d65c43adc500bc591
- CORE LGE II Implementation - 0x87Cde0888282084c4676FE973b62A10199297597
- CORE v2 Globals Proxy - https://etherscan.io/address/0x255ca4596a963883afe0ef9c85ea071cc050128b
- CORE v2 Globals Implementation - 0x22cc20d703c356a542af3814a631fdac31460672
- cBTC Proxy: 0x7b5982dcab054c377517759d0d2a3a5d02615ab8
- cBTC Implementation: 0xf3d513fa681ff6f8f7557533d19aea6a20b961f2
- TransferHandler01 Implementation - 0x9E674Ca13C796A827901D8612Da80116502D54AF
- TransferHandler01 Proxy - 0x2e2A33CECA9aeF101d679ed058368ac994118E7a

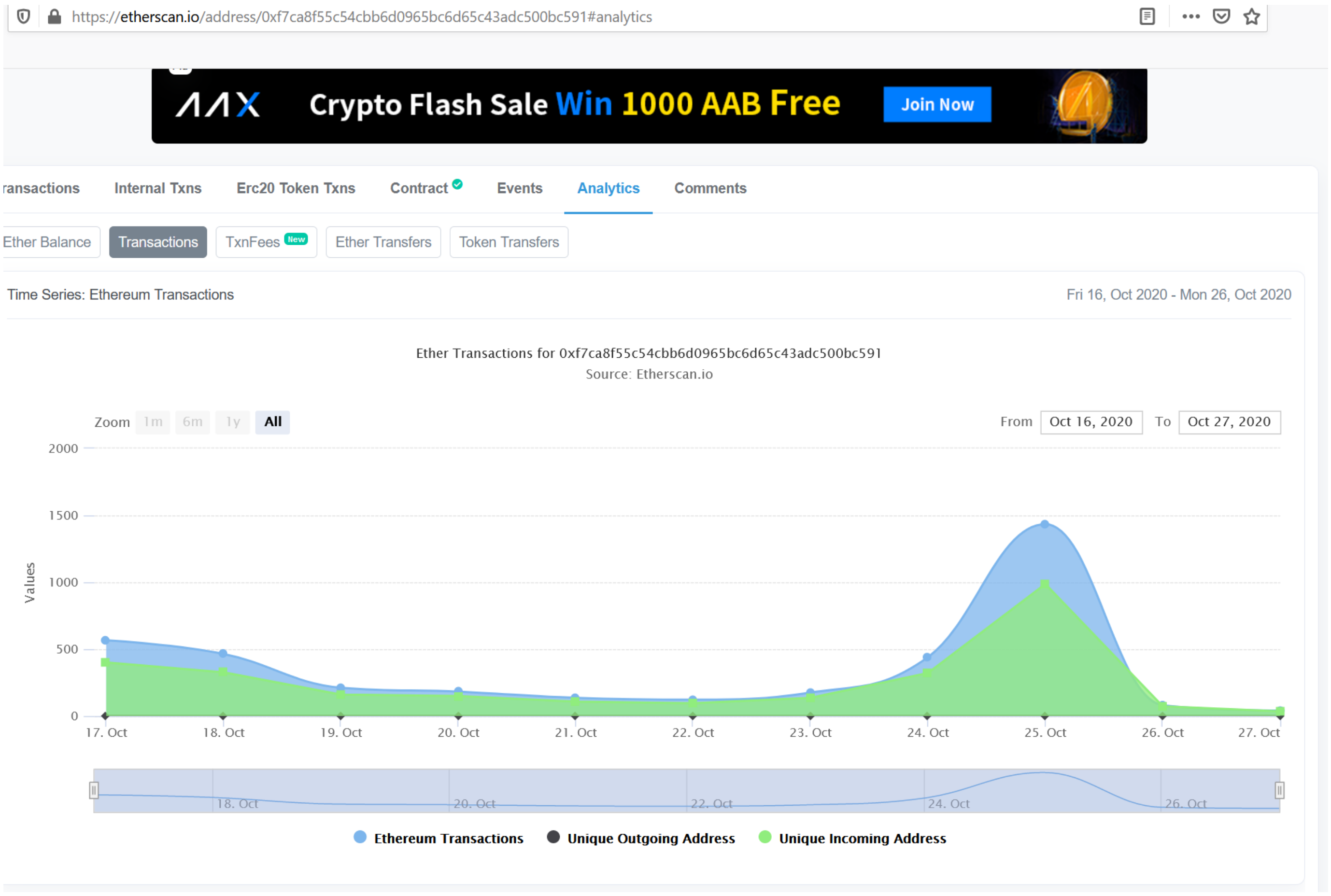
COREv1Router:

- CORE v1 Router Proxy - 0x0ee460204887d98c297bb431e40b713f63ba78e0
- CORE v1 Router Original Implementation - 0xbeb3075d3c231d23b03face34f50edf1f8d53a77

CORE Contracts:

- CORE Token - 0x62359ed7505efc61ff1d56fef82158ccaffa23d7
- CoreVault (Proxied) - 0xc5cacb708425961594b63ec171f4df27a9c0d8c9

Code Used Appendix



Example Code Appendix

```
// SPDX-License-Identifier: MIT
// COPYRIGHT cVault.finance TEAM
// NO COPY
// COPY = BAD
// This code is provided with no assurances or guarantees of any kind. Use a
//  _____ ___________ _____    _____ _       _           _
// /  __ \  _  | ___ \  ___|  |  __ \ |     | |         | |
// | /  \/ | | | |_/ / |__    | |  \/ | ___ | |__   __ _| |___
// | |   | | | |    /|  __|   | | __| |/ _ \| '_ \ / _` | / __|
// | \__/\ \_/ / |\ \| |___   | |_\ \ | (_) | |_) | (_| | \__ \
//  \____/\___/\_| \_\____/    \____/_|\___/|_.__/ \__,_|_|___/
//
// This contract stores all different CORE contract addreses
// and is responsible for contract authentification in the CORE smart contra
//
// BBBBBBBBBBBBBBBBBBBBBBBBBBB
// BMB---------------------B B
// BBB---------------------BBB
// BBB---------------------BBB
// BBB------CORE.exe-------BBB
// BBB---------------------BBB
// BBB---------------------BBB
// BBBBBBBBBBBBBBBBBBBBBBBBBBB
// BBBBB++++++++++++++BBBBBB
// BBBBB++BBBBB+++++++++BBBBBB
// BBBBB++BBBBB+++++++++BBBBBB
// BBBBB++BBBBB+++++++++BBBBBB
// BBBBB++++++++++++++++BBBBBB


import "@openzeppelin/contracts-ethereum-package/contracts/access/Ownable.s
import "hardhat/console.sol";


contract COREGlobals is OwnableUpgradeSafe {

    address public CORETokenAddress;
    address public COREGlobalsAddress;
    address public COREDelegatorAddress;
    address public COREVaultAddress;
    address public COREWETHUniPair;
    address public UniswapFactory;
    address public TransferHandler;

    function initialize(address _COREWETHUniPair, address _COREToken, addre
        OwnableUpgradeSafe.__Ownable_init();
        CORETokenAddress = _COREToken;
        COREGlobalsAddress = address(this);
        COREDelegatorAddress = _COREDelegator;
        COREVaultAddress = _COREVault;
        UniswapFactory = _uniFactory;
        TransferHandler = _transferHandler;
        COREWETHUniPair = _COREWETHUniPair;
    }

    function setCoreToken(address _COREToken) public onlyOwner {
        CORETokenAddress = _COREToken;
    }

    function setCoreDelegator(address _COREDelegator) public onlyOwner {
        COREDelegatorAddress = _COREDelegator;
    }

    function setCoreVaultAddress(address _COREVault) public onlyOwner {
        COREVaultAddress = _COREVault;
    }

    function setTransferHandler(address _transferHandler) public onlyOwner
        TransferHandler = _transferHandler;
    }

    mapping (address => bool) private delegatorStateChangeApproved;

    function addDelegatorStateChangePermission(address that, bool status) p
        return _addDelegatorStateChangePermission(that, status);
    }

    function _addDelegatorStateChangePermission(address that, bool status)
        require(isContract(that), "Only contracts");
        delegatorStateChangeApproved[that] = status;
    }

    // Only contracts.
    function isStateChangeApprovedContract(address that)  public view returr
        return _isStateChangeApprovedContract(that);
    }

    function _isStateChangeApprovedContract(address that) internal view retu
        return delegatorStateChangeApproved[that];
    }

    function isContract(address addr) public view returns (bool) {
        uint size;
        assembly { size := extcodesize(addr) }
        return size > 0;
    }
```

SLOC Appendix

Solidity Contracts

Language	Files	Lines	Blanks	Comments	Code	Complexity
Solidity	8	441	1063	1063	1176	118

Comments to Code 47/ 1949 = 90%

Javascript Tests

Language	Files	Lines	Blanks	Comments	Code	Complexity
JavaScript	2	290	54	68	68	59

Tests to Code 68 / 1176 = 5%