ApeSwap Finance Audit Report

Version 1.0.0

Serial No. 2021102500012016

Presented by Fairyproof

October 25, 2021

FAIRYPROOF

01. Introduction

This document includes the results of the audit performed by the Fairyproof team on the ApeSwap project.

Audit Start Time:

October 18, 2021

Audit End Time:

October 22, 2021

Audited Source Files' URL Addresses:

ApeFactory.sol: https://bscscan.com/address/0x0841BD0B734E4F5853f0dD8d7Ea041c241fb0Da6#code

ApeRouter.sol: https://bscscan.com/address/0xcF0feBd3f17CEf5b47b0cD257aCf6025c5BFf3b7#code

BananaToken: https://bscscan.com/address/0x603c7f932ED1fc6575303D8Fb018fDCBb0f39a95#code

MasterApe: https://bscscan.com/address/0x5c8D727b265DBAfaba67E050f2f739cAeEB4A6F9#code

Timelock: https://bscscan.com/address/0x2F07969090a2E9247C761747EA2358E5bB033460#code

BananaSplitBar: https://bscscan.com/address/0x86ef5e73edb2fea111909fe35afcc564572acc06#code

The audit only covered the above 6 files which mainly implement ApeSwap Finance's staking, token issurance and DEX functions.

Note: the third-party libraries this project relies on were not covered by the audit.

The goal of this audit is to review ApeSwap Finance's solidity implementation for its staking, token issurance and DEX functions, study potential security vulnerabilities, its general design and architecture, and uncover bugs that could compromise the software in production.

We make observations on specific areas of the code that present concrete problems, as well as general observations that traverse the entire codebase horizontally, which could improve its quality as a whole.

This audit only applies to the specified code, software or any materials supplied by the ApeSwap Finance team for specified versions. Whenever the code, software, materials, settings, enviroment etc is changed, the comments of this audit will no longer apply.

— Disclaimer

Note that as of the date of publishing, the contents of this report reflect the current understanding of known security patterns and state of the art regarding system security. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk.

The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. If the audited source files are smart contract files, risks or issues introduced by using data feeds from offchain sources are not extended by this review either.

Given the size of the project, the findings detailed here are not to be considered exhaustive, and further testing and audit is recommended after the issues covered are fixed.

To the fullest extent permitted by law, we disclaim all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

We do not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services.

FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

— Methodology

The above files' code was studied in detail in order to acquire a clear impression of how the its specifications were implemented. The codebase was then subject to deep analysis and scrutiny, resulting in a series of observations. The problems and their potential solutions are discussed in this document and, whenever possible, we identify common sources for such problems and comment on them as well.

The Fairyproof auditing process follows a routine series of steps:

1. Code review that includes the following
   i. Review of the specifications, sources, and instructions provided to Fairyproof to make sure we understand the size, scope, and functionality of the project's source code.
   ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Fairyproof describe.
2. Testing and automated analysis that includes the following:

    i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run the test cases.
    ii. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the source code to improve maintainability, security, and control based on the established industry and academic practices, recommendations, and research.

— Structure of the document

This report contains a list of issues and comments on all the above source files. Each issue is assigned a severity level based on the potential impact of the issue and recommendations to fix it, if applicable. For ease of navigation, an index by topic and another by severity are both provided at the beginning of the report.

— Documentation

For this audit, we used the following sources of truth about how the staking, token issuance and DEX functions should work:

https://apeswap.finance/

whitepaper

These were considered the specification.

— Comments from Auditor

No vulnerabilities with critical, high or medium-severity were found in the above source code.

Two vulnerabilities with low-severity were found in the above source code.

Additional notice: 0.

02. About Fairyproof

Fairyproof is a leading technology firm in the blockchain industry, providing consulting and security audits for organizations. Fairyproof has developed industry security standards for designing and deploying blockchain applications.

03. Introduction to ApeSwap Finance

ApeSwap is a leading decentralized exchange (DEX) on Binance Smart Chain and Polygon focused on offering a premier trading experience. Users are incentivized to pool liquidity on ApeSwap through yield farming to earn the native currency, $BANANA. Additionally, apes can use their earned $BANANA to stake and earn other tokens and unlock exclusive features. Built by DeFi apes, for DeFi apes, we have a dedicated team with years of experience who are committed to the DeFi community and growing the ApeSwap Jungle.

04. Major functions of audited code

The audited code mainly implements the following functions:

- AMM Based DEX

- A DEX like Uniswap V2
- The transaction charge is 0.2% among which one fourth is sent to the DEX's corresponding pool and the remaining three fourths is shared by the corresponding LP's providers.

- Issurance of The BANANA Token

The BANANA token is a BEP20 based token

- Name: ApeSwapFinance Banana
- Symbol: BANANA
- Precisions: 18
- Max Supply: flexible max supply
- Misc:
    - The Banana token can be used to vote in governance

- Issurance of The BANANASPLIT Token

The BANANASPLIT token is a BEP20 based token

- Name: BananaSplitBar Token
- Symbol: BANANASPLIT
- Precisions: 18
- Max Supply: flexible max supply
- Burnable: yes
- Misc:
    - The BANANASPLIT token can be used to vote in governance

Note: in ApeSwap, the BANANASPLIT token is a certificate token minted and burned by the MasterApe contract for a liquidity provider.

- MasterApe

This contract's functions are similar to Sushi's MasterChef V1's functions:

- After the contract is deployed, by default a number 0 pool will be added and the crypto token that can be staked in this pool is BANANA. After users stake their BANANA tokens in this pool, they will get the BANANASPLIT tokens accordingly. When users withdraw their BANANA tokens their BANANASPLIT tokens will be burned.

Note:

- This contract doesn't work with a token whose total supply is decreasing.
- This contract doesn't work with a scenario in which the staked token and the reward token are the same

- Timelock

This is a timelock contract like SushiSwap's Timelock.

05. Admin rights

The admin has the following previlleges:

- adding new pools for staking
- modifying the core parameters of the staking reward mechanism

06. Key points in audit

During the audit Fairyproof mainly worked on the following items:

- Integer Overflow/Underflow

We checked all the code sections, which had arithmetic operations and might introduce integer overflow or underflow if no safe libraries were used. All of them used safe libraries.

We didn't find issues or risks in these functions or areas at the time of writing.

- Access Control

We checked each of the functions that could modify a state, especially those functions that could only be accessed by "owner".

We didn't find issues or risks in these functions or areas at the time of writing.

- Admin Rights

We checked whether or not the admin had inappropriate rights to access crypto assets.

We didn't find issues or risks in these functions or areas at the time of writing.

- State Update

We checked some key state variables which should only be set at initialization.

We didn't find issues or risks in these functions or areas at the time of writing.

- Timelock

We checked whether or not `now` was appropriately used in time stampes to calculate timelocks.

We didn't find issues or risks in these functions or areas at the time of writing.

- Calculation of Precision

We checked whether or not the calculation of precision in all the division operations was properly handled.

We didn't find issues or risks in these functions or areas at the time of writing.

- Governance Security

We checked whether or not there were potential issues in the governance mechanism.

We found an issue. For more details please refer to "10 Issue descriptions".

- AMM Based DEX Security

We checked whether or not there were potential issues or risks in the AMM based DEX's mechanism.

We didn't find issues or risks in these functions or areas at the time of writing.

- Token Issurance

We checked whether or not there were potential issues or risks in the token issurance mechanisms

We found an issue. For more details please refer to "10 Issue descriptions".

07. Coverage of issues

The issues that the Fairyproof team covered when conducting the audit include but are not limited to the following ones:

- Re-entrancy Attack
- DDos Attack
- Integer Overflow
- Function Visibility
- Logic Vulnerability
- Uninitialized Storage Pointer
- Arithmetic Precision
- Tx.origin
- Shadow Variable

- Design Vulnerability
- Token Issurance
- Asset Security
- Access Control

08. Severity level reference

Every issue in this report was assigned a severity level from the following:

Critical severity issues need to be fixed as soon as possible.

High severity issues will probably bring problems and should be fixed.

Medium severity issues could potentially bring problems and should eventually be fixed.

Low severity issues are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

09. List of issues by severity

A. Critical

- N/A

B. High

- N/A

C. Medium

- N/A

D. Low

- Double-spend Risk in Voting with BANANA Or BANANASPLIT

- Issue in BANANASPLIT's Burn Function

10. Issue descriptions

- Double-spend Risk in Voting with BANANA Or BANANASPLIT: Low

Source and Description:

The `transfer` and `transferFrom` functions in both the `BananaToken.sol` and the `BananaSplitBar.sol` contract files have potential double-spend risks.

The BANANA token and the BANANASPLIT token can be used to vote in governance. A user's voting weight is positively correlated with the amount of the BANANA or BANANASPLIT tokens he/she holds. After a user transfers his/her BANANA or BANANASPLIT tokens the voting weight that corresponds to the amount of tokens transferred should be transferred as well otherwise a double-spend risk in voting may be exposed. The existing implementation doesn't transfer the token's voting weight.

Recommendation:

Consider tansferring voting weight when transferring the two tokens with `transfer` or `transferFrom`, or not enabling the voting functions with these two tokens

- Issue in BANANASPLIT's Burn Function: Low

Source and Description:

The `emergencyWithdraw` function defined in line 308 to line 315 of the `MasterApe.sol` contract file may impact the BANANASPLIT token's burn function.

In the 0 pool, after users stake their BANANA tokens they will get the same amount of the BANANASPLIT token. When they withdraw their BANANA tokens from the pool their BANANASPLIT tokens will be burned. The `emergencyWithdraw` function can be used to withdraw users' staked tokens as well. But this function doesn't burn users' BANANASPLIT tokens if it is used to withdraw users' staked tokens. Therefore malicious actors can use this function to repeat the staking and withdrawal actions to accrue the BANANASPLIT token.

Recommendation:

Consider adding a code implementation in the `emergencyWithdraw` function to burn the corresponding BANANASPLIT tokens for the withdrawn BANANA tokens.

Or consider not using the BANANASPLIT tokens that are in circulation after the contracts are deployed otherwise the DApps or services that use the BANANASPLIT tokens may suffer losses in assets.

11. Recommendations to enhance the overall security

We list some recommendations in this section. They are not mandatory but will enhance the overall security of the system if they are adopted.

- Updating Contract States Prior to Contract Interaction

The `emergencyWithdraw` function defined in line 311 to line 314 of the `MasterApe.sol` file has the following code section:

```
    pool.lpToken.safeTransfer(address(msg.sender), user.amount);
    emit EmergencyWithdraw(msg.sender, _pid, user.amount);
    user.amount = 0;
    user.rewardDebt = 0;
```

This code section does't follow the common rule of updating contract states prior to contract interacton.

Consider updating contract states prior to contract interacton.