0.7

Helmet.insure Process Quality Review

Score: 27%

Overview

This is a Helmet Insure Process Quality Review completed on July 28th 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 27%, 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

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:** 100%

They are available at website https://helmet-insure.gitbook.io/helmet/token/helmet-token#sfg-token-contract, 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 300 transactions a day on contract *Proxy.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/helmet-insure/contracts.

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:** 30%

With 45 commits and 1 branch, this is an unhealthy software 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

Location: No public team information was found.

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://helmet-insure.gitbook.io/helmet/.

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

> ✓ **Answer:** Yes

The Helmet.insure basic software functions are documented in the "Contract Guide" section of their documentation, as well as in the "Meet Helmet" section.

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

> **Answer:** 100%

Helmet has detailed documentation on all the option logic functions that constitute the core of all of their contracts. In addition, they have a detailed section for each of the six main functions (OptionFactory, LongOption, ShortOption, OptionOrder, HELMET Token, and Farm).

Since the software guidelines and parameters outlined in the Helmet documentation are hardcoded and automatically executed, they essentially fully cover their main executing code in their software function 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

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

> **Answer:** 74%

Code examples are in the Appendix. As per the SLOC, there is 74% 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%

There is a clear association between the software functions and documentation via non-explicit traceability to its implementation in the Helmet source code. Meaning you can easily find these functions in the source code, but you are not explicitly guided there. You need to find the source code yourself.

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 inside of the Helmet.insure 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%

No code coverage found in the Helmet.insure software repositories, and thy have no audit reports that we could check for code coverage in.

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

Scrips/Instructions location: No testing suite.

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%

No testing suite.

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 Formal Verification test was found in the Helmet.insure documentation.

16) Stress Testing environment (%)

> ⚠ **Answer:** 0%

There is no evidence of Helmet.insure's test-net smart contract usage in their documentation.

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:** 0%

Certik has performed an audit for Helmet.insure. However, the results are not public, and no indication as to fix recommendation or implementation by the Helmet team. Ergo, 0.

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%

There is no evidence of a Helmet.insure Bug Bounty Program in their documentation.

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%

There is a governance section in the Helmet.insure documentation. However, it only details token capabilities and does not actually provide any useful admin access control information.

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%

None of the information below can be found in the Helmet.insure 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%

There is no admin control information in the Helmet.insure 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 found in the Helmet.insure documentation or in their GitHub repositories.

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

PQ Audit Scoring Matrix (v0.7)	Total Points	Helmet.insure Answer	Helmet.insure Points
Total	260		71.2
Code and Team			**27%**
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	30%	1.5
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	100%	15
9) Are there sufficiently detailed comments for all functions within the deployed contract code (%)	5	74%	3.7
10) Is it possible to trace from software documentation to the implementation in code (%)	10	60%	6
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	0%	0
Security			
17) Did 3rd Party audits take place? (%)	70	0%	0
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 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	63%	
Documentation	45	88%	
Testing	50	0%	
Security	80	0%	
Access Controls	35	0%	

Executing Code Appendix

`HELMET` Token Contract Address

> Before any step, please check that the whether the contract address is the correct `HELMET` contract ! ! !

`HELMET` Token Contract

```
0x948d2a81086A075b3130BAc19e4c6DEe1D2E3fE8
```

Timelock Contract

```
0xdE5bFdBCa3C4AF7937c1a1104093a809116fCdD5
```

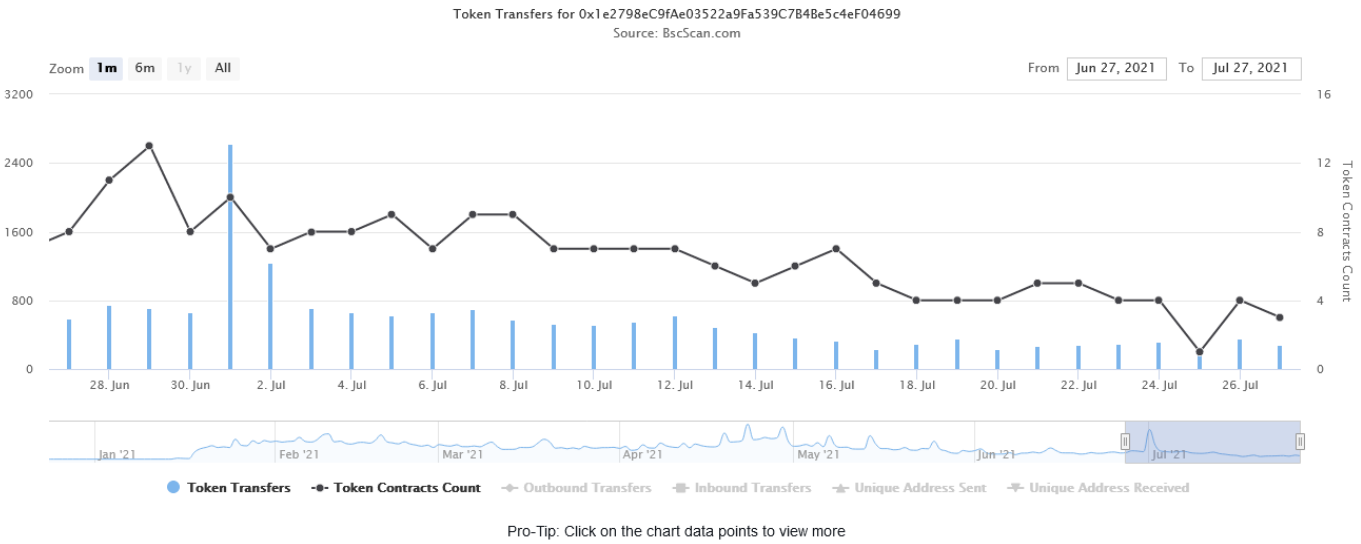
Farm Contract

```
0x1e2798eC9fAe03522a9Fa539C7B4Be5c4eF04699
```

Factory Contract

```
0x021297e233550eDBa8e6487EB7c6696cFBB63b88
```

Code Used Appendix



Pro-Tip: Click on the chart data points to view more

Example Code Appendix

```
1 /**
```

```
 * @title Proxy
 * @dev Implements delegation of calls to other contracts, with proper
 * forwarding of return values and bubbling of failures.
 * It defines a fallback function that delegates all calls to the address
 * returned by the abstract _implementation() internal function.
 */
abstract contract Proxy {
  /**
   * @dev Fallback function.
   * Implemented entirely in `_fallback`.
   */
  fallback () payable external {
    _fallback();
  }

  receive () payable external {
    _fallback();
  }

  /**
   * @return The Address of the implementation.
   */
  function _implementation() virtual internal view returns (address);

  /**
   * @dev Delegates execution to an implementation contract.
   * This is a low level function that doesn't return to its internal call site.
   * It will return to the external caller whatever the implementation returns.
   * @param implementation Address to delegate.
   */
  function _delegate(address implementation) internal {
    assembly {
      // Copy msg.data. We take full control of memory in this inline assembly
      // block because it will not return to Solidity code. We overwrite the
      // Solidity scratch pad at memory position 0.
      calldatacopy(0, 0, calldatasize())

      // Call the implementation.
      // out and outsize are 0 because we don't know the size yet.
      let result := delegatecall(gas(), implementation, 0, calldatasize(), 0, 0)

      // Copy the returned data.
      returndatacopy(0, 0, returndatasize())

      switch result
      // delegatecall returns 0 on error.
      case 0 { revert(0, returndatasize()) }
      default { return(0, returndatasize()) }
    }
  }

  /**
   * @dev Function that is run as the first thing in the fallback function.
```

```
55
     * Can be redefined in derived contracts to add functionality.
56
     * Redefinitions must call super._willFallback().
57
     */
58
    function _willFallback() virtual internal {
59
60
    }
61
62
    /**
63
     * @dev fallback implementation.
64
     * Extracted to enable manual triggering.
65
     */
66
    function _fallback() internal {
67
      if(OpenZeppelinUpgradesAddress.isContract(msg.sender) && msg.data.length == 0 && gasle
68
          return;
69
      _willFallback();
70
      _delegate(_implementation());
71
    }
72 }
73
74
75 /**
76  * @title BaseUpgradeabilityProxy
77  * @dev This contract implements a proxy that allows to change the
78  * implementation address to which it will delegate.
79  * Such a change is called an implementation upgrade.
80  */
81 abstract contract BaseUpgradeabilityProxy is Proxy {
82    /**
83     * @dev Emitted when the implementation is upgraded.
84     * @param implementation Address of the new implementation.
85     */
86    event Upgraded(address indexed implementation);
87
88    /**
89     * @dev Storage slot with the address of the current implementation.
90     * This is the keccak-256 hash of "eip1967.proxy.implementation" subtracted by 1, and is
91     * validated in the constructor.
92     */
93    bytes32 internal constant IMPLEMENTATION_SLOT = 0x360894a13ba1a3210667c828492db98dca3e20
94
95    /**
96     * @dev Returns the current implementation.
97     * @return impl Address of the current implementation
98     */
99    function _implementation() override internal view returns (address impl) {
100     bytes32 slot = IMPLEMENTATION_SLOT;
101     assembly {
102       impl := sload(slot)
103     }
104   }
105
106   /**
107    * @dev Upgrades the proxy to a new implementation.
```

```
108
     * @param newImplementation Address of the new implementation.
109    */
110   function _upgradeTo(address newImplementation) internal {
111     _setImplementation(newImplementation);
112     emit Upgraded(newImplementation);
113   }
114
115   /**
116    * @dev Sets the implementation address of the proxy.
117    * @param newImplementation Address of the new implementation.
118    */
119   function _setImplementation(address newImplementation) internal {
120     require(OpenZeppelinUpgradesAddress.isContract(newImplementation), "Cannot set a proxy
121
122     bytes32 slot = IMPLEMENTATION_SLOT;
123
124     assembly {
125       sstore(slot, newImplementation)
126     }
127   }
128 }
129
130
131 /**
132  * @title BaseAdminUpgradeabilityProxy
133  * @dev This contract combines an upgradeability proxy with an authorization
134  * mechanism for administrative tasks.
135  * All external functions in this contract must be guarded by the
136  * `ifAdmin` modifier. See ethereum/solidity#3864 for a Solidity
137  * feature proposal that would enable this to be done automatically.
138  */
139 contract BaseAdminUpgradeabilityProxy is BaseUpgradeabilityProxy {
140   /**
141    * @dev Emitted when the administration has been transferred.
142    * @param previousAdmin Address of the previous admin.
143    * @param newAdmin Address of the new admin.
144    */
145   event AdminChanged(address previousAdmin, address newAdmin);
146
147   /**
148    * @dev Storage slot with the admin of the contract.
149    * This is the keccak-256 hash of "eip1967.proxy.admin" subtracted by 1, and is
150    * validated in the constructor.
151    */
152
153   bytes32 internal constant ADMIN_SLOT = 0xb53127684a568b3173ae13b9f8a6016e243e63b6e8ee117
154
155   /**
156    * @dev Modifier to check whether the `msg.sender` is the admin.
157    * If it is, it will run the function. Otherwise, it will delegate the call
158    * to the implementation.
159    */
```

```
160   modifier ifAdmin() {
161
        if (msg.sender == _admin()) {
162       _;
163     } else {
164       _fallback();
165     }
166   }
167
168   /**
169    * @return The address of the proxy admin.
170    */
171   function admin() external ifAdmin returns (address) {
172     return _admin();
173   }
174
175   /**
176    * @return The address of the implementation.
177    */
178   function implementation() external ifAdmin returns (address) {
179     return _implementation();
180   }
181
182   /**
183    * @dev Changes the admin of the proxy.
184    * Only the current admin can call this function.
185    * @param newAdmin Address to transfer proxy administration to.
186    */
187   function changeAdmin(address newAdmin) external ifAdmin {
188     require(newAdmin != address(0), "Cannot change the admin of a proxy to the zero addres
189     emit AdminChanged(_admin(), newAdmin);
190     _setAdmin(newAdmin);
191   }
192
193   /**
194    * @dev Upgrade the backing implementation of the proxy.
195    * Only the admin can call this function.
196    * @param newImplementation Address of the new implementation.
197    */
198   function upgradeTo(address newImplementation) external ifAdmin {
199     _upgradeTo(newImplementation);
200   }
```

SLOC Appendix

Solidity Contracts

Language	Files	Lines	Blanks	Comments	Code	Complex
Solidity	14	11967	1811	4306	5850	830

Comments to Code 4306/5850 = 74%

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