0.7

UniSwap V2 Process Quality Review

Score : 86%

This is a UniSwap V2 Process Quality Audit completed on August 25, 2020. It was performed using the Process Audit process (version 0.5) and is documented here. The audit was performed by ShinkaRex of Caliburn Consulting. Check out our Telegram.

The final score of the audit is 86%, a great score. The breakdown of the scoring is in Scoring Appendix.

Summary of the Process

Very simply, the audit 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

Executing Code Verification

This section looks at the code deployed on the Mainnet that gets audited and its corresponding software repository. The document explaining these questions is here. This audit 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 address was easy to find. You just click on "Similar Contracts" in the documentation.

They are available at Address 0x5C69bEe701ef814a2B6a3EDD4B1652CB9cc5aA6f as indicated in the Appendix. This Audit only covers the contracts *UniSwapFactory and UniSwapV2Router02*. This contract was written on 4 May 2020.

Is the code actively being used? (%)

> Answer: 100%

Activity is in excess of 75,000 transactions a day, as indicated in the Appendix for the Router contract.

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

0x5C69bEe701ef814a2B6a3EDD4B1652CB9cc5aA6f and 0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D are the Etherscan verified contract address.

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

> Answer: 90%

New repository was easy to find. The releases in the GitHub wasn't very were well documented. Version 1.01 matched. For UniswapV2Router02_d, it was in the uniswap-periphery repo. This one did not have this version released. The release (which was called beta) had the earlier version of the router software. However, the up-to-date version was in the master branch. We will give it a score of 90% as the router was not perfectly documented.

Guidance:

100%	Code matches and Repository was clearly labelled
60 %	Code matches but no labelled repository. Repository was found manually
30%	Code does match perfectly and repository was found manually
0%	Matching Code could not be found

GitHub address : https://github.com/Uniswap/uniswap-v2-core

Deployed contracts in the following file;

UniSwapDeployed.rar 9KB
Binary

Matching Repository: https://github.com/Uniswap/uniswap-v2-core/releases/tag/v1.0.1

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%

Both the uniswap-v2-core and uniswap-interfaces were healthy repos with many commits.

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://uniswap.org/docs/v2

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

> ✓ Answer: Yes

Location: https://uniswap.org/docs/v2

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

> ✓ Answer: 100%

The documentation is quite complete, both describing the overall functions with good diagrams and in detail on each function for each contract.

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

> ⚠ Answer: 10%

There is very little commenting in the actual code. For this reason, you know score is unavoidable. As long as the documentation is used when reading the code, the situation is dramatically improved.

Code examples are in the Appendix. As per the SLOC, there is 3% 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: 60%

The documentation very clearly does trace to the code contract by contract and function by function describing the relevant details. However, with no commenting in the code to connect it is impossible to give 100%.

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%

A full typescript test suite is evident in the test directory.

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

> Answer: 50%

No indication of code coverage but clearly there is a reasonably complete set of tests

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

Using yarn, as indicated in the repo readme.

Packaged with the deployed code (Y/N)

> Answer: Yes

Report of the results (%)

> Answer: 0%

No report is evident for the typescript tests or code coverage.

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: 100%

Formal verification of some contracts was part of the audit by DApp.org; https://uniswap.org/audit.html.

Stress Testing environment (%)

> Answer: 100%

An active ropsten test net is evident at address 0x5C69bEe701ef814a2B6a3EDD4B1652CB9cc5aA6f.

Audits

> Answer: 90%

A single audit by DApp.org was performed; https://uniswap.org/audit.html.

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 (0%)

Appendices

Author Details

The author of this audit 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.

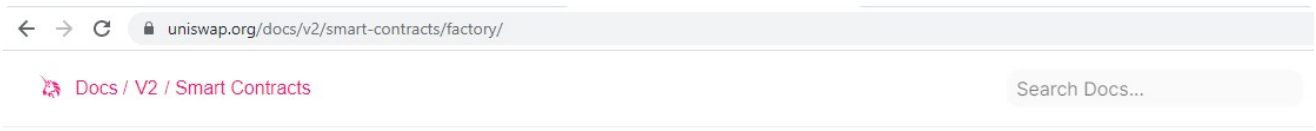
Process Quality Audits are an extension of the SecurEth guidelines that will further increase the quality processes in Solidity and Vyper development.

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

Scoring Appendix

	Total	UniSwap	
PQ Audit Scoring Matrix (v0.4 and 0.5)	**Points**	**Answer**	**Points**
Total	240		207.5
Executing Code Verification			**86%**
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	90%	18
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	100%	15
4. Are there sufficiently detailed comments for all functions within the deployed contract code (%)	10	10%	1
5. Is it possible to trace requirements to the implementation in code (%)	5	60%	3
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	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	100%	5
7. Stress Testing environment (%)	5	100%	5
Audits			
Audit done	70	90%	63
Section Scoring			
Executing Code Verification	70	97%	
Documentation	45	76%	
Testing	55	77%	
Audits	70	90%	

Executing Code Appendix



← Back

Smart Contracts
- Factory
- Pair
- Pair (ERC-20)
- Library
- Router01
- Router02

Smart Contracts

Factory

Improve this article

Code

`UniswapV2Factory.sol`

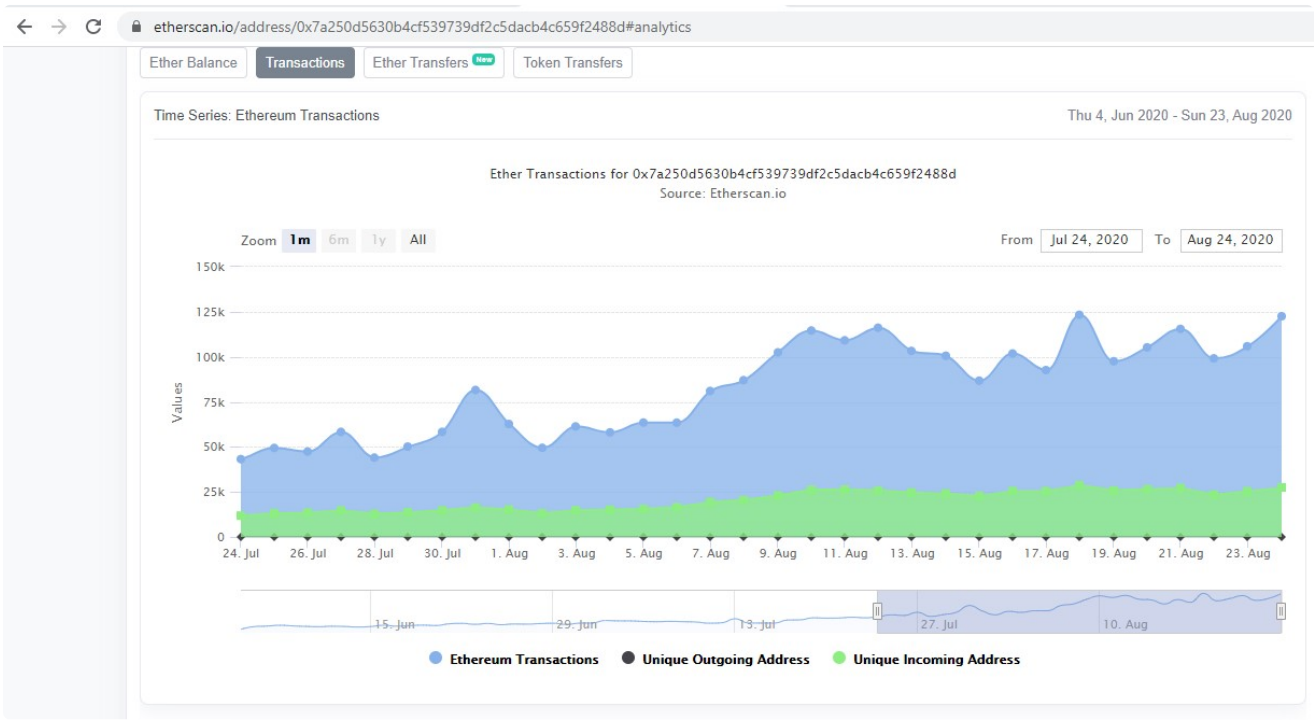
Address

`UniswapV2Factory` is deployed at `0x5C69bEe701ef814a2B6a3EDD4B1652CB9cc5aA6f` on the Ethereum mainnet, and the Ropsten, Rinkeby, Görli, and Kovan testnets. It was built from commit 8160750.

Events

PairCreated

Code Used Appendix



Example Code Appendix

```
contract UniswapV2Factory is IUniswapV2Factory {
    address public feeTo;
    address public feeToSetter;

    mapping(address => mapping(address => address)) public getPair;
    address[] public allPairs;

    event PairCreated(address indexed token0, address indexed token1, address pair, uint);

```

```
    constructor(address _feeToSetter) public {
        feeToSetter = _feeToSetter;
    }

    function allPairsLength() external view returns (uint) {
        return allPairs.length;
    }

    function createPair(address tokenA, address tokenB) external returns (address pair) {
        require(tokenA != tokenB, 'UniswapV2: IDENTICAL_ADDRESSES');
        (address token0, address token1) = tokenA < tokenB ? (tokenA, tokenB) : (tokenB, t
        require(token0 != address(0), 'UniswapV2: ZERO_ADDRESS');
        require(getPair[token0][token1] == address(0), 'UniswapV2: PAIR_EXISTS'); // singl
        bytes memory bytecode = type(UniswapV2Pair).creationCode;
        bytes32 salt = keccak256(abi.encodePacked(token0, token1));
        assembly {
            pair := create2(0, add(bytecode, 32), mload(bytecode), salt)
        }
        IUniswapV2Pair(pair).initialize(token0, token1);
        getPair[token0][token1] = pair;
        getPair[token1][token0] = pair; // populate mapping in the reverse direction
        allPairs.push(pair);
        emit PairCreated(token0, token1, pair, allPairs.length);
    }

    function setFeeTo(address _feeTo) external {
        require(msg.sender == feeToSetter, 'UniswapV2: FORBIDDEN');
        feeTo = _feeTo;
    }

    function setFeeToSetter(address _feeToSetter) external {
        require(msg.sender == feeToSetter, 'UniswapV2: FORBIDDEN');
        feeToSetter = _feeToSetter;
    }
}

```

SLOC Appendix

Solidity Contracts

Language	Files	Lines	Blanks	Comments	Code	Complex
Solidity	11	1223	108	29	1096	95

Comments to Code 29/ 1096 = 3%