0.7

ApeSwap Process Quality Review

Score: 42%

Overview

This is a ApeSwap Process Quality Review completed on June 16th 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 42%, 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

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: 100%

They are available at website https://apeswap.gitbook.io/apeswap-finance/general-help, as well as https://github.com/ApeSwapFinance/apeswap-banana-farm, 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 labelling not clear or easy to find
0%	Executing addresses could not be found

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

> ✓ Answer: 100%

Activity is 11,000 transactions a day on contract *MasterApe.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/apeswapfinance.

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: 100%

With 207 commits and 4 branches, this is a healthy 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

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

> ⚠ Answer: No

The team is made up of exclusively aliases. https://ape-swap.medium.com/meet-the-founding-monkeys-behind-apeswap-6f837113db00.

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://apeswap.gitbook.io/apeswap-finance/.

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

✓ Answer: Yes

Very minimal software documentation was found.

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: 10%

The only function docs I found in their repository are of the Timelock.sol contract at https://github.com/ApeSwapFinance/apeswap-banana-farm/blob/master/docs/send-timelock-tx.md.

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: 25%

Code examples are in the Appendix. As per the SLOC, there is 25% 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 inherent connection between the documentation and the 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 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: 40%

Code examples are in the Appendix. As per the SLOC, there is 40% testing to code (TtC).

This score is guided by the Test to Code ratio (TtC). Generally a good test to code ratio is over 100%. However the reviewers best judgement is the final deciding factor.

Guidance:

100%	TtC > 120%	Both unit and system test visible
80%	TtC > 80%	Both unit and system test visible
40%	TtC < 80%	Some tests visible
0%		No tests obvious

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: 30%

No test coverage evident in their GitHub or audits, but there is an incomplete set of test.

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: Yes

Scripts for the farming contracts at https://github.com/ApeSwapFinance/apeswap-banana-farm/tree/master/scripts.

No run test instructions in the farm repo, but there are some in their core repo at https://github.com/ApeSwapFinance/apeswap-swap-core.

14) Report of the results (%)

⚠ Answer: 0%

No test report was found in their GitHub.

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 ApeSwap formal verification was found in their documentation or on the web.

16) Stress Testing environment (%)

⚠ Answer: 0%

No evidence of test-net smart contract address 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: 50%

Certik did a ApeSwap audit on May 6th 2021.

Gemz did a ApeSwap audit on March 5th 2021.

Note: I gave them 50% because of all the issues highlighted, half of them were never addressed, and the other half were only partially fixed. None of the issues/recommendations highlighted are fully resolved and 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/apeswap/.

Program is active and rewards 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: 0%

No admin control information was found in any of their 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 access control documentation was found.

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 access control documentation was found.

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 documentation was found in any of their docs.

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

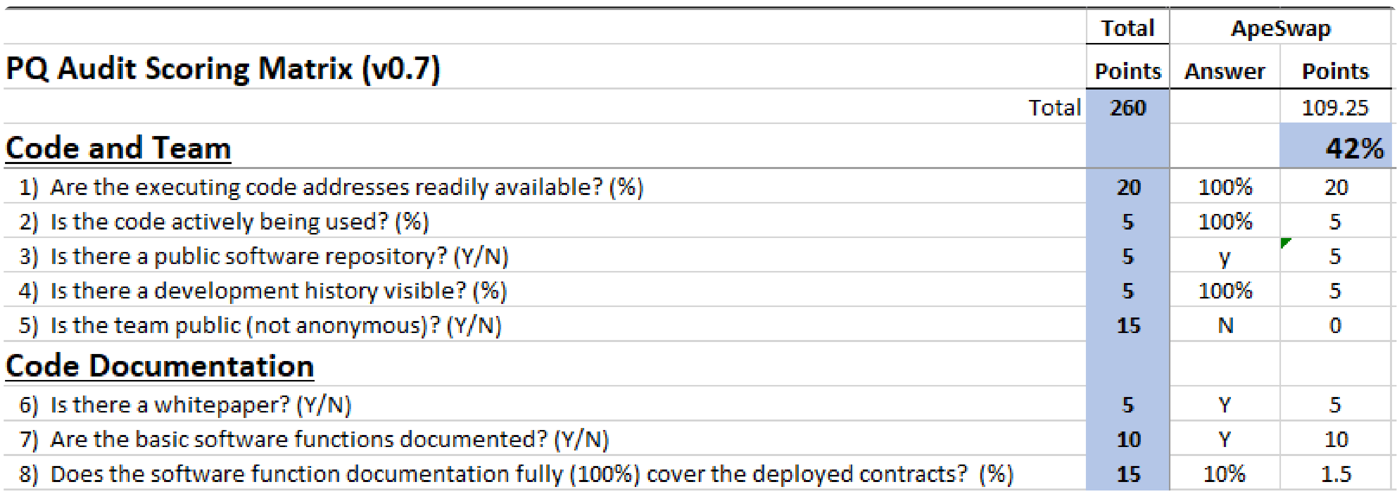
I started with Ethereum just before the DAO and that was a wonderful education. It showed the importance of code quality. The second Parity hack also showed the importance of good process. Here my aviation background offers some value. Aerospace knows how to make reliable code using quality processes.

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	ApeSwap	
PQ Audit Scoring Matrix (v0.7)	Points	Answer	Points
Total	260		109.25
Code and Team			42%
1) Are the executing code addresses readily available? (%)	20	100%	20
2) Is the code actively being used? (%)	5	100%	5
3) Is there a public software repository? (Y/N)	5	y	5
4) Is there a development history visible? (%)	5	100%	5
5) Is the team public (not anonymous)? (Y/N)	15	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	10%	1.5

9) Are there sufficiently detailed comments for all functions within the deployed contract code (%)	**5**	25%	1.25
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**	40%	8
12) Code coverage (Covers all the deployed lines of code, or explains misses) (%)	**5**	30%	1.5
13) Scripts and instructions to run the tests? (Y/N)	**5**	Y	5
14) Report of the results (%)	**10**	0%	0
15) Formal Verification test done (%)	**5**	0%	0
16) Stress Testing environment (%)	**5**	0%	0
Security			
17) Did 3rd Party audits take place? (%)	**70**	50%	35
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**	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	70%	
Documentation	45	39%	
Testing	50	29%	
Security	80	53%	
Access Controls	35	0%	

Executing Code Appendix

Key Project Information

Token : BANANA

Symbol : $BANANA

BananaToken: 0x603c7f932ED1fc6575303D8Fb018fDCBb0f39a95

MasterApe: 0x5c8D727b265DBAfaba67E050f2f739cAeEB4A6F9

SupportApe: 0x54aff400858Dcac39797a81894D9920f16972D1D

BananaSplitBar: 0x86Ef5e73EDB2Fea111909Fe35aFcC564572AcC06

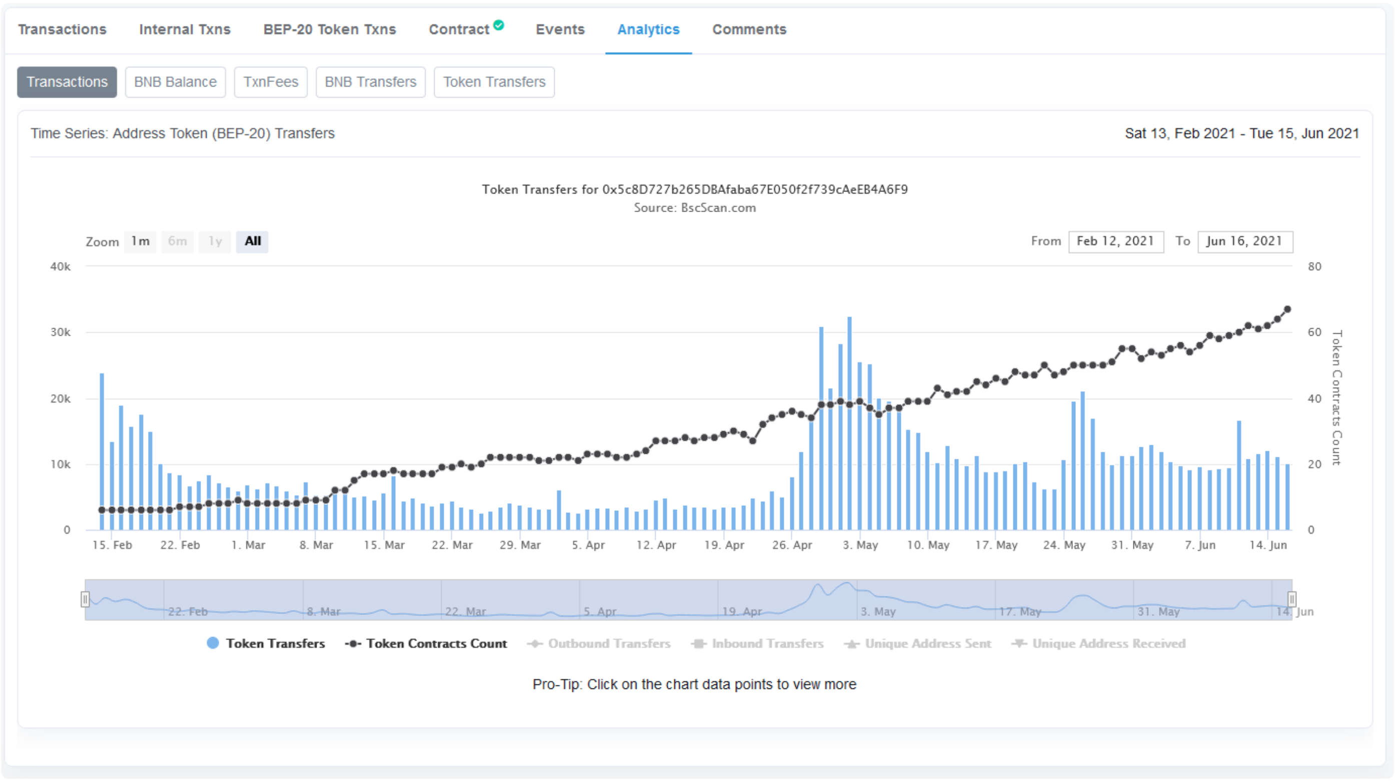
MultiCall: 0x38ce767d81de3940CFa5020B55af1A400ED4F657

Timelock: 0x2F07969090a2E9247C761747EA2358E5bB033460

ApeFactory: 0x0841BD0B734E4F5853f0dD8d7Ea041c241fb0Da6

ApeRouter: 0xC0788A3aD43d79aa53B09c2EaCc313A787d1d607

Code Used Appendix



Example Code Appendix

```
// MasterApe is the master of BANANA AND BANANASPLIT.
// He can make Banana and he is a fair guy.
//
// Note that it's ownable and the owner wields tremendous power. The ownership
// will be transferred to a governance smart contract once BANANA is sufficiently
// distributed and the community can show to govern itself.
//
// Have fun reading it. Hopefully it's bug-free. God bless.
contract MasterApe is Ownable {
    using SafeMath for uint256;
    using SafeBEP20 for IBEP20;

    // Info of each user.
    struct UserInfo {
        uint256 amount;     // How many LP tokens the user has provided.
        uint256 rewardDebt; // Reward debt. See explanation below.
        //
        // We do some fancy math here. Basically, any point in time, the amount of BANANAs
        // entitled to a user but is pending to be distributed is:
        //
        //   pending reward = (user.amount * pool.accBananaPerShare) - user.rewardDebt

        //
        // Whenever a user deposits or withdraws LP tokens to a pool. Here's what happens:
        //   1. The pool's `accCakePerShare` (and `lastRewardBlock`) gets updated.
        //   2. User receives the pending reward sent to his/her address.
        //   3. User's `amount` gets updated.
        //   4. User's `rewardDebt` gets updated.
    }

    // Info of each pool.
    struct PoolInfo {
        IBEP20 lpToken;           // Address of LP token contract.
        uint256 allocPoint;       // How many allocation points assigned to this pool. BAN
        uint256 lastRewardBlock;  // Last block number that BANANAs distribution occurs.
        uint256 accCakePerShare; // Accumulated BANANAs per share, times 1e12. See below.
    }

    // The BANANA TOKEN!
    BananaToken public cake;
    // The BANANA SPLIT TOKEN!
    BananaSplitBar public syrup;
    // Dev address.
    address public devaddr;
    // BANANA tokens created per block.
    uint256 public cakePerBlock;
    // Bonus muliplier for early banana makers.
    uint256 public BONUS_MULTIPLIER;


    // Info of each pool.
    PoolInfo[] public poolInfo;
    // Info of each user that stakes LP tokens.
    mapping (uint256 => mapping (address => UserInfo)) public userInfo;
    // Total allocation points. Must be the sum of all allocation points in all pools.
    uint256 public totalAllocPoint = 0;
    // The block number when BANANA mining starts.
    uint256 public startBlock;

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

    constructor(
        BananaToken _banana,
        BananaSplitBar _bananaSplit,
        address _devaddr,
        uint256 _bananaPerBlock,
        uint256 _startBlock,
        uint256 _multiplier
    ) public {
        cake = _banana;
        syrup = _bananaSplit;
        devaddr = _devaddr;
        cakePerBlock = _bananaPerBlock;
        startBlock = _startBlock;
        BONUS_MULTIPLIER = _multiplier;

        // staking pool
        poolInfo.push(PoolInfo({
            lpToken: _banana,
            allocPoint: 1000,
            lastRewardBlock: startBlock,
            accCakePerShare: 0
        }));

        totalAllocPoint = 1000;

    }

    modifier validatePool(uint256 _pid) {
        require(_pid < poolInfo.length, "validatePool: pool exists?");
        _;
    }

    function updateMultiplier(uint256 multiplierNumber) public onlyOwner {
        BONUS_MULTIPLIER = multiplierNumber;
    }

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

    // Detects whether the given pool already exists
    function checkPoolDuplicate(IBEP20 _lpToken) public view {
        uint256 length = poolInfo.length;
        for (uint256 _pid = 0; _pid < length; _pid++) {
            require(poolInfo[_pid].lpToken != _lpToken, "add: existing pool");
        }
    }

    // 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 d
    function add(uint256 _allocPoint, IBEP20 _lpToken, bool _withUpdate) public onlyOwner
        if (_withUpdate) {
            massUpdatePools();
        }
        checkPoolDuplicate(_lpToken);
        uint256 lastRewardBlock = block.number > startBlock ? block.number : startBlock;
        totalAllocPoint = totalAllocPoint.add(_allocPoint);
        poolInfo.push(PoolInfo({
            lpToken: _lpToken,
            allocPoint: _allocPoint,
            lastRewardBlock: lastRewardBlock,
            accCakePerShare: 0
        }));
        updateStakingPool();
    }

    // Update the given pool's BANANA allocation point. Can only be called by the owner.
    function set(uint256 _pid, uint256 _allocPoint, bool _withUpdate) public onlyOwner {
        if (_withUpdate) {
            massUpdatePools();
        }
        uint256 prevAllocPoint = poolInfo[_pid].allocPoint;
        poolInfo[_pid].allocPoint = _allocPoint;
        if (prevAllocPoint != _allocPoint) {
            totalAllocPoint = totalAllocPoint.sub(prevAllocPoint).add(_allocPoint);
            updateStakingPool();
```

SLOC Appendix

Solidity Contracts

Language	Files	Lines	Blanks	Comments	Code	Complex
Solidity	16	2454	358	431	1665	212

Comments to Code 431/1665 = 25%

Javascript Tests

Language	Files	Lines	Blanks	Comments	Code	Complex
JavaScript	7	773	83	15	675	0

Tests to Code 675/1665 = 40%