BarnBridge Process Quality Review

Score: 91%

This is a Process Quality Review of BarnBridge completed on April 9, 2021. It was performed using the Process Review process (version 0.6.2) and is documented here. The review was performed by ShinkaRex of DeFiSafety. Check out our Telegram.

The final score of the review is 91%, an awesome score. 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**

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.

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 questions;

1. Are the executing code addresses readily available? (Y/N)
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)

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

✓ Answer: Yes

BarnBridge has published some their yield farming smart contract addresses, but they have not published the addresses for their other repositories found in their github (BarnBridge-SmartYieldBonds,BarnBridge-Barn,BarnBridge-DAO)

They are available at website https://github.com/BarnBridge/BarnBridge-YieldFarming as indicated in the Appendix.

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 addresses 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 *25* transactions a day on contract *yeildfarm.sol*, 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

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

> ✓ Answer: Yes

GitHub: https://github.com/BarnBridge

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

Is there a development history visible? (%)

> ✓ Answer: 100%

With 153 commits and 17 branches, this is clearly a healthy repository.

This 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 10 commits

How to improve this score

Continue to test and perform other verification activities after deployment, including routine maintenance updating to new releases of testing and deployment tools. A public development history indicates clearly to the public the level of continued investment and activity by the developers on the application. This gives a level of security and faith in the application.

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

✓ Answer: Yes

The names of the team members can be found on their about page.

For a yes in this question the real names of some team members must be public on the website or other documentation. If the team is anonymous and then this question is a No.

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 software functions documented? (Y/N)
3. Does the software function documentation 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 from software documentation to the implementation in codee (%)

Is there a whitepaper? (Y/N)

> ✓ Answer: Yes

Location: https://github.com/BarnBridge/BarnBridge-Whitepaper

How to improve this score

Ensure the white paper is available for download from your website or at least the software repository. Ideally update the whitepaper to meet the capabilities of your present application.

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

> ✓ Answer: Yes

Some of the Software functions for the BarnBridge_Barn are explained in their github repository. Barnbridge additionally has a spec.md file for their BarnBridge-SmartYeildBonds contracts.

How to improve this score

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

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

contracts? (%)

> ✓ Answer: 70%

Testable specs exist for their software. While there is not much dedicated software documentation, specs are an excellent base for tests and transparency,

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 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%

Code examples are in the Appendix. As per the SLOC, there is 19% commenting to code (CtC). But the comment quality is good, so a score of 40% is given. No NatSpec keywords seen.

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.

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

Answer: 40%

The documentation lists some of the functions and describes their functions.

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%

With a TtC ratio of 172%, this is clearly a full test suite.

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 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: 95%

There is no indication of code coverage in the deployed docs, but it is indicated in the Quantstamp audit, but there is clearly 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

Packaged with the deployed code (Y/N)

> ✓ Answer: Yes

Report of the results (%)

> ✓ Answer: 80%

Test report (https://github.com/BarnBridge/BarnBridge-Barn/blob/master/test-results.md) is quite complete. Reasons for code coverage misses not included.

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.

Formal Verification test done (%)

> ⚠ Answer: 0%

There is no evidence of any formal verification testing having been done.

Stress Testing environment (%)

> ✓ Answer: 100%

BarnBridge Testnet sson Kovan.
- Kovan tokens available: BOND, USDC, DAI, USDT
- Use the faucets to get tokens

https://testnet.app.barnbridge.com/

Audits

> ✓ Answer: 100%

Hacken did a review on September 29th, 2020. The is a Quantstamp audit also.

BarnBridge was first released on November 17th, 2020.

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 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

Executing Code Appendix

Code Used Appendix

Example Code Appendix

```
// SPDX-License-Identifier: Apache-2.0
pragma solidity 0.7.6;
pragma experimental ABIEncoderV2;

import "../libraries/LibDiamondStorage.sol";
import "../interfaces/IDiamondLoupe.sol";
import "../interfaces/IERC165.sol";

contract DiamondLoupeFacet is IDiamondLoupe, IERC165 {
    // Diamond Loupe Functions
    ////////////////////////////////////////////////////////////////////
    /// These functions are expected to be called frequently by tools.
    //
    // struct Facet {
    //     address facetAddress;
    //     bytes4[] functionSelectors;
    // }
    /// @notice Gets all facets and their selectors.
    /// @return facets_ Facet
    function facets() external override view returns (Facet[] memory facets_
        LibDiamondStorage.DiamondStorage storage ds = LibDiamondStorage.diar
        uint256 selectorCount = ds.selectors.length;

        // create an array set to the maximum size possible
        facets_ = new Facet[](selectorCount);

        // create an array for counting the number of selectors for each fa
        uint8[] memory numFacetSelectors = new uint8[](selectorCount);

        // total number of facets
        uint256 numFacets;

        // loop through function selectors
        for (uint256 selectorIndex; selectorIndex < selectorCount; selectorI
            bytes4 selector = ds.selectors[selectorIndex];
            address facetAddress_ = ds.facets[selector].facetAddress;
            bool continueLoop = false;

            // find the functionSelectors array for selector and add select
            for (uint256 facetIndex; facetIndex < numFacets; facetIndex++) 
                if (facets_[facetIndex].facetAddress == facetAddress_) {
                    facets_[facetIndex].functionSelectors[numFacetSelectors
                    // probably will never have more than 256 functions fror
                    require(numFacetSelectors[facetIndex] < 255);
                    numFacetSelectors[facetIndex]++;
                    continueLoop = true;
                    break;
                }
            }

            // if functionSelectors array exists for selector then continue
            if (continueLoop) {
                continueLoop = false;
                continue;
            }

            // create a new functionSelectors array for selector
            facets_[numFacets].facetAddress = facetAddress_;
            facets_[numFacets].functionSelectors = new bytes4[](selectorCou
            facets_[numFacets].functionSelectors[0] = selector;

            numFacetSelectors[numFacets] = 1;
            numFacets++;
        }

        for (uint256 facetIndex; facetIndex < numFacets; facetIndex++) {
            uint256 numSelectors = numFacetSelectors[facetIndex];
            bytes4[] memory selectors = facets_[facetIndex].functionSelecto
            // setting the number of selectors
            assembly {
                mstore(selectors, numSelectors)
            }
        }

        // setting the number of facets
        assembly {
            mstore(facets_, numFacets)
        }
    }

    /// @notice Gets all the function selectors supported by a specific fac
    /// @param _facet The facet address.
    /// @return _facetFunctionSelectors The selectors associated with a fac
    function facetFunctionSelectors(address _facet) external override view 
        LibDiamondStorage.DiamondStorage storage ds = LibDiamondStorage.dia

        uint256 selectorCount = ds.selectors.length;
        uint256 numSelectors;
        _facetFunctionSelectors = new bytes4[](selectorCount);

        // loop through function selectors
        for (uint256 selectorIndex; selectorIndex < selectorCount; selector
            bytes4 selector = ds.selectors[selectorIndex];
            address facetAddress_ = ds.facets[selector].facetAddress;
            if (_facet == facetAddress_) {
                _facetFunctionSelectors[numSelectors] = selector;
                numSelectors++;
            }
        }

        // Set the number of selectors in the array
        assembly {
            mstore(_facetFunctionSelectors, numSelectors)
        }
    }

    /// @notice Get all the facet addresses used by a diamond.
    /// @return facetAddresses_
    function facetAddresses() external override view returns (address[] memo
        LibDiamondStorage.DiamondStorage storage ds = LibDiamondStorage.diam

        uint256 selectorCount = ds.selectors.length;
        // create an array set to the maximum size possible
        facetAddresses_ = new address[](selectorCount);
        uint256 numFacets;

        // loop through function selectors
        for (uint256 selectorIndex; selectorIndex < selectorCount; selectorI
            bytes4 selector = ds.selectors[selectorIndex];
            address facetAddress_ = ds.facets[selector].facetAddress;
            bool continueLoop = false;

            // see if we have collected the address already and break out of
            for (uint256 facetIndex; facetIndex < numFacets; facetIndex++) {
                if (facetAddress_ == facetAddresses_[facetIndex]) {
                    continueLoop = true;
                    break;
                }
            }

            // continue loop if we already have the address
            if (continueLoop) {
                continueLoop = false;
                continue;
            }

            // include address
            facetAddresses_[numFacets] = facetAddress_;
            numFacets++;
        }

        // Set the number of facet addresses in the array
        assembly {
            mstore(facetAddresses_, numFacets)
        }
    }

    /// @notice Gets the facet address that supports the given selector.
    /// @dev If facet is not found return address(0).
    /// @param _functionSelector The function selector.
    /// @return facetAddress_ The facet address.
    function facetAddress(bytes4 _functionSelector) external override view r
        LibDiamondStorage.DiamondStorage storage ds = LibDiamondStorage.diam
        facetAddress_ = ds.facets[_functionSelector].facetAddress;
    }

    // This implements ERC-165.
    function supportsInterface(bytes4 _interfaceId) external override view r
        LibDiamondStorage.DiamondStorage storage ds = LibDiamondStorage.diam
        return ds.supportedInterfaces[_interfaceId];
    }
}

```

SLOC Appendix

Solidity Contracts

Language	Files	Lines	Blanks	Comments	Code	Complexity
Solidity	32	4421	905	576	2940	335

Comments to Code 576/2940 = 19%

Javascript Tests

Language	Files	Lines	Blanks	Comments	Code	Complexity
JavaScript	36	6827	1653	93	5081	

Tests to Code 5081/2940 = 172%