0.7

AutoFarm Process Quality Review

Score: 47%

Overview

This is a AutoFarm Process Quality Review completed on June 21st 2021. It was performed using the Process Review process (version 0.7.2) and is documented here. The review was performed by Nic of DeFiSafety. Check out our Telegram.

The final score of the review is 47%, 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, Polygon**

Guidance:
Ethereum
Binance Smart Chain
Polygon

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? (%)
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://autofarm.gitbook.io/autofarm-network/protocol/tokenomics as indicated in the Appendix.

Note: Gave them a 70% because the AutoFarm executing smart contract addresses were found at the bottom of a page in their GitBooks called "Tokenomics", and scattered in each page of the vaults.

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 labelling 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 wrt to the final score.

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

> ✓ Answer: 100%

Activity is 200 transactions a day on contract *AutoFarmV2_CrossChain.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

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

> ✓ Answer: Yes

GitHub: https://github.com/autofarmnetwork

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.

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

> ⓘ Answer: 70%

With 74 commits and 1 branch, this is an adequate 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.

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

> ⚠ Answer: No

Team members are all anonymous and use aliases as seen here https://autofarm.gitbook.io/autofarm-network/team. The team did a KYC with the binance corpdev team before listing, although they are still anonymous.

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;

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://autofarm.gitbook.io/autofarm-network/.

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

> ⚠ Answer: No

No basic function documentation in their GitHub or GitBooks.

How to improve this score

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

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

> ⚠ Answer: 0%

There is no software documentation in their GitHub or GitBooks.

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.

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

> ⓘ Answer: 33%

Code examples are in the Appendix. As per the SLOC, there is 33% 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%

No software documentation, and therefore no code implementation traceability.

Guidance:
100% Clear explicit traceability between code and documentation at a requirement level for all code
60% Clear association between code and documents via non explicit traceability
40% Documentation lists all the functions and describes their functions
0% No connection between documentation and code

How to improve this score

This score can improve by adding traceability from 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;

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%

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

Note: No test files in any of their repositories.

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.

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

> ⚠ Answer: 0%

There is no indication of code coverage.

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.

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

> ⚠ Answer: No

There are no scripts or instructions to run tests, as the tests themselves are non-existent.

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 test report was found in any of their repositories.

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 formal verification of AutoFarm was found in their documentation or in web searches.

16) Stress Testing environment (%)

⚠ Answer: 0%

No evidence of test-net smart contract address usage was found in any of 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: 100%

SlowMist did a AutoFarm audit in April 2021.

Vidar released a AutoFarm audit on January 19th 2021. (Outdated, pre Autofarmv2)

Certik released a AutoFarm security assessment on March 28th 2021.

Certik released another AutoFarm security assessment on May 4th 2021.

Note: AutoFarm BSC was released on December 15th 2020, and AutoFarm Polygon was release on May 23rd 2021.

Note 2: These audits are mostly AutoFarm V2 reports, which was released on January 21st 2021.

Note 3: The Vidar audit, which was the only to be published before the V2 release, does not give updates as to solution implementation.

Note 4: About 50% of changes were implemented.

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, question

18) Is the bounty value acceptably high (%)

> ⓘ Answer: 70%

Bug bounty program found at https://immunefi.com/bounty/autofarm/.

Program is active and offers a reward as high as 100k.

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

Active program means a third party actively driving hackers to the site. Inactive program would be static

mention 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?
2`) 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: 40%

Basic access control documentation can be found in their gitbook.

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: 60%

Contract clearly labelled as upgradeable at https://autofarm.gitbook.io/autofarm-network/security-and-risks.

AutoFarm describes their use of Gnosis Safe MultiSig at https://autofarm.gitbook.io/autofarm-network/protocol/platform, as well as releasing the names of those who own the multisig.

Note: Capabilities for change in the contracts are not described.

Guidance:
All the contracts are immutable -- 100% OR

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

How to improve this score

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

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

> ⓘ Answer: 30%

Note: Security and risk section in their GitHub is very software language-y.

Note 2: MultiSig info is pretty clear.

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%

All AutoFarm contracts are equipped with time-locks, but no emergency pause or pause control documentation was found.

Guidance:
100% All the contracts are immutable or no pause control needed and this is explained OR
100% Pause control(s) are clearly documented and there is records of at least one test within 3 months
80% Pause control(s) explained clearly but no evidence of regular tests
40% Pause controls mentioned with no detail on capability or tests
0% Pause control not documented or explained

How to improve this score

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

Appendices

Author Details

The author of this review is Rex of DeFi Safety.

Email : rex@defisafety.com Twitter : @defisafety

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

I was coaxed to go to EthDenver 2018 and there I started SecuEth.org with Bryant and Roman. We created guidelines on good processes for blockchain code development. We got EthFoundation funding to assist in their development.

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

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

Scoring Appendix

	Total	AutoFarm	
PQ Audit Scoring Matrix (v0.7)	**Points**	**Answer**	**Points**
Total	**260**		122.15
Code and Team			**47%**
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**	70%	3.5
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**	N	0
8) Does the software function documentation fully (100%) cover the deployed contracts? (%)	**15**	0%	0
9) Are there sufficiently detailed comments for all functions within the deployed contract code (%)	**5**	33%	1.65
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**	0%	0
Security			
17) Did 3rd Party audits take place? (%)	**70**	100%	70
18) Is the bug bounty acceptable high? (%)	**10**	70%	7
Access Controls			
19) Can a user clearly and quickly find the status of the admin controls	**5**	40%	2
20) Is the information clear and complete	**10**	60%	6
21) Is the information in non-technical terms	**10**	30%	3
22) Is there Pause Control documentation including records of tests	**10**	0%	0

Section Scoring		
Code and Team	50	55%
Documentation	45	15%
Testing	50	0%
Security	80	96%
Access Controls	35	31%

Executing Code Appendix

Contract Addresses:

Binance Smart Chain: (AUTO)

- Contract address: 0xa184088a740c695e156f91f5cc086a06bb78b827
- Burn address: https://bscscan.com/token/0xa184088a740c695e156f91f5cc086a06bb78b827?a=0x000000000000000000000000000000000000dead

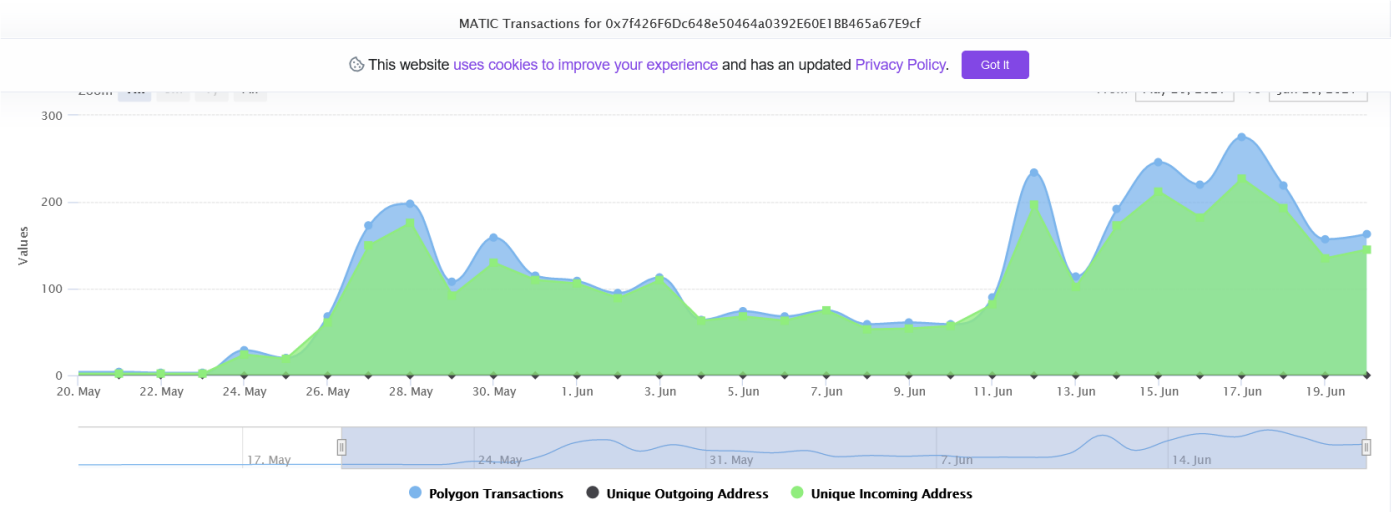
Polygon Chain: (pAUTO)

- Contract address: 0x7f426F6Dc648e50464a0392E60E1BB465a67E9cf

Klaytn Chain: (kAUTO)

- Contract address: 0x8583063110b5d29036eced4db1cc147e78a86a77

Code Used Appendix



Example Code Appendix

```
1 // For interacting with our own strategy
```

```
interface IStrategy {
    // Total want tokens managed by stratfegy
    function wantLockedTotal() external view returns (uint256);

    // Sum of all shares of users to wantLockedTotal
    function sharesTotal() external view returns (uint256);

    // Main want token compounding function
    function earn() external;

    // Transfer want tokens autoFarm -> strategy
    function deposit(address _userAddress, uint256 _wantAmt)
        external
        returns (uint256);

    // Transfer want tokens strategy -> autoFarm
    function withdraw(address _userAddress, uint256 _wantAmt)
        external
        returns (uint256);

    function inCaseTokensGetStuck(
        address _token,
        uint256 _amount,
        address _to
    ) external;
}

contract AutoFarmV2_CrossChain is Ownable, ReentrancyGuard {
    using SafeMath for uint256;
    using SafeERC20 for IERC20;

    // Info of each user.
    struct UserInfo {
        uint256 shares; // How many LP tokens the user has provided.
    }

    struct PoolInfo {
        IERC20 want; // Address of the want token.
        uint256 allocPoint; // How many allocation points assigned to this pool. AUTO to d
        uint256 lastRewardBlock; // Last block number that AUTO distribution occurs.
        uint256 accAUTOPerShare; // Accumulated AUTO per share, times 1e12. See below.
        address strat; // Strategy address that will auto compound want tokens
    }

    PoolInfo[] public poolInfo; // Info of each pool.
    mapping(uint256 => mapping(address => UserInfo)) public userInfo; // Info of each user
    uint256 public totalAllocPoint = 1; // Total allocation points. Must be the sum of all

    event Add(
        uint256 _allocPoint,
        IERC20 _want,
        bool _withUpdate,
        address _strat
```

```
    );
    event Deposit(address indexed user, uint256 indexed pid, uint256 amount);
    event Withdraw(address indexed user, uint256 indexed pid, uint256 amount);

    function poolLength() external view returns (uint256) {
        return poolInfo.length;
    }

    // Add a new lp to the pool. Can only be called by the owner.
    // XXX DO NOT add the same LP token more than once. Rewards will be messed up if you do

    function add(
        uint256 _allocPoint,
        IERC20 _want,
        bool _withUpdate,
        address _strat
    ) public onlyOwner {
        totalAllocPoint = totalAllocPoint.add(_allocPoint);
        poolInfo.push(
            PoolInfo({
                want: _want,
                allocPoint: _allocPoint,
                lastRewardBlock: 0,
                accAUTOPerShare: 0,
                strat: _strat
            })
        );

        emit Add(_allocPoint, _want, _withUpdate, _strat);
    }

    // View function to see staked Want tokens on frontend.
    function stakedWantTokens(uint256 _pid, address _user)
        external
        view
        returns (uint256)
    {
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][_user];

        uint256 sharesTotal = IStrategy(pool.strat).sharesTotal();
        uint256 wantLockedTotal =
            IStrategy(poolInfo[_pid].strat).wantLockedTotal();
        if (sharesTotal == 0) {
            return 0;
        }
        return user.shares.mul(wantLockedTotal).div(sharesTotal);
    }

    // Want tokens moved from user -> AUTOFarm (AUTO allocation) -> Strat (compounding)
    function deposit(uint256 _pid, uint256 _wantAmt) public nonReentrant {
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][msg.sender];
```

```
108
```

SLOC Appendix

Solidity Contracts

Language	Files	Lines	Blanks	Comments	Code	Complex
Solidity	28	6089	851	1290	3948	285

Comments to Code 1290/3948 = 33%

Javascript Tests

Language	Files	Lines	Blanks	Comments	Code	Complex
JavaScript	0	0	0	0	0	0

Tests to Code 0/0 = 0%