dForce Finance PQ Review

Score: 70%

This is a dForce Network Process Quality Review completed on 3 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 70%, a clear 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. Is 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? (%)

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

> ✓ Answer: Yes

The addresses are found in the README of the GitHub repository for each implementation; USDxProtocol, dToken, staking and xswap.

They are available at Address 0xD2fA07cD6Cd4A5A96aa86BacfA6E50bB3aaDBA8B as indicated in the Appendix. This review only covers the contract Unipool for dDai staking.

Is the code actively being used? (%)

> ✓ Answer: 70%

Activity is *1 or 2 a day* 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

0xD2fA07cD6Cd4A5A96aa86BacfA6E50bB3aaDBA8B is the Etherscan verified contract address.

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

Answer: 60%

The repos were easily found with releases but the releases were for web apps, not the contracts. The latest all matched the master branch

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/dforce-network

Deployed contracts in the following file;

dFocre_Deployed.rar dFocre_Deployed.rar - 17KB

Example Matching Repository: https://github.com/dforce-network/dToken

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%

217 commits and labelled branches make this 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://github.com/dforce-network/documents/tree/master/white_papers/en

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

⚠ Answer: No

No documentation for the software was found.

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: 0%

No documentation for the software was found.

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: 40%

Some functions such as MoneyMarketHandler.sol have good comments with NatSpec, while others such as DTokenController.sol have virtually no comments. For this reason and to be in line with other scoring projects a 40% is given.

Code examples are in the Appendix. As per the SLOC, there is 27% 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%

With no software documentation, tracing to code is obviously impossible.test

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: 50%

While the dToken repo has a full test suite, the staking and USDxProtocol have no test in their repos.

How to improve this score

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

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

Answer: 30%

No evidence of code coverage. Since only a fraction of the repos have test suites, a 50% is not valid, therefore the 30% score seemed more correct.

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

The build and script instructions were in each repo

Packaged with the deployed code (Y/N)

> ✓ Answer: Yes

Report of the results (%)

> ⚠ Answer: 0%

There were no test reports 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 was found.

Stress Testing environment (%)

> ✓ Answer: 100%

With the readme's in the getup repositories there were tests network addresses in addition to the main Ethereum addresses. Many had been in use over the past couple months.

Audits

> ✓ Answer: 100%

There are multiple audits which take place at different times during the development from both Peckshield and Trail of Bits at the following address: https://github.com/dforce-network/documents

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

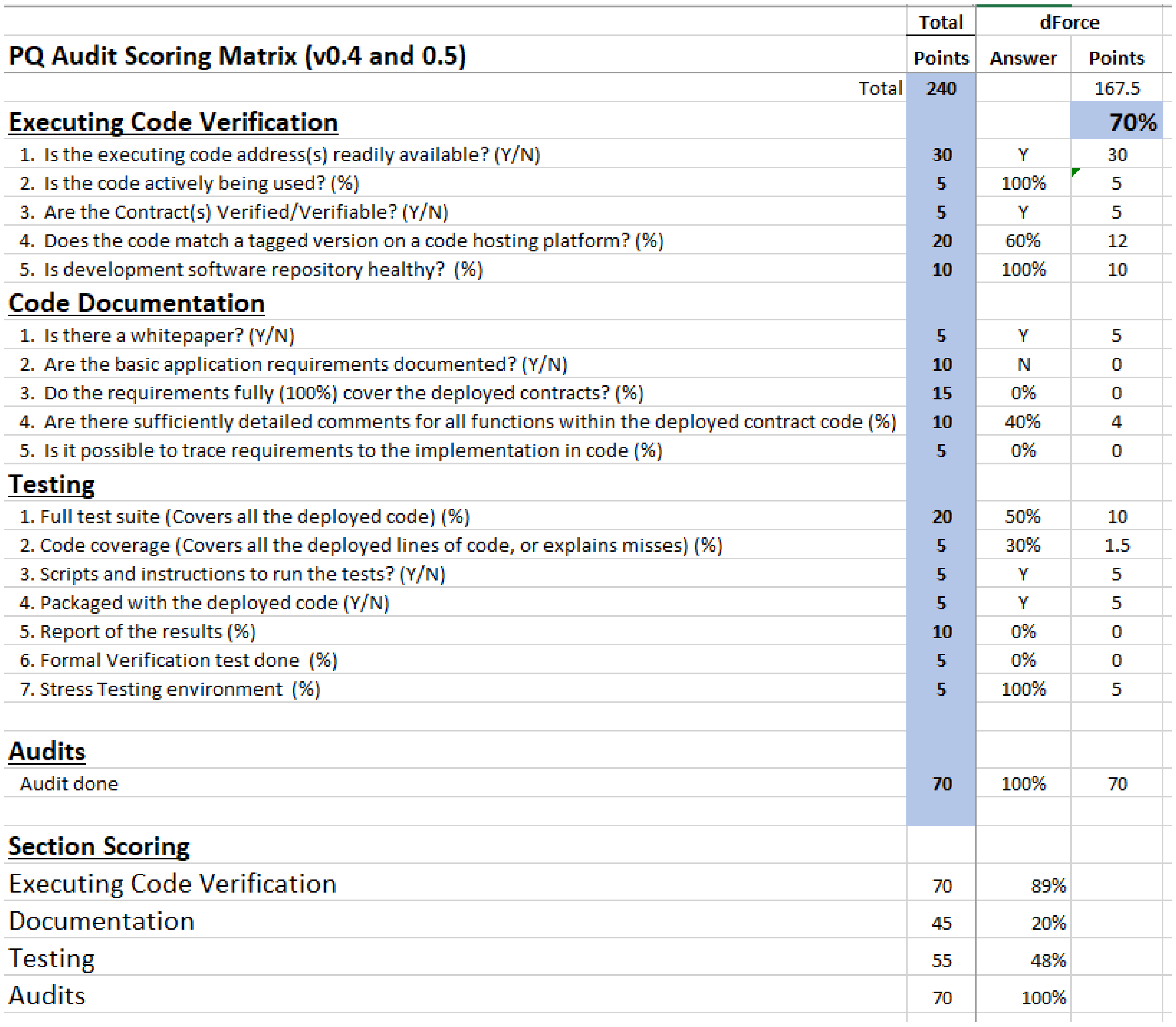
I started with Ethereum just before the DAO and that was a wonderful education. It showed the importance of code quality. The second Parity hack also showed the importance of good process. Here my aviation background offers some value. Aerospace knows how to make reliable code using quality processes.

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.

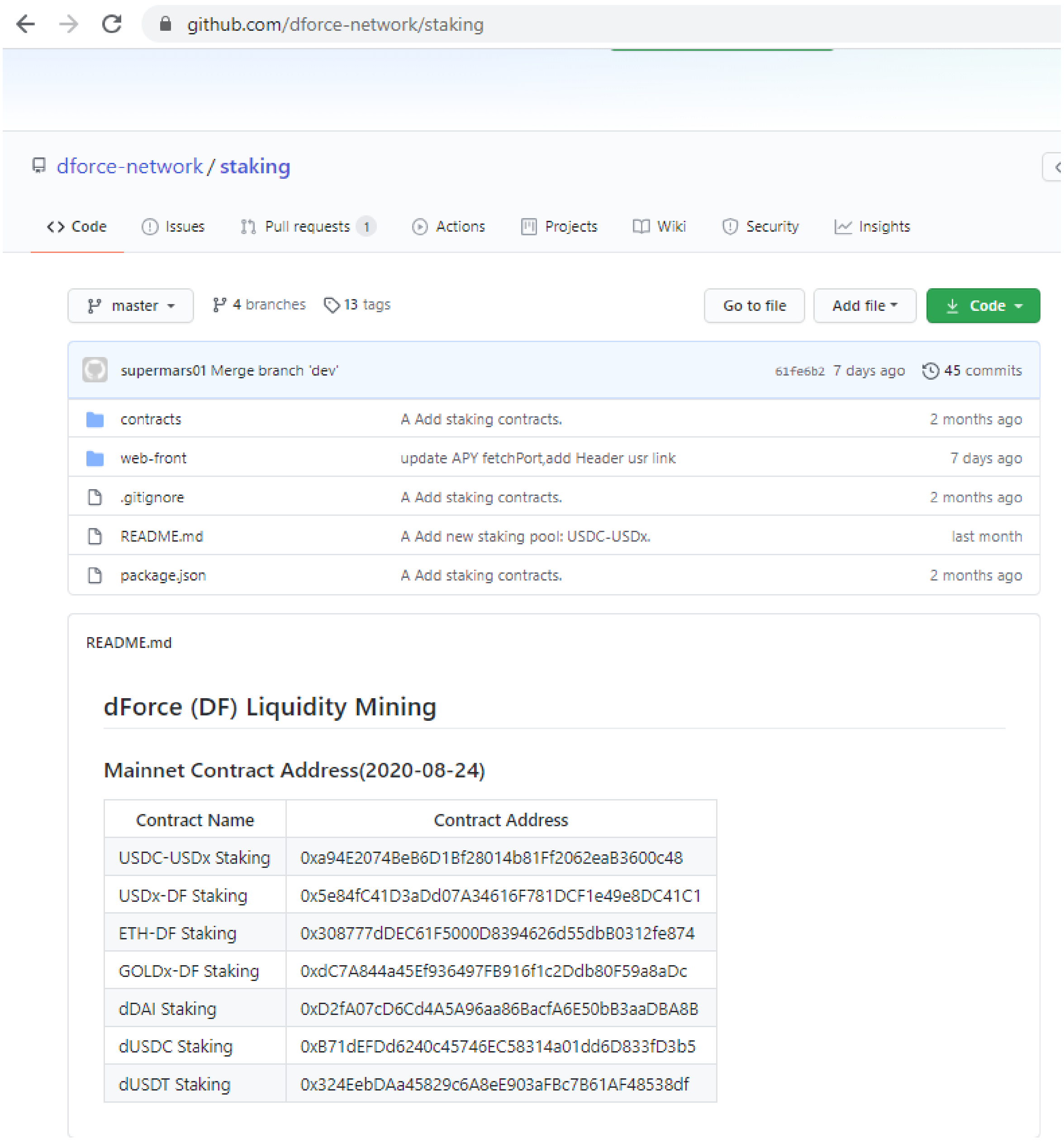
Career wise I am a business development manager for an avionics supplier.

Scoring Appendix



	Total	dForce	
PQ Audit Scoring Matrix (v0.4 and 0.5)	Points	Answer	Points
Total	240		167.5
Executing Code Verification			70%
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	60%	12
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	N	0
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	40%	4
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	50%	10
2. Code coverage (Covers all the deployed lines of code, or explains misses) (%)	5	30%	1.5
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	0%	0
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	89%	
Documentation	45	20%	
Testing	55	48%	
Audits	70	100%	

Executing Code Appendix



github.com/dforce-network/staking

dforce-network / staking

Code | Issues | Pull requests 1 | Actions | Projects | Wiki | Security | Insights

master | 4 branches | 13 tags | Go to file | Add file | Code

supermars01 Merge branch 'dev' 61fe6b2 7 days ago 45 commits

contracts	A Add staking contracts.	2 months ago
web-front	update APY fetchPort,add Header usr link	7 days ago
.gitignore	A Add staking contracts.	2 months ago
README.md	A Add new staking pool: USDC-USDx.	last month
package.json	A Add staking contracts.	2 months ago

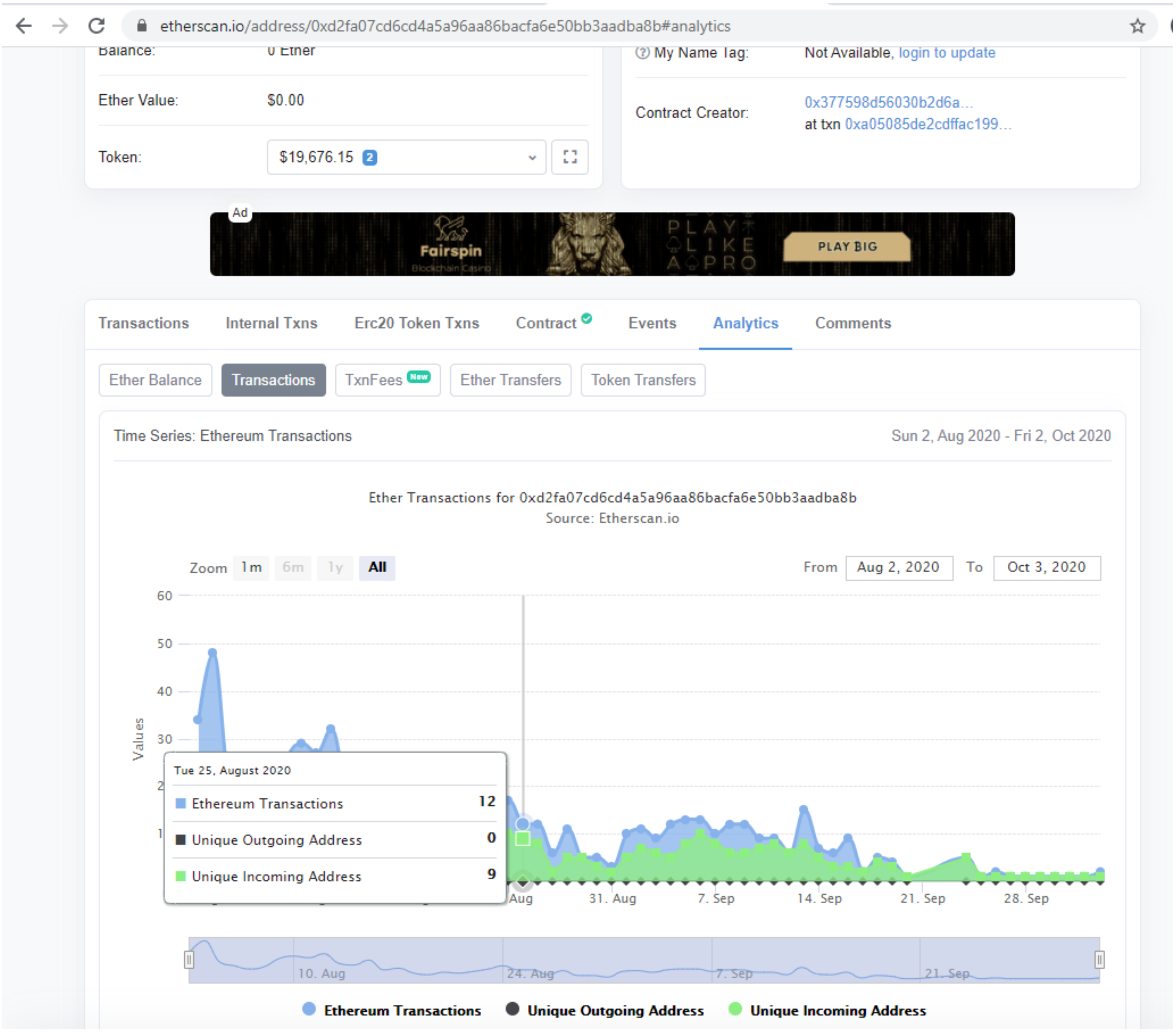
README.md

dForce (DF) Liquidity Mining

Mainnet Contract Address(2020-08-24)

Contract Name	Contract Address
USDC-USDx Staking	0xa94E2074BeB6D1Bf28014b81Ff2062eaB3600c48
USDx-DF Staking	0x5e84fC41D3aDd07A34616F781DCF1e49e8DC41C1
ETH-DF Staking	0x308777dDEC61F5000D8394626d55dbB0312fe874
GOLDx-DF Staking	0xdC7A844a45Ef936497FB916f1c2Ddb80F59a8aDc
dDAI Staking	0xD2fA07cD6Cd4A5A96aa86BacfA6E50bB3aaDBA8B
dUSDC Staking	0xB71dEFDd6240c45746EC58314a01dd6D833fD3b5
dUSDT Staking	0x324EebDAa45829c6A8eE903aFBc7B61AF48538df

Code Used Appendix



Balance: 0 Ether

Ether Value: $0.00

Token: $19,676.15 2

My Name Tag: Not Available, login to update

Contract Creator: 0x377598d56030b2d6a... at txn 0xa05085de2cdffac199...

Time Series: Ethereum Transactions — Sun 2, Aug 2020 - Fri 2, Oct 2020

Ether Transactions for 0xd2fa07cd6cd4a5a96aa86bacfa6e50bb3aadba8b
Source: Etherscan.io

Tue 25, August 2020
Ethereum Transactions 12
Unique Outgoing Address 0
Unique Incoming Address 9

Example Code Appendix

```
contract IRewardDistributionRecipient is Ownable {
    address rewardDistribution;

    function notifyRewardAmount(uint256 reward) external;

    modifier onlyRewardDistribution() {
        require(_msgSender() == rewardDistribution, "Caller is not reward d
        _;
    }

    function setRewardDistribution(address _rewardDistribution)
        external
        onlyOwner
    {
        rewardDistribution = _rewardDistribution;
    }
}
```

```


contract LPTokenWrapper {

    using SafeMath for uint256;
    using SafeERC20 for IERC20;

    IERC20 public lp = IERC20(0x460067f15e9B461a5F4c482E80217A2F45269385);

    uint256 private _totalSupply;
    mapping(address => uint256) private _balances;

    function totalSupply() public view returns(uint256) {
        return _totalSupply;
    }

    function balanceOf(address account) public view returns(uint256) {
        return _balances[account];
    }

    function stake(uint256 amount) public {
        _totalSupply = _totalSupply.add(amount);
        _balances[msg.sender] = _balances[msg.sender].add(amount);
        lp.safeTransferFrom(msg.sender, address(this), amount);
    }

    function withdraw(uint256 amount) public {
        _totalSupply = _totalSupply.sub(amount);
        _balances[msg.sender] = _balances[msg.sender].sub(amount);
        lp.safeTransfer(msg.sender, amount);
    }
}


contract Unipool is LPTokenWrapper, IRewardDistributionRecipient {

    IERC20 public df = IERC20(0x431ad2ff6a9C365805eBaD47Ee021148d6f7DBe0);

    uint256 public constant DURATION = 7 days;

    uint256 public periodFinish = 0;
    uint256 public rewardRate = 0;
    uint256 public lastUpdateTime;
    uint256 public rewardPerTokenStored;
    mapping(address => uint256) public userRewardPerTokenPaid;
    mapping(address => uint256) public rewards;

    event RewardAdded(uint256 reward);
    event Staked(address indexed user, uint256 amount);
    event Withdrawn(address indexed user, uint256 amount);
    event RewardPaid(address indexed user, uint256 reward);

    modifier updateReward(address account) {
        rewardPerTokenStored = rewardPerToken();
        lastUpdateTime = lastTimeRewardApplicable();
        if (account != address(0)) {
            rewards[account] = earned(account);
            userRewardPerTokenPaid[account] = rewardPerTokenStored;
        }
        _;
    }

    function lastTimeRewardApplicable() public view returns(uint256) {
        return Math.min(block.timestamp, periodFinish);
    }

    function rewardPerToken() public view returns(uint256) {
        if (totalSupply() == 0) {
            return rewardPerTokenStored;
        }
        return rewardPerTokenStored.add(
            lastTimeRewardApplicable().sub(lastUpdateTime).mul(rewardRate).
        );
    }

    function earned(address account) public view returns(uint256) {
        return balanceOf(account).mul(
            rewardPerToken().sub(userRewardPerTokenPaid[account])
        ).div(1e18).add(rewards[account]);
    }

    function stake(uint256 amount) public updateReward(msg.sender) {
        require(amount > 0, "Cannot stake 0");
        super.stake(amount);
        emit Staked(msg.sender, amount);
    }

    function withdraw(uint256 amount) public updateReward(msg.sender) {
        require(amount > 0, "Cannot withdraw 0");
        super.withdraw(amount);
        emit Withdrawn(msg.sender, amount);
    }

    function exit() public {
        withdraw(balanceOf(msg.sender));
        getReward();
    }

    function getReward() public updateReward(msg.sender) {
        uint256 reward = earned(msg.sender);
        if (reward > 0) {
            rewards[msg.sender] = 0;
            df.safeTransfer(msg.sender, reward);
            emit RewardPaid(msg.sender, reward);
        }
    }

    function notifyRewardAmount(uint256 reward) external onlyRewardDistribu
        if (block.timestamp >= periodFinish) {
            rewardRate = reward.div(DURATION);
        } else {
            uint256 remaining = periodFinish.sub(block.timestamp);
            uint256 leftover = remaining.mul(rewardRate);
            rewardRate = reward.add(leftover).div(DURATION);
        }
        lastUpdateTime = block.timestamp;
        periodFinish = block.timestamp.add(DURATION);
        emit RewardAdded(reward);
    }

    function lockedDetails() external view returns (bool, uint256) {
        return (false, periodFinish);
    }
}

```

SLOC Appendix

Solidity Contracts

Language	Files	Lines	Blanks	Comments	Code	Complexity
Solidity	12	2984	418	541	2009	145

Comments to Code 541/ 2009 = 27%