0.7

Badger Finance 0.7 Process Quality Review

Score: 85%

Overview

This is a Badger Finance Process Quality Review completed on 21/09/2021. 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 final score of the review is **85%**, 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 blockchain used by this protocol.

> **Chain:** Ethereum, Polygon, Arbitrum

Guidance:

Ethereum
Binance Smart Chain
Polygon
Avalanche
Terra

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://badger.wiki/addresses, 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:** 100%

Activity is 260 transactions a day on contract 0x3472A5A71965499acd81997a54BBA8D852C6E53d, 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/Badger-Finance/.

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%

An astonishing 1059 commits paired with 103 branches make BadgerDAO's development history exceptionally rich.

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://badger.wiki/badger

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://badger-finance.gitbook.io/badger-finance/

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

> ✓ **Answer:** Yes

All basic software functions are documented under the "Developer Info" subheading.

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

> **Answer:** 80%

The most important deployed contracts are covered in the "Developer Info" section of their 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:** 30%

Code examples are in the Appendix. As per the SLOC, there is 30% 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%

Documentation covers some of the BadgerDao code, but almost all functions are non-explicitly traceable to their source 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 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:** 100%

Code examples are in the Appendix. As per the SLOC, there is 147% 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

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

> ⓘ **Answer:** 55%

An audit conducted in August 2021 found 43.78% of Badger Finance's deployed code to be covered. In additon, the Zokyo audit has performed a code coverage test that returned an average of around 55%.

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/Instructions location: https://github.com/Badger-Finance/badger-system/tree/multichain

14) Report of the results (%)

⚠ **Answer:** 0%

No test result 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%

Badger Finance has not undergone a formal verification test.

16) Stress Testing environment (%)

> ✓ **Answer:** 100%

Badger Finance launches new vaults with stringent limits for an initial testing period to allow for bugs to be fixed before these vaults are open to the general public. There is also clear evidence of Rinkeby testnet usage at https://github.com/Badger-Finance/badger-system/blob/master/badger-rinkeby.json.

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%

Four different audits have taken place in the past year, all of which are public, and have been published pre-mainnet launch.

Notes On Audit Reports:

The Zokyo audit found that BadgerDAO was very secure and well-written. The only issue that was underlined in the report was an informational language usage flag in the Badger code. Essentially, they use internal functions for modifier roles in some of their contracts, but they should just use modifiers instead. Overall, Zokyo found nothing that could actively pose a risk to the smart contracts' integrity.

The Haechi audit found several minor and informational issues for the Badger team to work on. Unfortunately, there is no indication as to what the team did to resolve them. The underlined issues include a StakingReward bug where the contract's notifyRewardAmount() function would not check if it received rewards. This could lead to higher rewards for more active stakers, and potentially no rewards for others. In the same contract, another bug includes the notifyRewardAmount() function, where users could potentially be subjected to lower rewards rates. These are the most important findings, and all the other ones touch upon the language use and how it can be optimized.

The audit performed by Defi Yield did not find any issues. Rather, the report's only recommendations were to change the Controller and Sett contracts' governance addresses to "real" governance addresses. This would imply that both of these contracts are not linked to the actual BadgerDAO governance addresses.

The Quantstamp audit report unveiled multiple issues. Several of them were of medium risk, and one of them was high risk. The issue is that most of them, including the high-risk one, are not yet resolved. The high risk issue comprises the fact that the Core Badger contract has unbounded trust in its peaks. Peaks, as defined by the Badger documentation, are any third-party integration within the protocol. The issue here is that these peak contracts are telling the Core contract how many tokens to redeem, mint, or burn without limits or any form of verification. This means that any malicious peak contract could completely mess with the overrall Badger token integrity. As this issue is yet unresolved, this poses a serious problem. All other mentionned issues are either medium, low, or of unknown risk, and mostly affect the Core contract.

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:** 90%

Badger Finance has a $750k active bug bounty program.

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%

The access controls are clearly outlined under the security section of their website.

Notes On Mutability:

The Badger Finance code is clearly upgradeable due to their use of multiple proxies, namely UpgradeabilityProxy, as well as their use of numerous external calls to third-party sourced contracts. The combination of these facilitates implementation upgrades, which is something that is essential for a DAO.

To further optimize this, the initialize() functions is used multiple times throughout the Badger contracts. This allows for an easy way to upgrade a contract, even after a deployment to the mainnet,

In addition, migration is possible due to Badger's proxy structure, which facilitates contract upgrades to newer versions. This can be seen, most notably, in the Sett V1, V3, and V4 contracts due to the imported Upgradeability contracts from the OpenZeppelin library, as well as the presence of interface contracts (Interface contracts allow external contract calls).

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:** 80%

a) All contracts are clearly labelled as upgradeable (or not) -- 20% -- all important contracts are clearly labelled as upgradeable in a governance proposal, though not all deployed contracts are covered.

b) The type of ownership is clearly indicated (OnlyOwner / MultiSig / Defined Roles) -- 30% -- the ownership is clearly outlined in both the security section of the website and in the previous governance proposal.

c) The capabilities for change in the contracts are described -- 30% -- contract upgradeability is identified in the security pages.

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:** 90%

Description of admin controls is in clear and non-technical terms, and relates to user funds' safety.

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:** 40%

The documents mentions a "guardian" capable of pausing the protocol, but there is little elaboration.

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

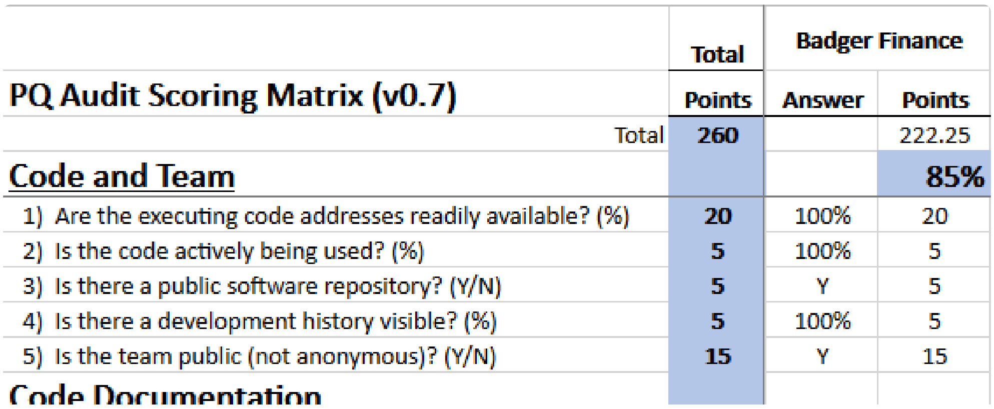
I started with Ethereum just before the DAO and that was a wonderful education. It showed the importance of code quality. The second Parity hack also showed the importance of good process. Here my aviation background offers some value. Aerospace knows how to make reliable code using quality processes.

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.

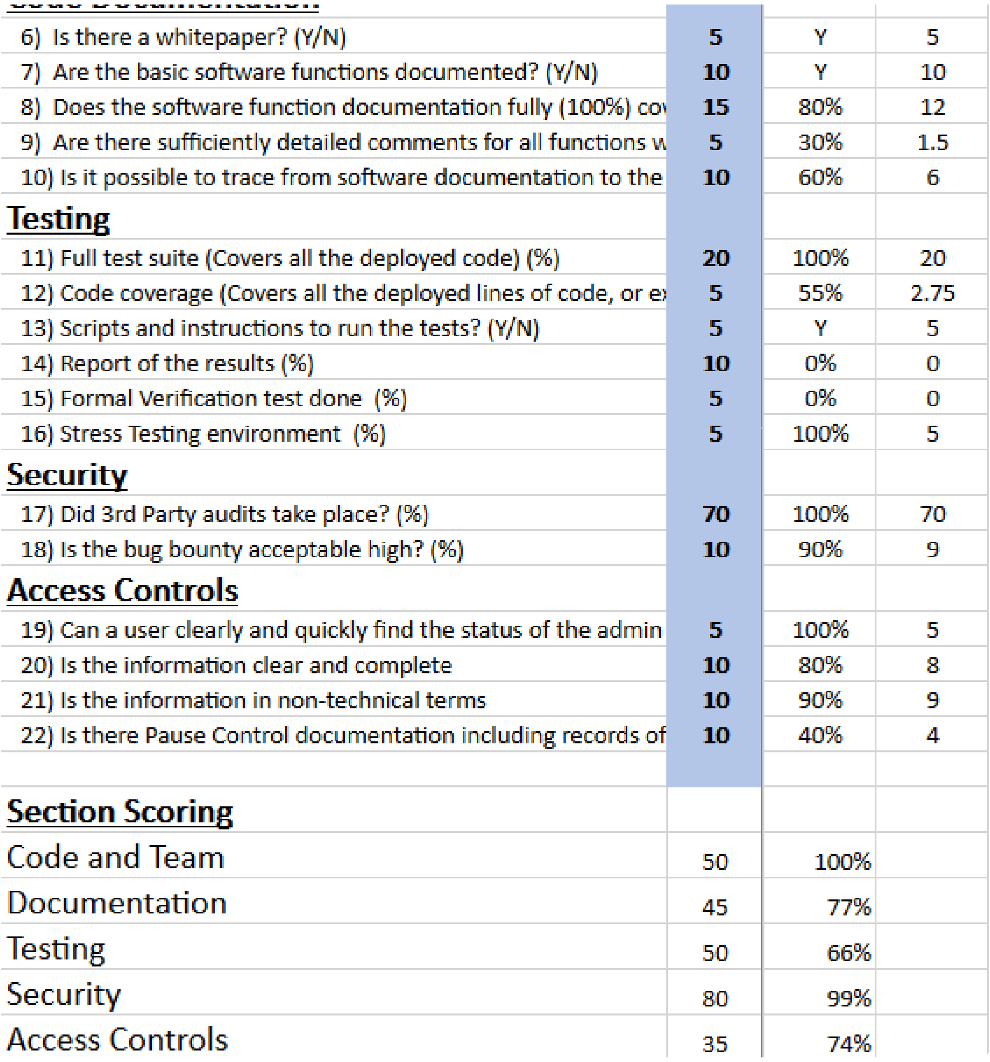
Process Quality Reviews are an extension of the SecurEth guidelines that will further increase the quality processes in Solidity and Vyper development.

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

Scoring Appendix

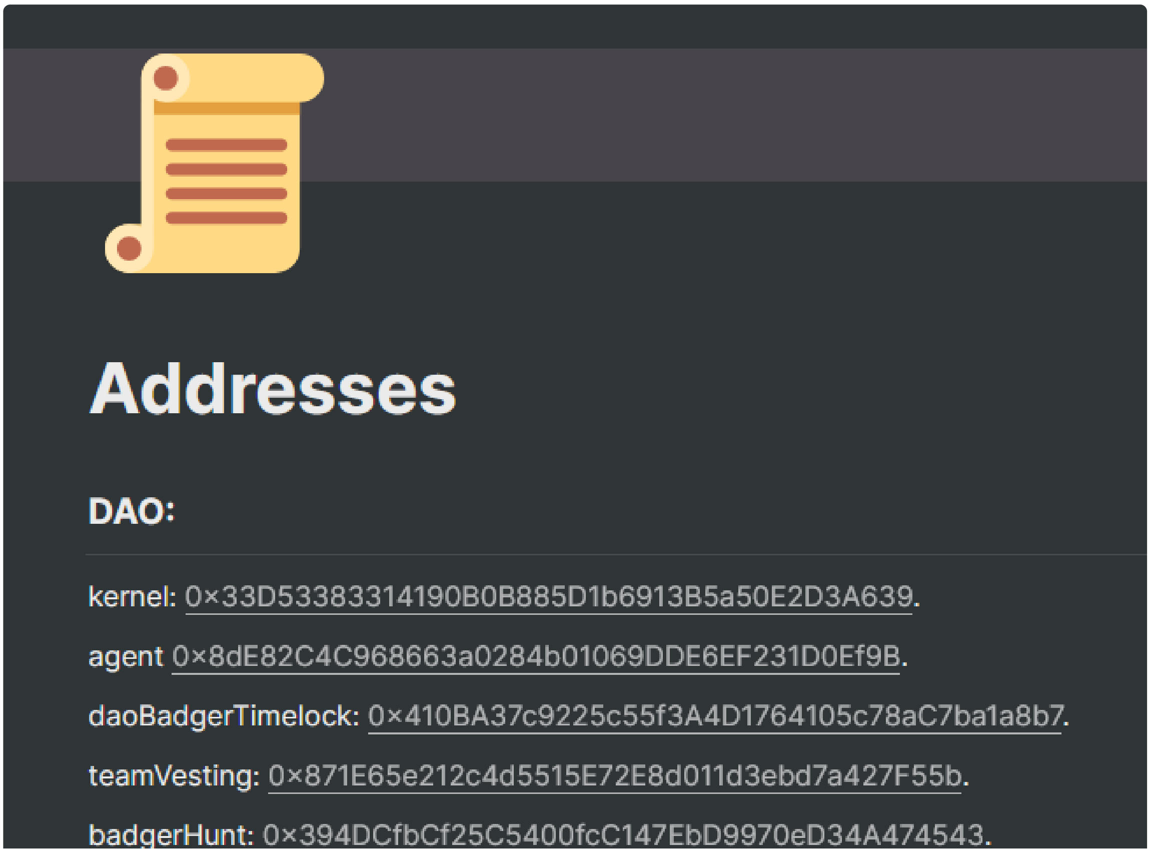


	Total	Badger Finance	
PQ Audit Scoring Matrix (v0.7)	Points	Answer	Points
Total	260		222.25
Code and Team			85%
1) Are the executing code addresses readily available? (%)	20	100%	20
2) Is the code actively being used? (%)	5	100%	5
3) Is there a public software repository? (Y/N)	5	Y	5
4) Is there a development history visible? (%)	5	100%	5
5) Is the team public (not anonymous)? (Y/N)	15	Y	15
Code Documentation			



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**	30%	1.5
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**	100%	20
12) Code coverage (Covers all the deployed lines of code, or ex	**5**	55%	2.75
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**	90%	9
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**	80%	8
21) Is the information in non-technical terms	**10**	90%	9
22) Is there Pause Control documentation including records of	**10**	40%	4
Section Scoring			
Code and Team	50	100%	
Documentation	45	77%	
Testing	50	66%	
Security	80	99%	
Access Controls	35	74%	

Executing Code Appendix



Addresses

DAO:

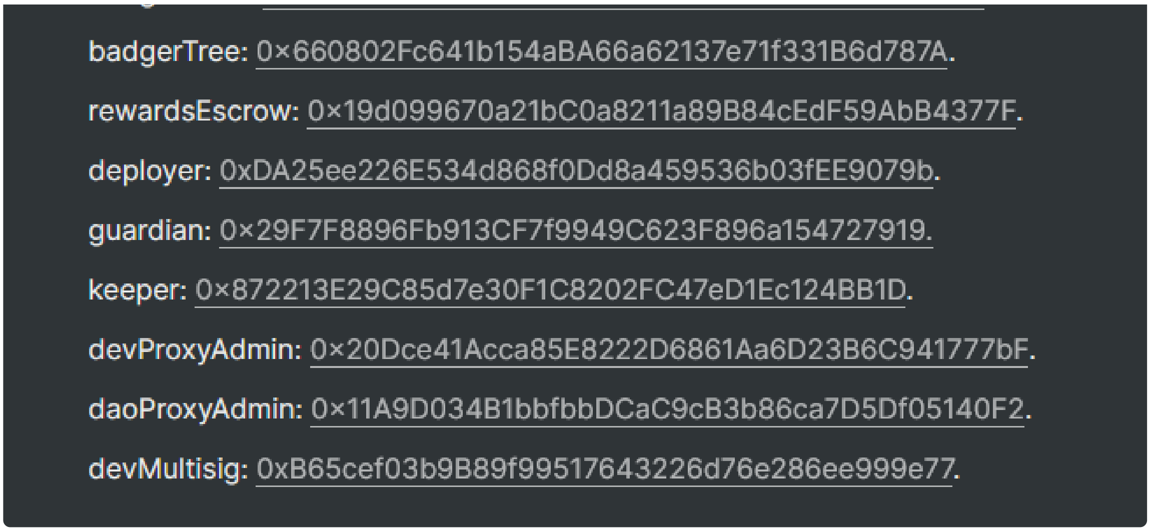
kernel: 0×33D53383314190B0B885D1b6913B5a50E2D3A639.

agent 0×8dE82C4C968663a0284b01069DDE6EF231D0Ef9B.

daoBadgerTimelock: 0×410BA37c9225c55f3A4D1764105c78aC7ba1a8b7.

teamVesting: 0×871E65e212c4d5515E72E8d011d3ebd7a427F55b.

badgerHunt: 0×394DCfbCf25C5400fcC147EbD9970eD34A474543.



badgerTree: 0×660802Fc641b154aBA66a62137e71f331B6d787A.

rewardsEscrow: 0×19d099670a21bC0a8211a89B84cEdF59AbB4377F.

deployer: 0xDA25ee226E534d868f0Dd8a459536b03fEE9079b.

guardian: 0×29F7F8896Fb913CF7f9949C623F896a154727919.

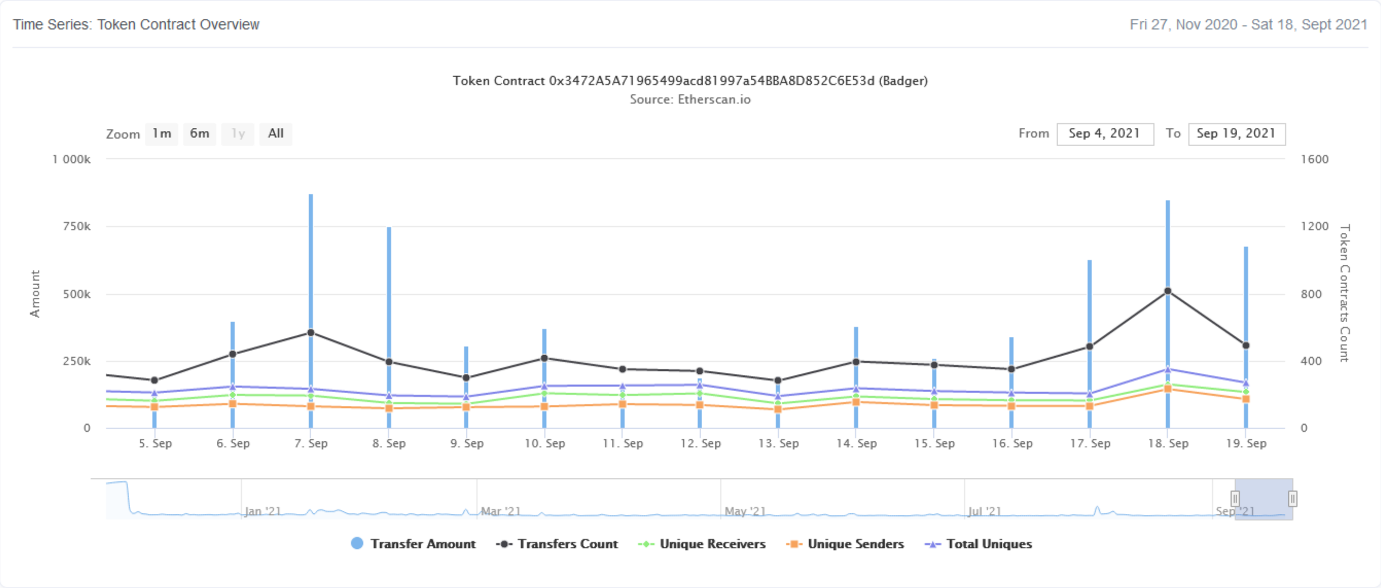
keeper: 0×872213E29C85d7e30F1C8202FC47eD1Ec124BB1D.

devProxyAdmin: 0×20Dce41Acca85E8222D6861Aa6D23B6C941777bF.

daoProxyAdmin: 0×11A9D034B1bbfbbDCaC9cB3b86ca7D5Df05140F2.

devMultisig: 0xB65cef03b9B89f99517643226d76e286ee999e77.

Code Used Appendix



Example Code Appendix

```
contract BadgerBridgeAdapter is OwnableUpgradeable, ReentrancyGuardUpgradeable {
    using SafeMathUpgradeable for uint256;
    using SafeERC20 for IERC20;

    IERC20 public renBTC;
    IERC20 public wBTC;

    // RenVM gateway registry.
    IGatewayRegistry public registry;
    // Swap router that handles swap routing optimizations.
    ISwapStrategyRouter public router;

    event RecoverStuck(uint256 amount, uint256 fee);
    event Mint(uint256 renbtc_minted, uint256 wbtc_swapped, uint256 fee);
    event Burn(uint256 renbtc_burned, uint256 wbtc_transferred, uint256 fee);

    event SwapError(bytes error);

    address public rewards;
    address public governance;

    uint256 public mintFeeBps;
    uint256 public burnFeeBps;
    uint256 private percentageFeeRewardsBps;
    uint256 private percentageFeeGovernanceBps;

    uint256 public constant MAX_BPS = 10000;

    mapping(address => bool) public approvedVaults;

    // Configurable permissionless curve lp token wrapper.
    address curveTokenWrapper;

    // Make struct for mint args, otherwise too many local vars (stack too deep).
    struct MintArguments {
        uint256 _mintAmount;
        uint256 _mintAmountMinusFee;
        uint256 _fee;
        uint256 _slippage;
        address _vault;
        address _user;
        address _token;
    }

    function initialize(
        address _governance,
        address _rewards,
        address _registry,
        address _router,
        address _wbtc,
        uint256[4] memory _feeConfig
    ) public initializer {
        __Ownable_init();
        __ReentrancyGuard_init();

        require(_governance != address(0x0), "must set governance address");
        require(_rewards != address(0x0), "must set rewards address");
        require(_registry != address(0x0), "must set registry address");
        require(_router != address(0x0), "must set router address");
        require(_wbtc != address(0x0), "must set wBTC address");

        governance = _governance;
        rewards = _rewards;

        registry = IGatewayRegistry(_registry);
        router = ISwapStrategyRouter(_router);
        renBTC = registry.getTokenBySymbol("BTC");
        wBTC = IERC20(_wbtc);

        mintFeeBps = _feeConfig[0];
        burnFeeBps = _feeConfig[1];
        percentageFeeRewardsBps = _feeConfig[2];
        percentageFeeGovernanceBps = _feeConfig[3];
    }

    function version() external pure returns (string memory) {
        return "1.1";
    }

    // NB: This recovery fn only works for the BTC gateway (hardcoded and only one support
    function recoverStuck(
        // encoded user args
        bytes calldata encoded,
        // darkdnode args
        uint256 _amount,
        bytes32 _nHash,
        bytes calldata _sig
    ) external nonReentrant {
        // Ensure sender matches sender of original tx.
        uint256 start = encoded.length - 32;
        address sender = abi.decode(encoded[start:], (address));
        require(sender == msg.sender);

        bytes32 pHash = keccak256(encoded);
        uint256 _mintAmount = registry.getGatewayBySymbol("BTC").mint(pHash, _amount, _nHa
        uint256 _fee = _processFee(renBTC, _mintAmount, mintFeeBps);

        emit RecoverStuck(_mintAmount, _fee);

        renBTC.safeTransfer(msg.sender, _mintAmount.sub(_fee));
    }

    function mint(
        // user args
        address _token, // either renBTC or wBTC
        uint256 _slippage,
        address _user,
        address _vault,
        // darknode args
        uint256 _amount,
        bytes32 _nHash,
        bytes calldata _sig
    ) external nonReentrant {
        require(_token == address(renBTC) || _token == address(wBTC), "invalid token addre

        // Mint renBTC tokens
        bytes32 pHash = keccak256(abi.encode(_token, _slippage, _user, _vault));
        uint256 mintAmount = registry.getGatewayBySymbol("BTC").mint(pHash, _amount, _nHas

        require(mintAmount > 0, "zero mint amount");

        uint256 fee = _processFee(renBTC, mintAmount, mintFeeBps);
        uint256 fee = _processFee(renBTC, mintAmount, mintFeeBps);
        uint256 mintAmountMinusFee = mintAmount.sub(fee);

        MintArguments memory args = MintArguments(mintAmount, mintAmountMinusFee, fee, _sl
        bool success = mintAdapter(args);

        if (!success) {
            renBTC.safeTransfer(_user, mintAmountMinusFee);
        }
    }

    function burn(
        // user args
        address _token, // either renBTC or wBTC
        address _vault,
        uint256 _slippage,
        bytes calldata _btcDestination,
        uint256 _amount
    ) external nonReentrant {
        require(_token == address(renBTC) || _token == address(wBTC), "invalid token addres
        require(!(_vault != address(0) && !approvedVaults[_vault]), "Vault not approved");

        bool isVault = _vault != address(0);
        bool isRenBTC = _token == address(renBTC);
        IERC20 token = isRenBTC ? renBTC : wBTC;
        uint256 startBalanceRenBTC = renBTC.balanceOf(address(this));
        uint256 startBalanceWBTC = wBTC.balanceOf(address(this));

        // Vaults can require up to two levels of unwrapping.
        if (isVault) {
            // First level of unwrapping for sett tokens.
            IERC20(_vault).safeTransferFrom(msg.sender, address(this), _amount);
            IERC20 vaultToken = IBridgeVault(_vault).token();

            uint256 beforeBalance = vaultToken.balanceOf(address(this));
            IBridgeVault(_vault).withdraw(IERC20(_vault).balanceOf(address(this)));
            uint256 balance = vaultToken.balanceOf(address(this)).sub(beforeBalance);

            // If the vault token does not match requested burn token, then we need to fur
            // vault token (e.g. withdrawing from crv sett gets us crv lp tokens which nee
            if (address(vaultToken) != _token) {
                vaultToken.safeTransfer(curveTokenWrapper, balance);
                ICurveTokenWrapper(curveTokenWrapper).unwrap(_vault);
            }
        } else {
            token.safeTransferFrom(msg.sender, address(this), _amount);
        }

        uint256 wbtcTransferred = wBTC.balanceOf(address(this)).sub(startBalanceWBTC);

        if (!isRenBTC) {
            _swapWBTCForRenBTC(wbtcTransferred, _slippage);
        }

        uint256 toBurnAmount = renBTC.balanceOf(address(this)).sub(startBalanceRenBTC);
        uint256 fee = _processFee(renBTC, toBurnAmount, burnFeeBps);

        uint256 burnAmount = registry.getGatewayBySymbol("BTC").burn(_btcDestination, toBu

        emit Burn(burnAmount, wbtcTransferred, fee);
    }

    function mintAdapter(MintArguments memory args) internal returns (bool) {
        if (args._vault != address(0) && !approvedVaults[args._vault]) {
            return false;
        }

        uint256 wbtcExchanged;
        bool isVault = args._vault != address(0);
        bool isRenBTC = args._token == address(renBTC);
        IERC20 token = isRenBTC ? renBTC : wBTC;

if (!isRenBTC) {
            // Try and swap and transfer wbtc if token wbtc specified.
            uint256 startBalance = token.balanceOf(address(this));
            if (!_swapRenBTCForWBTC(args._mintAmountMinusFee, args._slippage)) {
                return false;
            }
            uint256 endBalance = token.balanceOf(address(this));
            wbtcExchanged = endBalance.sub(startBalance);
        }

        emit Mint(args._mintAmount, wbtcExchanged, args._fee);

        uint256 amount = isRenBTC ? args._mintAmountMinusFee : wbtcExchanged;


```

SLOC Appendix

Solidity Contracts

Language	Files	Lines	Blanks	Comments	Code	Complex
Solidity	54	7522	1339	1413	4770	446

Comments to Code 1413/4770 = 30%

Javascript Tests

Language	Files	Lines	Blanks	Comments	Code	Complex

JavaScript	1	246	26	24	196	6
Python	47	11646	2304	2736	6606	203
JSON	2	227	0	0	227	0
Total	50	12119	2330	2760	7029	209

Tests to Code 7029/4770 = 147%