0.7

Kalmar Process Quality Review

Score: 31%

Overview

This is a Kalmar Process Quality Review completed on August 4th 2021. It was performed using the Process Review process (version 0.7.3) and is documented here. The review was performed by Nic of DeFiSafety. Check out our Telegram.

The final score of the review is 31%, a FAIL. 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:** Binance Smart Chain

Guidance:

Ethereum
Binance Smart Chain
Polygon
Avalanche

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:** 70%

They are available at website https://docs.kalmar.io/leveraged-yield-farming, as indicated in the Appendix.

Note: Only the Yield Farming Bank and Token addresses are publicly available.

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

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 towards the final score.

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

> **Answer:** 100%

Activity is 50 transactions a day on contract *Bank.sol*, 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/kalmar-io/leverage-yield-contracts-busd.

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:** 0%

With 1 commit and 1 branch, this is an unhealthy repository.

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

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.

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

⚠ **Answer:** No

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.kalmar.io/#teams-philosophy-and-experience.

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

> **Answer:** Yes

The basic global parameters in the Kalmar Leveraged Yield Farming system are documented at https://docs.kalmar.io/leveraged-yield-farming/risk-parameters.

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

> **Answer:** 33%

Out of the 3 Kalmar applications detailed in their documentation (Farming, Exchange Aggregator, and Token), only 1 of them has its basic software functions documented (Farming).

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:** 29%

Code examples are in the Appendix. As per the SLOC, there is 29% 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:** 0%

There is no traceability between the Kalmar software documentation and its 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:** 0%

There is no testing suite in the Kalmar's GitHub repository.

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:** 0%

There is no evidence of any code coverage in any of the Kalmar documentation or in their Hacken audit reports.

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:** No

There are scripts and/or instructions to run tests inside Kalmar's main contracts repository.

How to improve this score:

Add the scripts to the repository and ensure they work. Ask an outsider to create the environment and run the tests. Improve the scripts and docs based on their feedback.

14) Report of the results (%)

> ⚠ **Answer:** 0%

There is no evidence of a Kalmar test report inside of their GitHub repository.

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%

No evidence of a Kalmar Formal Verification test was found in any of their documentation.

16) Stress Testing environment (%)

> ✓ **Answer:** 100%

There is evidence of Kalmar's stress testing a https://docs.kalmar.io/public-testnet.

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: 4**0%

Hacken has published two Kalmar audit reports on their Yield Farming contracts. However, the 2 audit reports each highlighted sever weaknesses in the design. In the first they mentioned that no test cases exists, implying the software is untested. In the second the auditor mention that without even general

documentation does not exist so they had to audit code that they did not know the functionality of. For this reason the score is reduced to 40%.

The first one was published on April 3rd 2021, which was pre-V1 launch.

The second one was published on June 8th 2021, which was pre-V2 launch.

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:** 20%

Kalmar's Bug Bounty program is active and rewards participating users with up to 15k for the most critical of finds.

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:** 0%

As of the time this review was written, there is no access control/governance information in the Kalmar documentation.

Note: It seems like there was a "Transparency" section in their GitBooks, but it has since been deleted and is inaccessible. Even though it still pops up in the search bar, you cannot click on it.

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:** 0%

As of the time this review was written, there is no access control/governance information in the Kalmar documentation.

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:** 0%

As of the time this review was written, there is no access control/governance information in the Kalmar documentation.

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 or similar functions were detailed in the Kalmar documentation.

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	Kalmar	
PQ Audit Scoring Matrix (v0.7)	**Points**	**Answer**	**Points**
Total	**260**		80.4
Code and Team			**31%**
1) Are the executing code addresses readily available? (%)	**20**	70%	14
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**	0%	0
5) Is the team public (not anonymous)? (Y/N)	**15**	N	0
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%) cover the deployed contracts? (%	**15**	33%	4.95
9) Are there sufficiently detailed comments for all functions within the deployed contract cc	**5**	29%	1.45
10) Is it possible to trace from software documentation to the implementation in code (%)	**10**	0%	0
Testing			
11) Full test suite (Covers all the deployed code) (%)	**20**	0%	0
12) Code coverage (Covers all the deployed lines of code, or explains misses) (%)	**5**	0%	0
13) Scripts and instructions to run the tests? (Y/N)	**5**	N	0
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**	40%	28
18) Is the bug bounty acceptable high? (%)	**10**	20%	2
Access Controls			
19) Can a user clearly and quickly find the status of the admin controls	**5**	0%	0
20) Is the information clear and complete	**10**	0%	0
21) Is the information in non-technical terms	**10**	0%	0
22) Is there Pause Control documentation including records of tests	**10**	0%	0

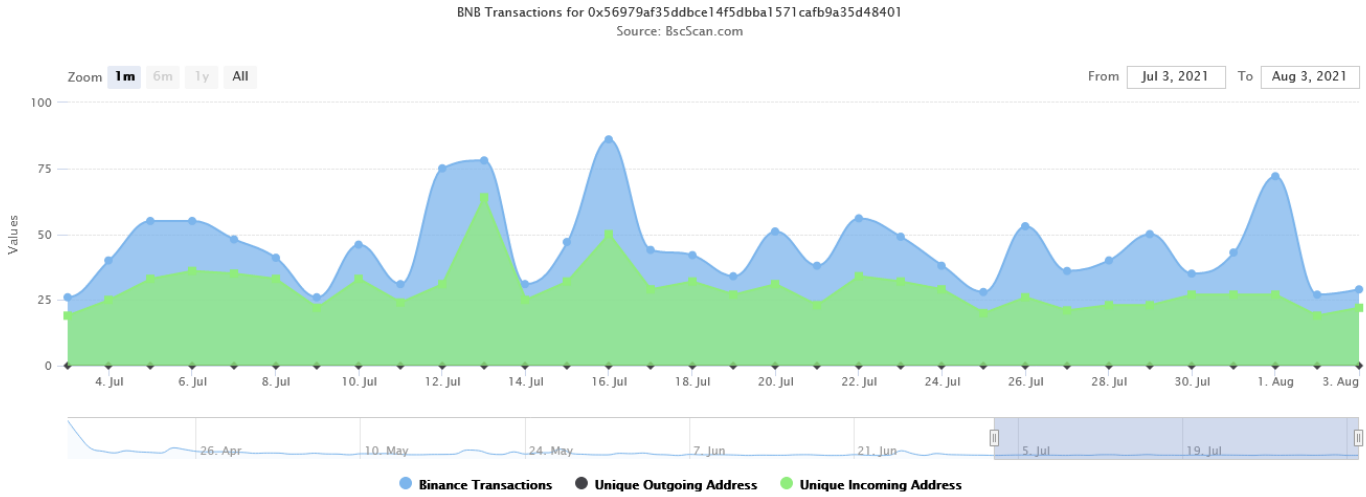
Section Scoring			
Code and Team	50	48%	
Documentation	45	48%	
Testing	50	10%	
Security	80	38%	
Access Controls	35	0%	

Executing Code Appendix

IBNB Contract:

https://bscscan.com/address/0x56979af35ddbce14f5dbba1571cafb9a35d48401

Code Used Appendix



Example Code Appendix

```
contract Bank is ERC20, ReentrancyGuard, Ownable {
    /// @notice Libraries
    using SafeToken for address;
    using SafeMath for uint256;

    /// @notice Events
    event AddDebt(uint256 indexed id, uint256 debtShare);
    event RemoveDebt(uint256 indexed id, uint256 debtShare);
    event Work(uint256 indexed id, uint256 loan);
    event Kill(uint256 indexed id, address indexed killer, uint256 prize, uint256 left);

    address public token;
```

```
    string public name = "Interest Bearing BUSD";
    string public symbol = "iBUSD";
    uint8 public decimals = 18;

    bool public killBpsToTreasury;
    address public treasuryAddr;

    struct Position {
        address goblin;
        address owner;
        uint256 debtShare;
    }

    BankConfig public config;
    mapping (uint256 => Position) public positions;
    uint256 public nextPositionID = 1;

    uint256 public glbDebtShare;
    uint256 public glbDebtVal;
    uint256 public lastAccrueTime;
    uint256 public reservePool;

    /// @dev Require that the caller must be an EOA account to avoid flash loans.
    modifier onlyEOA() {
        require(msg.sender == tx.origin, "not eoa");
        _;
    }

    /// @dev Get token from msg.sender
    modifier transferTokenToVault(uint256 value) {
        SafeToken.safeTransferFrom(token, msg.sender, address(this), value);
        _;
    }

    /// @dev Add more debt to the global debt pool.
    modifier accrue(uint256 msgValue) {
        if (now > lastAccrueTime) {
            uint256 interest = pendingInterest(msgValue);
            uint256 toReserve = interest.mul(config.getReservePoolBps()).div(10000);
            reservePool = reservePool.add(toReserve);
            glbDebtVal = glbDebtVal.add(interest);
            lastAccrueTime = now;
        }
        _;
    }

    constructor(
        BankConfig _config,
        address _token,
        bool _killBpsToTreasury,
        address _treasuryAddr
        ) public {
        config = _config;
```

```
66          token = _token;
67          killBpsToTreasury = _killBpsToTreasury;
68          treasuryAddr = _treasuryAddr;
69          lastAccrueTime = now;
70      }
71
72      /// @dev Return the pending interest that will be accrued in the next call.
73      /// @param msgValue Balance value to subtract off address(this).balance when called fr
74      function pendingInterest(uint256 msgValue) public view returns (uint256) {
75          if (now > lastAccrueTime) {
76              uint256 timePast = now.sub(lastAccrueTime);
77              uint256 balance = token.myBalance().sub(msgValue);
78              uint256 ratePerSec = config.getInterestRate(glbDebtVal, balance);
79              return ratePerSec.mul(glbDebtVal).mul(timePast).div(1e18);
80          } else {
81              return 0;
82          }
83      }
84
85      /// @dev Return the ETH debt value given the debt share. Be careful of unaccrued intere
86      /// @param debtShare The debt share to be converted.
87      function debtShareToVal(uint256 debtShare) public view returns (uint256) {
88          if (glbDebtShare == 0) return debtShare; // When there's no share, 1 share = 1 val
89          return debtShare.mul(glbDebtVal).div(glbDebtShare);
90      }
91
92      /// @dev Return the debt share for the given debt value. Be careful of unaccrued intere
93      /// @param debtVal The debt value to be converted.
94      function debtValToShare(uint256 debtVal) public view returns (uint256) {
95          if (glbDebtShare == 0) return debtVal; // When there's no share, 1 share = 1 val.
96          return debtVal.mul(glbDebtShare).div(glbDebtVal);
97      }
98
99      /// @dev Return ETH value and debt of the given position. Be careful of unaccrued inte
100     /// @param id The position ID to query.
101     function positionInfo(uint256 id) public view returns (uint256, uint256) {
102         Position storage pos = positions[id];
103         return (Goblin(pos.goblin).health(id), debtShareToVal(pos.debtShare));
104     }
105
106     /// @dev Return the total ETH entitled to the token holders. Be careful of unaccrued i
107     function totalBEP20() public view returns (uint256) {
108         return token.myBalance().add(glbDebtVal).sub(reservePool);
109     }
110
111     /// @dev Add more ETH to the bank. Hope to get some good returns.
112     function deposit(uint256 amountToken) external transferTokenToVault(amountToken) accru
113         uint256 total = totalBEP20().sub(amountToken);
114         uint256 share = total == 0 ? amountToken : amountToken.mul(totalSupply()).div(tota
115         _mint(msg.sender, share);
116     }
117
118     /// @dev Withdraw ETH from the bank by burning the share tokens.
```

```
119
    function withdraw(uint256 share) external accrue(0) nonReentrant {
120         uint256 amount = share.mul(totalBEP20()).div(totalSupply());
121         _burn(msg.sender, share);
122         SafeToken.safeTransfer(token, msg.sender, amount);
123     }
124
125     /// @dev Create a new farming position to unlock your yield farming potential.
126     /// @param id The ID of the position to unlock the earning. Use ZERO for new position.
127     /// @param goblin The address of the authorized goblin to work for this position.
128     /// @param loan The amount of ETH to borrow from the pool.
129     /// @param maxReturn The max amount of ETH to return to the pool.
130     /// @param data The calldata to pass along to the goblin for more working context.
131     function work(uint256 id, address goblin, uint256 principalAmount, uint256 loan, uint25
132         external
133         onlyEOA transferTokenToVault(principalAmount) accrue(principalAmount) nonReentrant
134     {
135         // 1. Sanity check the input position, or add a new position of ID is 0.
136         if (id == 0) {
137             id = nextPositionID++;
138             positions[id].goblin = goblin;
139             positions[id].owner = msg.sender;
140         } else {
141             require(id < nextPositionID, "bad position id");
142             require(positions[id].goblin == goblin, "bad position goblin");
143             require(positions[id].owner == msg.sender, "not position owner");
144         }
145         emit Work(id, loan);
146         // 2. Make sure the goblin can accept more debt and remove the existing debt.
147         require(config.isGoblin(goblin), "not a goblin");
148         require(loan == 0 || config.acceptDebt(goblin), "goblin not accept more debt");
149         uint256 debt = _removeDebt(id).add(loan);
150         // 3. Perform the actual work, using a new scope to avoid stack-too-deep errors.
151         uint256 back;
152         {
153             uint256 sendBEP20 = principalAmount.add(loan);
154             require(sendBEP20 <= token.myBalance(), "Vault::work:: insufficient funds in th
155             uint256 beforeBEP20 = token.myBalance().sub(sendBEP20);
156             SafeToken.safeTransfer(token, goblin, sendBEP20);
157             Goblin(goblin).work(id, msg.sender, debt, data);
158             back = token.myBalance().sub(beforeBEP20);
159         }
160         // 4. Check and update position debt.
161         uint256 lessDebt = Math.min(debt, Math.min(back, maxReturn));
162         debt = debt.sub(lessDebt);
163         if (debt > 0) {
164             require(debt >= config.minDebtSize(), "too small debt size");
165             uint256 health = Goblin(goblin).health(id);
166             uint256 workFactor = config.workFactor(goblin, debt);
167             require(health.mul(workFactor) >= debt.mul(10000), "bad work factor");
168             _addDebt(id, debt);
169         }
170         // 5. Return excess ETH back.
171         if (back > lessDebt) SafeToken.safeTransfer(token, msg.sender, back.sub(lessDebt));
```

```
171         if (back > lessDebt) SafeToken.safeTransfer(token, msg.sender, back.sub(lessDebt));
172
        }
173
174     /// @dev Kill the given to the position. Liquidate it immediately if killFactor condit
175     /// @param id The position ID to be killed.
176     function kill(uint256 id) external onlyEOA accrue(0) nonReentrant {
177         // 1. Verify that the position is eligible for liquidation.
178         Position storage pos = positions[id];
179         require(pos.debtShare > 0, "no debt");
180         uint256 debt = _removeDebt(id);
181         uint256 health = Goblin(pos.goblin).health(id);
182         uint256 killFactor = config.killFactor(pos.goblin, debt);
183         require(health.mul(killFactor) < debt.mul(10000), "can't liquidate");
184         // 2. Perform liquidation and compute the amount of ETH received.
185         uint256 beforeETH = token.myBalance();
186         Goblin(pos.goblin).liquidate(id);
187         uint256 back = token.myBalance().sub(beforeETH);
188         uint256 prize = back.mul(config.getKillBps()).div(10000);
189         uint256 rest = back.sub(prize);
190         // 3. Clear position debt and return funds to liquidator and position owner.
191         if (prize > 0) {
192             address rewardTo = killBpsToTreasury == true ? treasuryAddr : msg.sender;
193             SafeToken.safeTransfer(token, rewardTo, prize);
194         }
195         uint256 left = rest > debt ? rest - debt : 0;
196         if (left > 0) SafeToken.safeTransfer(token, pos.owner, left);
197         emit Kill(id, msg.sender, prize, left);
198     }
```

SLOC Appendix

Solidity Contracts

Language	Files	Lines	Blanks	Comments	Code	Complex
Solidity	14	1516	164	304	1048	94

Comments to Code 304/1048 = 29%

Javascript Tests

Language	Files	Lines	Blanks	Comments	Code	Complex
JavaScript	N/A	N/A	N/A	N/A	N/A	N/A

Tests to Code = N/A