0.7

Keep3r V2 0.7 UPDATE Process Quality Review

Score: 71%

Overview

This is Keep3r a Process Quality Review completed on 13/10/2021. It was performed using the Process Review process (version 0.7.3) and is documented here. The review was performed by Nick of DeFiSafety. Check out our Telegram. The previous version of this report is here.

The final score of the review is **71%,** a **PASS.** The breakdown of the scoring is in Scoring Appendix. For our purposes, a pass is **70%.**

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**
- **Here are the admin controls and strategies**

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.

Chain

This section indicates the blockchains used by this protocol. This report covers all of the blockchains upon which the protocol is deployed.

> ✓ **Chain:** Ethereum, Binance Smart Chain, Polygon, Fantom

Guidance:

Ethereum
Binance Smart Chain
Polygon
Avalanche
Terra
Celo
Arbitrum
Solana

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 following questions:

1) Are the executing code addresses readily available? (%)
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)

1) Are the executing code addresses readily available? (%)

> ✓ **Answer:** 100%

They are available at website https://docs.keep3r.network/registry, as indicated in the Appendix.

Guidance:

100% Clearly labelled and on website, docs or repo, quick to find
70% Clearly labelled and on website, docs or repo but takes a bit of looking
40% Addresses in mainnet.json, in discord or sub graph, etc
20% Address found but labeling not clear or easy to find
0% Executing addresses could not be found

2) Is the code actively being used? (%)

Answer: 70%

Activity is more than 10 transactions a week on contract *Keep3rLiquiditManagerJob*, as indicated in the Appendix.

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

3) Is there a public software repository? (Y/N)

Answer: Yes

GitHub: https://github.com/keep3r-network/keep3r.network

Is there a public software repository with the code at a minimum, but also normally test and scripts. Even if the repository was created just to hold the files and has just 1 transaction, it gets a **"Yes"**. For teams with private repositories, this answer is **"No"**.

4) Is there a development history visible? (%)

Answer: 100%

At 172 commits, Andre Cronje defends his reputation as one of the most prolific builders in the DeFi space by demonstrating an unfailing commitment to development history.

This metric 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 30 commits

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

> ✓ **Answer:** Yes

Location: https://github.com/keep3r-network/keep3r.network/graphs/contributors

For a **"Yes"** in this question, the real names of some team members must be public on the website or other documentation (LinkedIn, etc). If the team is anonymous, then this question is a **"No"**.

Documentation

This section looks at the software documentation. The document explaining these questions is here.

Required questions are;

6) Is there a whitepaper? (Y/N)
7) Are the basic software functions documented? (Y/N)
8) Does the software function documentation fully (100%) cover the deployed contracts? (%)
9) Are there sufficiently detailed comments for all functions within the deployed contract code (%)
10) Is it possible to trace from software documentation to the implementation in
code (%)

6) Is there a whitepaper? (Y/N)

> ✓ **Answer:** Yes

Location: https://docs.keep3r.network/

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

> ✓ **Answer:** Yes

Basic software functions are identified in the docs.

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

> **Answer:** 80%

The major contracts are covered, but more elaboration is required to reach full coverage in the 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 be improved by adding content to the software functions 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.

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

> **Answer:** 48%

Code examples are in the Appendix. As per the SLOC, there is 48% 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.

10) Is it possible to trace from software documentation to the implementation in code (%)

> **Answer:** 60%

Keep3r's docs have clear association with its code, though there is no explicit 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 documentation to code such that it is clear where each outlined function is coded in the source code. 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;

11) Full test suite (Covers all the deployed code) (%)
12) Code coverage (Covers all the deployed lines of code, or explains misses) (%)
13) Scripts and instructions to run the tests (Y/N)
14) Report of the results (%)
15) Formal Verification test done (%)
16) Stress Testing environment (%)

11) Is there a Full test suite? (%)

> **Answer:** 40%

Code examples are in the Appendix. As per the SLOC, there is 26% testing to code (TtC).

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

How to improve this score:

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

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

Answer: 30%

No code coverage testing was found, though some testing been conducted on Keep3r.

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 improved by adding tests that achieve full code coverage. A clear report and scripts in the software repository will guarantee a high score.

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

Answer: Yes

Scripts: https://github.com/keep3r-network/keep3r.network/tree/master/scripts.

14) Report of the results (%)

Answer: 0%

No test report was found.

Guidance:

100%	Detailed test report as described below
70%	GitHub code coverage report visible
0%	No test report evident

How to improve this score

Add a report with the results. The test scripts should generate the report or elements of it.

15) Formal Verification test done (%)

> ⚠ **Answer:** 0%

Formal verification testing could not be found.

16) Stress Testing environment (%)

> ✓ **Answer:** 100%

Keep3r is deployed to Ropsten, Kovan and Rinkeby testnets.

Security

This section looks at the 3rd party software audits done. It is explained in this document. This section answers the following questions;

17) Did 3rd Party audits take place? (%)
18) Is the bounty value acceptably high?

17) Did 3rd Party audits take place? (%)

> ✓ **Answer:** 100%

Keep3r V1 has been audited pre-launch, as well as the newer v3 of their pools and their new OLM. Most of the fix recommendations have been implemented by the team.

Guidance:

100% Multiple Audits performed before deployment and results public and implemented or not required
90% Single audit performed before deployment and results public and implemented or not required
70% Audit(s) performed after deployment and no changes required. Audit report is public

50% Audit(s) performed after deployment and changes needed but not implemented
20% No audit performed

0% Audit Performed after deployment, existence is public, report is not public and
no improvements deployed OR smart contract address' not found, (where question 1 is 0%)

Deduct 25% if code is in a private repo and no note from auditors that audit is applicable to deployed code

18) Is the bounty value acceptably high (%)

> ⚠ **Answer:** 0%

No bug bounty information was found.

Guidance:

100% Bounty is 10% TVL or at least $1M AND active program (see below)
90% Bounty is 5% TVL or at least 500k AND active program
80% Bounty is 5% TVL or at least 500k
70% Bounty is 100k or over AND active program
60% Bounty is 100k or over
50% Bounty is 50k or over AND active program
40% Bounty is 50k or over
20% Bug bounty program bounty is less than 50k
0% No bug bounty program offered

An active program means that a third party (such as Immunefi) is actively driving hackers to the site. An inactive program would be static mentions on the docs.

Access Controls

This section covers the documentation of special access controls for a DeFi protocol. The admin access controls are the contracts that allow updating contracts or coefficients in the protocol. Since these contracts can allow the protocol admins to "change the rules", complete disclosure of capabilities is vital for user's transparency. It is explained in this document. The questions this section asks are as follow;

19) Can a user clearly and quickly find the status of the admin controls?
20) Is the information clear and complete?
21) Is the information in non-technical terms that pertain to the investments?
22) Is there Pause Control documentation including records of tests?

19) Can a user clearly and quickly find the status of the access controls (%)

> ✓ **Answer:** 100%

Access control information relating to the jobs themselves is clearly labelled in the docs.

Guidance:

100% Clearly labelled and on website, docs or repo, quick to find
70% Clearly labelled and on website, docs or repo but takes a bit of looking
40% Access control docs in multiple places and not well labelled
20% Access control docs in multiple places and not labelled
0% Admin Control information could not be found

20) Is the information clear and complete (%)

> **Answer:** 30%

a) All contracts are clearly labelled as upgradeable (or not) -- 15% -- Voting insinuates upgradeability, but there is no additional information.

b) The type of ownership is clearly indicated (OnlyOwner / MultiSig / Defined Roles) -- 15% -- Defined roles are identified, but are vague. More information needed.

c) No real information on capabilities for change in the contracts.

Guidance:

All the contracts are immutable -- 100% OR

a) All contracts are clearly labelled as upgradeable (or not) -- 30% AND
b) The type of ownership is clearly indicated (OnlyOwner / MultiSig / Defined Roles) -- 30% AND
c) The capabilities for change in the contracts are described -- 30%

How to improve this score:

Create a document that covers the items described above. An example is enclosed.

21) Is the information in non-technical terms that pertain to the investments (%)

> **Answer:** 30%

The limited information is in software specific language.

Guidance:

100% All the contracts are immutable
90% Description relates to investments safety and updates in clear, complete non-software l language
30% Description all in software specific language
0% No admin control information could not be found

How to improve this score:

Create a document that covers the items described above in plain language that investors can understand. An example is enclosed.

22) Is there Pause Control documentation including records of tests (%)

> ⚠ **Answer:** 0%

No pause control information was found.

Guidance:

100% All the contracts are immutable or no pause control needed and this is explained OR
100% Pause control(s) are clearly documented and there is records of at least one test within 3 months

80% Pause control(s) explained clearly but no evidence of regular tests
40% Pause controls mentioned with no detail on capability or tests
0% Pause control not documented or explained

How to improve this score**:**

Create a document that covers the items described above in plain language that investors can understand. An example is enclosed.

Appendices

Author Details

The author of this review is Rex of DeFi Safety.

Email : rex@defisafety.com Twitter : @defisafety

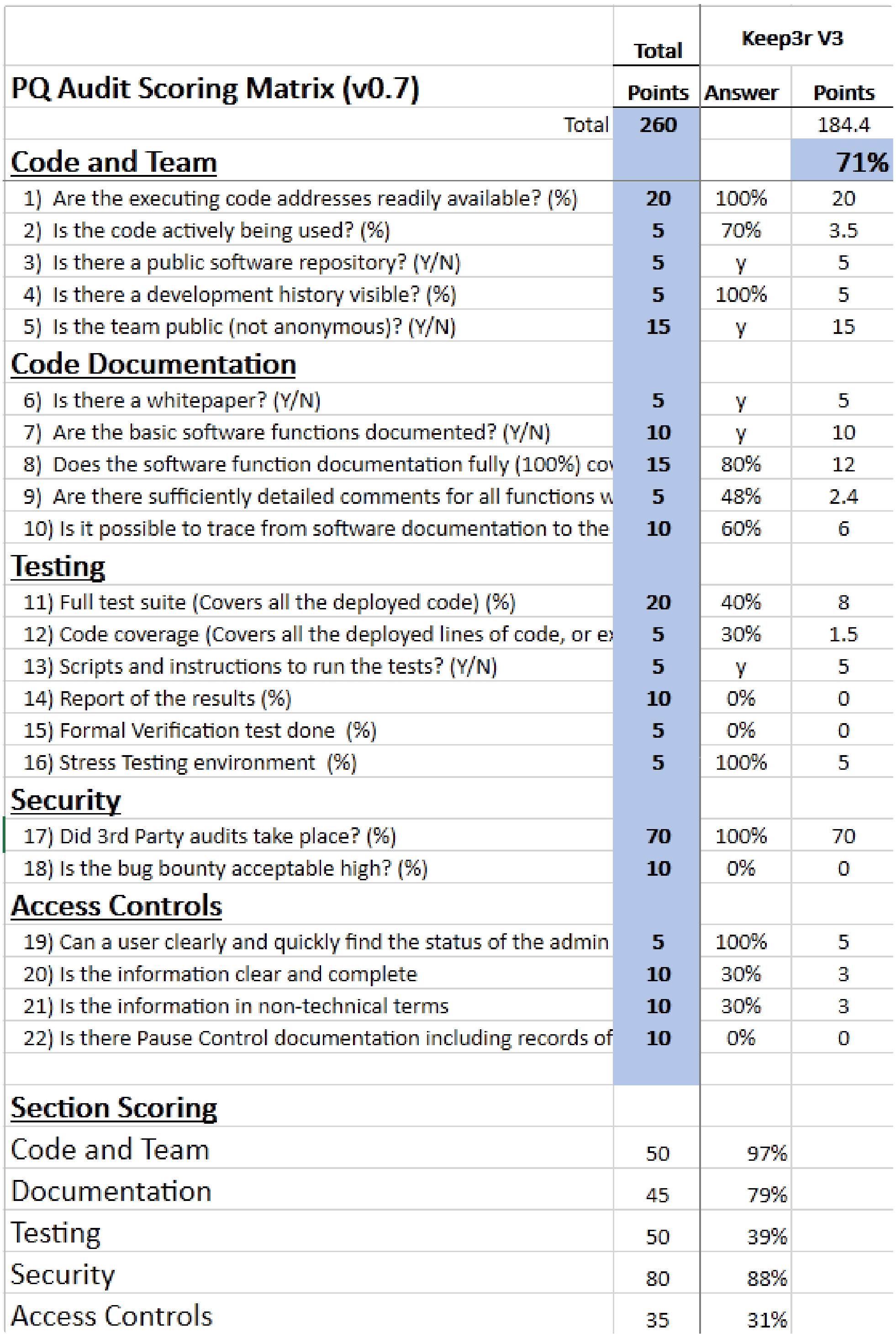
I started with Ethereum just before the DAO and that was a wonderful education. It showed the importance of code quality. The second Parity hack also showed the importance of good process. Here my aviation background offers some value. Aerospace knows how to make reliable code using quality processes.

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.

DeFiSafety is my full time gig and we are working on funding vehicles for a permanent staff.

Scoring Appendix



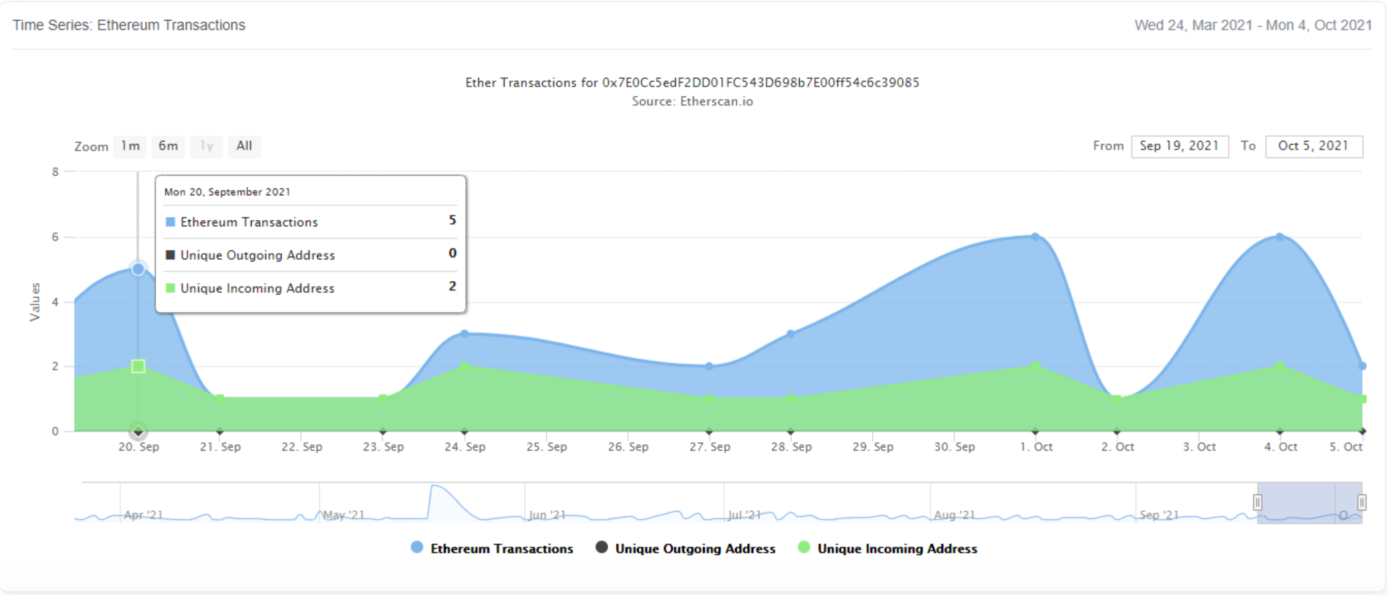
	Total	Keep3r V3	
PQ Audit Scoring Matrix (v0.7)	**Points**	**Answer**	**Points**
Total	**260**		184.4
Code and Team			**71%**
1) Are the executing code addresses readily available? (%)	**20**	100%	20
2) Is the code actively being used? (%)	**5**	70%	3.5
3) Is there a public software repository? (Y/N)	**5**	y	5
4) Is there a development history visible? (%)	**5**	100%	5
5) Is the team public (not anonymous)? (Y/N)	**15**	y	15
Code Documentation			
6) Is there a whitepaper? (Y/N)	**5**	y	5
7) Are the basic software functions documented? (Y/N)	**10**	y	10
8) Does the software function documentation fully (100%) cov	**15**	80%	12
9) Are there sufficiently detailed comments for all functions w	**5**	48%	2.4
10) Is it possible to trace from software documentation to the	**10**	60%	6
Testing			
11) Full test suite (Covers all the deployed code) (%)	**20**	40%	8
12) Code coverage (Covers all the deployed lines of code, or ex	**5**	30%	1.5
13) Scripts and instructions to run the tests? (Y/N)	**5**	y	5
14) Report of the results (%)	**10**	0%	0
15) Formal Verification test done (%)	**5**	0%	0
16) Stress Testing environment (%)	**5**	100%	5
Security			
17) Did 3rd Party audits take place? (%)	**70**	100%	70
18) Is the bug bounty acceptable high? (%)	**10**	0%	0
Access Controls			
19) Can a user clearly and quickly find the status of the admin	**5**	100%	5
20) Is the information clear and complete	**10**	30%	3
21) Is the information in non-technical terms	**10**	30%	3
22) Is there Pause Control documentation including records of	**10**	0%	0
Section Scoring			
Code and Team	50	97%	
Documentation	45	79%	
Testing	50	39%	
Security	80	88%	
Access Controls	35	31%	

Executing Code Appendix

Job	Address
HegicPoolKeep3r	0x5DDe926b0A31346f2485900C5e64c2577F43F774
YearnV1EarnKeep3r	0xe7F4ab593aeC81EcA754Da1B3B7cE0C42a13Ec0C
MetaKeep3r	0x93Dfa873b15ad496BA8116Ce6CfEC52eF30a9372

DforceStrategyKeep3r	0x30084324619D9645019C3f2cb3a94611601a3078
HegicKeep3rV2	0xB1aCE96072654e3A2564A90D64Be99Dd3Ac195F4
MMStrategyKeeperV1	0x4E504c6ca43cD1bBd9096A2c2E77A176D10910B1
LidoKeep3r	0x1EE5C83C4B43aaEd21613D5cc7835D36078ce03F
HegicBotKeep3r	0x6b405609B78241112a3030E8a85570F06fbd3aca
BzxLiquidateProxy	0xB59A6dCE95bc446aD098B4C4b415bbe766068cb8
YearnLiquidationKeep3r	0xf35eE77197b8E222549A54D7A43fc4DC60eBbeeb
YearnV1EarnKeep3rV2	0xF8106d779246612FF7a6A623EF7026a9ccFaf709
Generic Keep3r for Mushrooms Finance	0x0bD1d668d8E83d14252F2e01D5873df77A6511f0
YearnTendV2Keep3rJob	0x7b28163e7a3db17eF2dba02BCf7250A8Dc505057
PartialKeep3rV1OracleJob	0x5efD850044Ba76b8ffE49437CB301be3568bA696
Keep3rLiquidityManagerJob	0x7E0Cc5edF2DD01FC543D698b7E00ff54c6c39085
HarvestV2Keep3rJob	0x620bd1E1D1d845c8904aC03F6cd6b87706B7596b
CrvStrategyKeep3rJob	0x02027bDA2425204f152B8aa35Fb78687D65E1AF5
Keep3rV2OracleFactoryV2	0xaed599AADfEE8e32Cedb59db2b1120d33A7bACFD

Code Used Appendix



Time Series: Ethereum Transactions

Wed 24, Mar 2021 - Mon 4, Oct 2021

Ether Transactions for 0x7E0Cc5edF2DD01FC543D698b7E00ff54c6c39085

Source: Etherscan.io

Example Code Appendix

```
// From https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/math/
// Subject to the MIT license.

/**
 * @dev Wrappers over Solidity's arithmetic operations with added overflow
 * checks.
 *
 * Arithmetic operations in Solidity wrap on overflow. This can easily result
 * in bugs, because programmers usually assume that an overflow raises an
 * error, which is the standard behavior in high level programming languages.
 * `SafeMath` restores this intuition by reverting the transaction when an
 * operation overflows.
 *
 * Using this library instead of the unchecked operations eliminates an entire
 * class of bugs, so it's recommended to use it always.
 */
library SafeMath {
    /**
     * @dev Returns the addition of two unsigned integers, reverting on overflow.
     *
     * Counterpart to Solidity's `+` operator.
     *
     * Requirements:
     * - Addition cannot overflow.
     */
    function add(uint a, uint b) internal pure returns (uint) {
        uint c = a + b;
        require(c >= a, "add: +");

        return c;
    }

    /**
     * @dev Returns the addition of two unsigned integers, reverting with custom message o
     *
     * Counterpart to Solidity's `+` operator.
     *
     * Requirements:
     * - Addition cannot overflow.
     */
    function add(uint a, uint b, string memory errorMessage) internal pure returns (uint)
        uint c = a + b;
        require(c >= a, errorMessage);

        return c;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting on underflow (when
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
```

```
     * - Subtraction cannot underflow.
     */
    function sub(uint a, uint b) internal pure returns (uint) {
        return sub(a, b, "sub: -");
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting with custom messag
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     * - Subtraction cannot underflow.
     */
    function sub(uint a, uint b, string memory errorMessage) internal pure returns (uint)
        require(b <= a, errorMessage);
        uint c = a - b;

        return c;
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, reverting on overflow.
     *
     * Counterpart to Solidity's `*` operator.
     *
     * Requirements:
     * - Multiplication cannot overflow.
     */
    function mul(uint a, uint b) internal pure returns (uint) {
        // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
        // benefit is lost if 'b' is also tested.
        // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
        if (a == 0) {
            return 0;
        }

        uint c = a * b;
        require(c / a == b, "mul: *");

        return c;
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, reverting on overflow.
     *
     * Counterpart to Solidity's `*` operator.
     *
     * Requirements:
     * - Multiplication cannot overflow.
     */
    function mul(uint a, uint b, string memory errorMessage) internal pure returns (uint)
        // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
        // benefit is lost if 'b' is also tested.
        // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
        if (a == 0) {
            return 0;
        }

        uint c = a * b;
        require(c / a == b, errorMessage);

        return c;
    }

    /**
     * @dev Returns the integer division of two unsigned integers.
     * Reverts on division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     */
    function div(uint a, uint b) internal pure returns (uint) {
        return div(a, b, "div: /");
    }

    /**
     * @dev Returns the integer division of two unsigned integers.
     * Reverts with custom message on division by zero. The result is rounded towards zero
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     */
    function div(uint a, uint b, string memory errorMessage) internal pure returns (uint)
        // Solidity only automatically asserts when dividing by 0
        require(b > 0, errorMessage);
        uint c = a / b;
        // assert(a == b * c + a % b); // There is no case in which this doesn't hold

        return c;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modu
     * Reverts when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an

     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     */
    function mod(uint a, uint b) internal pure returns (uint) {
        return mod(a, b, "mod: %");
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modu
     * Reverts with custom message when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     */
    function mod(uint a, uint b, string memory errorMessage) internal pure returns (uint) {
        require(b != 0, errorMessage);
        return a % b;
    }
}

/**
 * @dev Contract module that helps prevent reentrant calls to a function.
 *
 * Inheriting from `ReentrancyGuard` will make the {nonReentrant} modifier
 * available, which can be applied to functions to make sure there are no nested
 * (reentrant) calls to them.
 *
 * Note that because there is a single `nonReentrant` guard, functions marked as
 * `nonReentrant` may not call one another. This can be worked around by making
 * those functions `private`, and then adding `external` `nonReentrant` entry
 * points to them.
 *
 * TIP: If you would like to learn more about reentrancy and alternative ways
 * to protect against it, check out our blog post
 * https://blog.openzeppelin.com/reentrancy-after-istanbul/[Reentrancy After Istanbul].
 */
contract ReentrancyGuard {
    // Booleans are more expensive than uint256 or any type that takes up a full
    // word because each write operation emits an extra SLOAD to first read the
    // slot's contents, replace the bits taken up by the boolean, and then write
    // back. This is the compiler's defense against contract upgrades and
    // pointer aliasing, and it cannot be disabled.

    // The values being non-zero value makes deployment a bit more expensive,
    // but in exchange the refund on every call to nonReentrant will be lower in
    // amount. Since refunds are capped to a percentage of the total
    // amount. Since refunds are capped to a percentage of the total
    // transaction's gas, it is best to keep them low in cases like this one, to

    // increase the likelihood of the full refund coming into effect.
    uint256 private constant _NOT_ENTERED = 1;
    uint256 private constant _ENTERED = 2;

    uint256 private _status;

    constructor () internal {
        _status = _NOT_ENTERED;
    }

    /**
     * @dev Prevents a contract from calling itself, directly or indirectly.
     * Calling a `nonReentrant` function from another `nonReentrant`
     * function is not supported. It is possible to prevent this from happening
     * by making the `nonReentrant` function external, and make it call a
     * `private` function that does the actual work.
     */
    modifier nonReentrant() {
        // On the first call to nonReentrant, _notEntered will be true
        require(_status != _ENTERED, "ReentrancyGuard: reentrant call");

        // Any calls to nonReentrant after this point will fail
        _status = _ENTERED;

        _;

        // By storing the original value once again, a refund is triggered (see
        // https://eips.ethereum.org/EIPS/eip-2200)
        _status = _NOT_ENTERED;
    }
}
```

SLOC Appendix

Solidity Contracts

Language	Files	Lines	Blanks	Comments	Code	Complex
Solidity	11	3360	480	935	1945	267

Comments to Code 935/1945 = 48%

Solidity Tests

Language	Files	Lines	Blanks	Comments	Code	Complex

Solidity	6	679	125	49	505	46

Tests to Code 505/1945 = 26%