

0.7

BunnyPark Process Quality Review

Score: 23%

Overview

This is a [BunnyPark](#) Process Quality Review completed on August 9th 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 **23%**, 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.

This completed report is copyright (c) DeFiSafety 2021. Permission is given to copy in whole, retaining this copyright label.

Chain

This section indicates the blockchain used by this protocol.

 **Chain:** Binance Smart Chain

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:** 20%

They are available at website <https://www.bunnypark.com/?lang=EN>, as indicated in the [Appendix](#).

Note: BunnyPark gets a 20% here for not having a clearly displayed "Smart Contract Addresses" section in their documentation or website. In addition, you have to go into each of the various products on their website, click on a pool/farm/pair and only then will you see the "View on BSCscan" button. Therefore, BunnyPark smart contract addresses are not well-labelled nor easy to find.

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 over 10,000 transactions a day on contract *PancakePair.sol*, as indicated in the [Appendix](#).

Note: Could only check the farming pairs because the individual BunnyPark pools II have restricted "read contract" sections.

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:** No

BunnyPark has a private GitHub repository.

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%

As BunnyPark has a private GitHub repository, we could not check for a development history.

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

No public team info 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://docs.bunnypark.com/>.

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

 **Answer:** 0%

There are no basic software functions documented in the BunnyPark documentation.

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 are no software functions documented in the BunnyPark 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:** 0%

As BunnyPark has a private GitHub repository, we cannot download the source code needed to calculate the CtC.

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 are no software functions documented in the BunnyPark documentation, and therefore no traceability 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:** 0%

As BunnyPark has a private GitHub repository, we cannot download the source code needed to calculate the TtC.

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 be 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 BunnyPark code coverage in their documentation or in their audit reports by Certik and SlowMist.

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 be 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

As there is no public BunnyPark GitHub repository, we cannot check for instructions to run tests or scripts.

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%

As there is no public BunnyPark GitHub repository, we cannot check for test result reports.

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%

There is no evidence of a BunnyPark Formal Verification test in their documentations or in further web research.

16) Stress Testing environment (%)

 **Answer:** 0%

There is no evidence of any BunnyPark testnet smart contract usage 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:** 65%

SlowMist has published a BunnyPark [audit report](#) on April 27th 2021 (this was before BP's mainnet launch).

Certik has published a BunnyPark [audit report](#) on July 29th 2021, which was after the BP mainnet launch.

Note: -25% because of a private GitHub repository, = 65%.

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%

No evidence of a BunnyPark Bug Bounty program was found in their documentation or in further web searches.

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%

No admin access control information was found in the BunnyPark documentation.

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%

No admin access control information was found in the BunnyPark 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%

No admin access control information was found in the BunnyPark documentation.

Guidance:

- | | |
|------|--|
| 100% | All the contracts are immutable |
| 90% | Description relates to investments safety and updates in clear, complete non-software I 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%

There is no evidence of a Pause Control or similar functions in the BunnyPark 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.

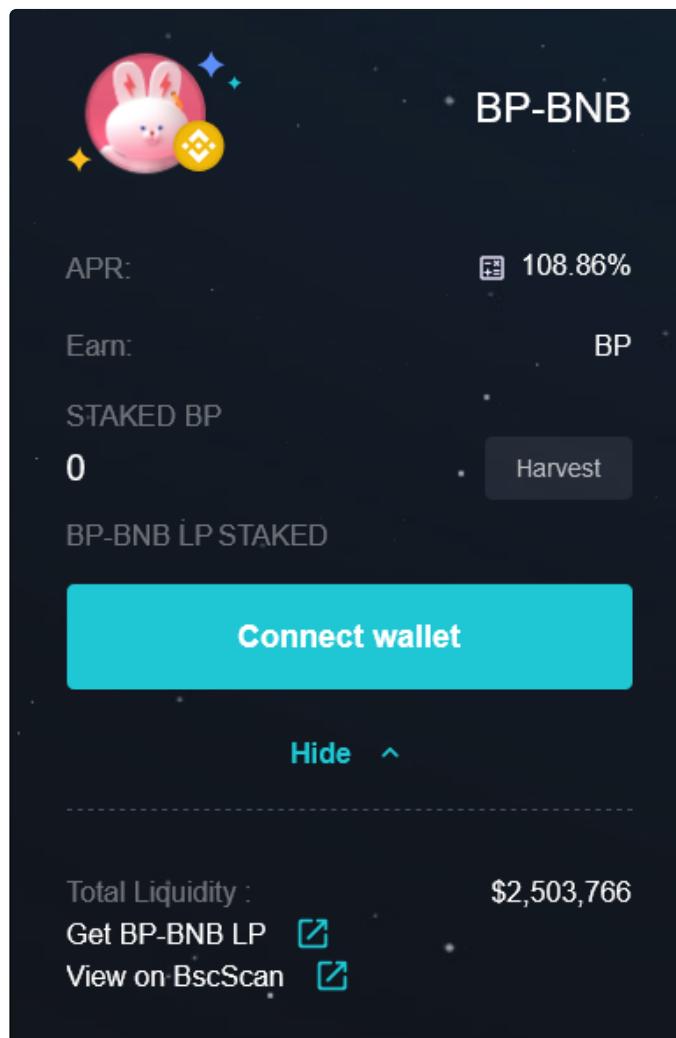
Process Quality Reviews are an extension of the SecurEth guidelines that will further increase the quality processes in Solidity and Vyper development.

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

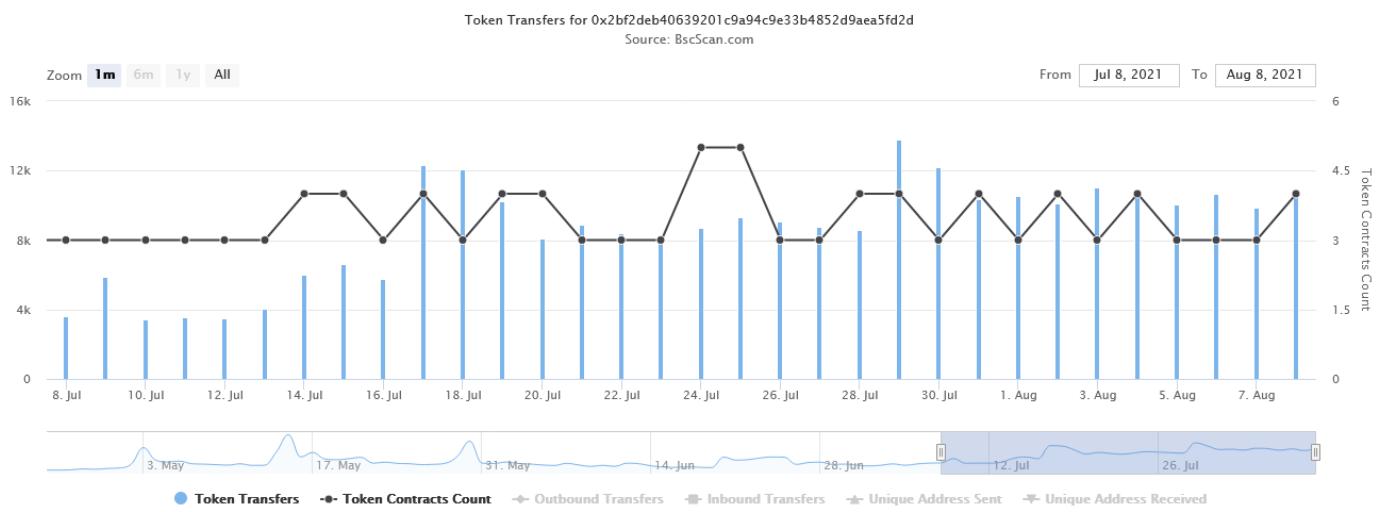
Scoring Appendix

PQ Audit Scoring Matrix (v0.7)	Total	BunnyPark	
	Points	Answer	Points
Code and Team	Total	260	59.5
1) Are the executing code addresses readily available? (%)	20	20%	4
2) Is the code actively being used? (%)	5	100%	5
3) Is there a public software repository? (Y/N)	5	N	0
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	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? (%)	5	0%	0
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	65%	45.5
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	18%	
Documentation	45	11%	
Testing	50	0%	
Security	80	57%	

Executing Code Appendix



Code Used Appendix



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

Example Code Appendix

```
1 contract PancakePair is IPancakePair, PancakeERC20 {
2     using SafeMath for uint;
3     using UQ112x112 for uint224;
4
5     uint public constant MINIMUM_LIQUIDITY = 10***3;
6     bytes4 private constant SELECTOR = bytes4(keccak256(bytes('transfer(address,uint256)')));
7
8     address public factory;
9     address public token0;
10    address public token1;
11
12    uint112 private reserve0;           // uses single storage slot, accessible via getReserves()
13    uint112 private reserve1;           // uses single storage slot, accessible via getReserves()
14    uint32 private blockTimestampLast; // uses single storage slot, accessible via getReserves()
15
16    uint public price0CumulativeLast;
17    uint public price1CumulativeLast;
18    uint public kLast; // reserve0 * reserve1, as of immediately after the most recent liquidity event
19
20    uint private unlocked = 1;
21    modifier lock() {
22        require(unlocked == 1, 'Pancake: LOCKED');
23        unlocked = 0;
24        _;
25        unlocked = 1;
26    }
27
28    function getReserves() public view returns (uint112 _reserve0, uint112 _reserve1, uint32 _blockTimestampLast) {
29        _reserve0 = reserve0;
30        _reserve1 = reserve1;
31        _blockTimestampLast = blockTimestampLast;
32    }
33
34    function _safeTransfer(address token, address to, uint value) private {
35        (bool success, bytes memory data) = token.call(abi.encodeWithSelector(SELECTOR, to));
36        require(success && (data.length == 0 || abi.decode(data, (bool))), 'Pancake: TRANSFER_FAILED');
37    }
38
39    event Mint(address indexed sender, uint amount0, uint amount1);
40    event Burn(address indexed sender, uint amount0, uint amount1, address indexed to);
41    event Swap(
42        address indexed sender,
43        uint amount0In,
44        uint amount1In,
45        uint amount0Out,
46        uint amount1Out,
47        address indexed to
48    );
49    event Sync(uint112 reserve0, uint112 reserve1);
50}
```

```

51     constructor() public {
52         factory = msg.sender;
53     }
54
55     // called once by the factory at time of deployment
56     function initialize(address _token0, address _token1) external {
57         require(msg.sender == factory, 'Pancake: FORBIDDEN'); // sufficient check
58         token0 = _token0;
59         token1 = _token1;
60     }
61
62     // update reserves and, on the first call per block, price accumulators
63     function _update(uint balance0, uint balance1, uint112 _reserve0, uint112 _reserve1) private {
64         require(balance0 <= uint112(-1) && balance1 <= uint112(-1), 'Pancake: OVERFLOW');
65         uint32 blockTimestamp = uint32(block.timestamp % 2**32);
66         uint32 timeElapsed = blockTimestamp - blockTimestampLast; // overflow is desired
67         if (timeElapsed > 0 && _reserve0 != 0 && _reserve1 != 0) {
68             // * never overflows, and + overflow is desired
69             price0CumulativeLast += uint(UQ112x112.encode(_reserve1).uqdiv(_reserve0)) * timeElapsed;
70             price1CumulativeLast += uint(UQ112x112.encode(_reserve0).uqdiv(_reserve1)) * timeElapsed;
71         }
72         reserve0 = uint112(balance0);
73         reserve1 = uint112(balance1);
74         blockTimestampLast = blockTimestamp;
75         emit Sync(reserve0, reserve1);
76     }
77
78     // if fee is on, mint liquidity equivalent to 8/25 of the growth in sqrt(k)
79     function _mintFee(uint112 _reserve0, uint112 _reserve1) private returns (bool feeOn) {
80         address feeTo = IPancakeFactory(factory).feeTo();
81         feeOn = feeTo != address(0);
82         uint _kLast = kLast; // gas savings
83         if (feeOn) {
84             if (_kLast != 0) {
85                 uint rootK = Math.sqrt(uint(_reserve0).mul(_reserve1));
86                 uint rootKLast = Math.sqrt(_kLast);
87                 if (rootK > rootKLast) {
88                     uint numerator = totalSupply.mul(rootK.sub(rootKLast)).mul(8);
89                     uint denominator = rootK.mul(17).add(rootKLast.mul(8));
90                     uint liquidity = numerator / denominator;
91                     if (liquidity > 0) _mint(feeTo, liquidity);
92                 }
93             }
94         } else if (_kLast != 0) {
95             kLast = 0;
96         }
97     }
98
99     // this low-level function should be called from a contract which performs important safety checks
100    function mint(address to) external lock returns (uint liquidity) {
101        (uint112 _reserve0, uint112 _reserve1,) = getReserves(); // gas savings
102        uint balance0 = IERC20(token0).balanceOf(address(this));
103        uint balance1 = IERC20(token1).balanceOf(address(this));

```

```

104     uint amount0 = balance0.sub(_reserve0);
105     uint amount1 = balance1.sub(_reserve1);
106
107     bool feeOn = _mintFee(_reserve0, _reserve1);
108     uint _totalSupply = totalSupply; // gas savings, must be defined here since totalsSupply is used in the mintFee calculation
109     if (_totalSupply == 0) {
110         liquidity = Math.sqrt(amount0.mul(amount1)).sub(MINIMUM_LIQUIDITY);
111         _mint(address(0), MINIMUM_LIQUIDITY); // permanently lock the first MINIMUM_LIQUIDITY tokens
112     } else {
113         liquidity = Math.min(amount0.mul(_totalSupply) / _reserve0, amount1.mul(_totalSupply) / _reserve1);
114     }
115     require(liquidity > 0, 'Pancake: INSUFFICIENT_LIQUIDITY_MINTED');
116     _mint(to, liquidity);
117
118     _update(balance0, balance1, _reserve0, _reserve1);
119     if (feeOn) kLast = uint(reserve0).mul(reserve1); // reserve0 and reserve1 are updated in the mintFee calculation
120     emit Mint(msg.sender, amount0, amount1);
121 }
122
123 // this low-level function should be called from a contract which performs important safety checks
124 function burn(address to) external lock returns (uint amount0, uint amount1) {
125     (uint112 _reserve0, uint112 _reserve1,) = getReserves(); // gas savings
126     address _token0 = token0; // gas savings
127     address _token1 = token1; // gas savings
128     uint balance0 = IERC20(_token0).balanceOf(address(this));
129     uint balance1 = IERC20(_token1).balanceOf(address(this));
130     uint liquidity = balanceOf[address(this)];
131
132     bool feeOn = _mintFee(_reserve0, _reserve1);
133     uint _totalSupply = totalSupply; // gas savings, must be defined here since totalsSupply is used in the mintFee calculation
134     amount0 = liquidity.mul(balance0) / _totalSupply; // using balances ensures pro-rata distribution
135     amount1 = liquidity.mul(balance1) / _totalSupply; // using balances ensures pro-rata distribution
136     require(amount0 > 0 && amount1 > 0, 'Pancake: INSUFFICIENT_LIQUIDITY_BURNED');
137     _burn(address(this), liquidity);
138     _safeTransfer(_token0, to, amount0);
139     _safeTransfer(_token1, to, amount1);
140     balance0 = IERC20(_token0).balanceOf(address(this));
141     balance1 = IERC20(_token1).balanceOf(address(this));
142
143     _update(balance0, balance1, _reserve0, _reserve1);
144     if (feeOn) kLast = uint(reserve0).mul(reserve1); // reserve0 and reserve1 are updated in the mintFee calculation
145     emit Burn(msg.sender, amount0, amount1, to);
146 }
147
148 // this low-level function should be called from a contract which performs important safety checks
149 function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external lock {
150     require(amount0Out > 0 || amount1Out > 0, 'Pancake: INSUFFICIENT_OUTPUT_AMOUNT');
151     (uint112 _reserve0, uint112 _reserve1,) = getReserves(); // gas savings
152     require(amount0Out < _reserve0 && amount1Out < _reserve1, 'Pancake: INSUFFICIENT_LIQUIDITY');
153
154     uint balance0;
155     uint balance1;
156     { // scope for _token{0,1}, avoids stack too deep errors

```

```

157     address _token0 = token0;
158     address _token1 = token1;
159     require(to != _token0 && to != _token1, 'Pancake: INVALID_TO');
160     if (amount0out > 0) _safeTransfer(_token0, to, amount0out); // optimistically trans
161     if (amount1out > 0) _safeTransfer(_token1, to, amount1out); // optimistically trans
162     if (data.length > 0) IPancakeCallee(to).pancakeCall(msg.sender, amount0out, amount1
163     balance0 = IERC20(_token0).balanceOf(address(this));
164     balance1 = IERC20(_token1).balanceOf(address(this));
165   }
166   uint amount0In = balance0 > _reserve0 - amount0out ? balance0 - (_reserve0 - amount0
167   uint amount1In = balance1 > _reserve1 - amount1out ? balance1 - (_reserve1 - amount1
168   require(amount0In > 0 || amount1In > 0, 'Pancake: INSUFFICIENT_INPUT_AMOUNT');
169   { // scope for reserve{0,1}Adjusted, avoids stack too deep errors
170     uint balance0Adjusted = (balance0.mul(10000).sub(amount0In.mul(25)));
171     uint balance1Adjusted = (balance1.mul(10000).sub(amount1In.mul(25)));
172     require(balance0Adjusted.mul(balance1Adjusted) >= uint(_reserve0).mul(_reserve1).mul(
173   }
174
175   _update(balance0, balance1, _reserve0, _reserve1);
176   emit Swap(msg.sender, amount0In, amount1In, amount0out, amount1out, to);
177 }
178
179 // force balances to match reserves
180 function skim(address to) external lock {
181   address _token0 = token0; // gas savings
182   address _token1 = token1; // gas savings
183   _safeTransfer(_token0, to, IERC20(_token0).balanceOf(address(this)).sub(reserve0))
184   _safeTransfer(_token1, to, IERC20(_token1).balanceOf(address(this)).sub(reserve1))
185 }
186
187 // force reserves to match balances
188 function sync() external lock {
189   _update(IERC20(token0).balanceOf(address(this)), IERC20(token1).balanceOf(address(this)))
190 }
191 }
```

SLOC Appendix

Solidity Contracts

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

Comments to Code = N/A

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