Tetragaze

Smart contract audit

ApeSwap

Tetragaze

March 12, 2022 | v 1.0

Pass

Tetragaze Security Team has concluded that there are no issues that can have an impact on contract security. The contract is well written and is production-ready.

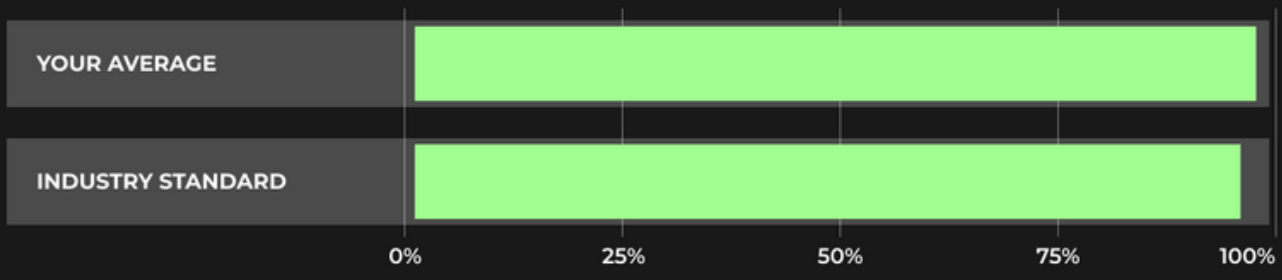
Technical summary

This document outlines the overall security of the ApeSwap smart contracts, evaluated by Tetragaze Blockchain Security team.
The scope of this audit was to analyze and document the ApeSwap smart contract codebase for quality, security, and correctness

Contract Status

There were no critical issues found during the audit.

Testable Code



Testable code is 98%, which is close to the industry standard of 95%.

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract, rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that's able to withstand the BNB Chain network's fast-paced and rapidly changing environment, we at Tetragaze recommend that the ApeSwap team put in place a bug bounty program to encourage further and active analysis of the smart contract.

Table of content

Auditing Strategy and Techniques Applied

Requirements: FIP8 spec

During the review process, we made sure to carefully check that the token contract:

- Adheres to established Token standards and ensures effective implementation;
- Ensures that documentation and code comments accurately reflect the logic and behavior of the code;
- Ensures that token distribution is in line with calculations;
- Utilizes best practices to efficiently use gas, avoiding unnecessary waste;
- Employs methods that are safe from reentrance attacks and immune to the latest vulnerabilities;
- Follows best practices for code readability

Tetragaze team of expert pentesters and smart contract developers carefully evaluated the repository for security issues, code quality, and compliance with specifications and best practices. They conducted a line-by-line review, documenting any issues that were identified during the process. Special care was taken to ensure that the review was thorough and comprehensive.

1	Due diligence in assessing the overall code quality of the codebase.	2	Cross-comparison with other, similar smart contracts by industry leaders.
3	Testing contract logic against common and uncommon attack vectors.	4	Thorough, manual review of the codebase, line-by-line.

Summary

As a decentralized exchange, ApeSwap offers numerous services and tools for users and partners to access decentralized finance opportunities. Tetragaze's audit specifically focuses on the Pairing and Factory features of ApeSwap, examining their main features and providing recommendations for improved security, performance, and potential optimizations.

Overview

Contract Analysis

The ApeSwap Factory and Pair contract combine the following features:

- Pairs between ERC20 interfaces
- Price Oracle
- Fast asset transactions
- Fees distribution to liquidity providers This audit will focus on the Pair contract which holds liquidity provider funds and the Factory contract which is used to create Pair contracts.

Price Oracle

Apeswap provides an approximate price based on the reserves of both tokens and the time since the last swap. To calculate the average price from two cumulative price observations, we take the difference between the cumulative prices at the start and end of the period and divide it by the time elapsed in seconds. Pairs contain both price0CumulativeLast (for the first token) and price1CumulativeLast (for the second token), which are ratios of reserves of token1/token0 and token0/token1 respectively. The price of token0 is expressed in terms of token1/token0, while the price of token1 is expressed in terms of token0/token1.

Liquidity Providers

ApeSwap offers an automated liquidity protocol, in which every pair of tokens on the platform has a pool of liquidity. There is a small fee of 0.3% for swapping tokens, which is split among liquidity providers according to their contribution to the pool. This fee is immediately deposited into the liquidity reserves, which increases the value of the liquidity tokens and serves as a payout to all liquidity providers based on their share of the pool. The fees are collected by burning a proportional amount of liquidity tokens, which removes a corresponding share of the underlying reserves. For traders, there will always be a 0.3% fee on all trades, with 83.3% of that fee (or 0.25% of the amount traded) going to liquidity providers and 16.6% of the fee (or 0.05% of the amount traded) going to the fee address.

Structure and Organization of Document

For ease of navigation, sections are arranged from most critical to least critical. Issues are tagged “Resolved” or “Unresolved” depending on whether they have been fixed or addressed. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

Critical

The issue affects the ability of the contract to compile or operate in a significant way.

High

The issue affects the ability of the contract to compile or operate in a significant way.

Medium

The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

Low

The issue has minimal impact on the contract’s ability to operate.

Informational

The issue has no impact on the contract’s ability to operate.

Complete Analysis

During the audit, Tetragaze experts found no Critical and Hight issues in the code of the smart contract.

L04 - Conformance to Solidity Naming Conventions

Low

There is a specific naming convention defined in Solidity that should be adhered to, with the following exceptions:

- Constant variables, symbols, and decimals are allowed to be written in lowercase.
- Private variables and unused parameters are allowed to have an underscore at the beginning of their mixed_case names.

Recomendation:

Follow the Solidity naming convention.
https://docs.soliditylang.org/en/v0.4.25/style-guide.html#naming-conventions

DSV - Different Solidity Versions

Low

Different pragma directives are used

pragma solidity =0.5.16;
pragma solidity >=0.5.0;

Recomendation:

Use one Solidity version.

Complete Analysis

During the audit, Tetragaze experts found no Critical and Hight issues in the code of the smart contract.

MZC - Missing Zero Check

Low

Certain parameters should be verified to not be equal to the zero address.

```
function initialize(address _token0, address _token1) external {
require(msg.sender == factory, 'ApeSwap: FORBIDDEN'); // sufficient check
token0 = _token0;
token1 = _token1;
}
```

Recomendation:

Check that the address is not zero.

Code Analysis and Attacks

Code analysis

The resulting code analysis and metrics is as follows:

FILE	% STMTS	% BRANCH	% FUNCS	% LINES	NOTES
exchange/	93.31	79.80	93.10	93.41	
ApeERC20.sol	100.00	80.70	100.00	100.00	
ApeFactory.sol	100.00	83.33	100.00	100.00	
ApePair.sol	86.00	90.32	82.63	87.71	
interfaces/IApeCallee.sol	86.00	90.32	85.61	83.31	
interfaces/IApeERC20.sol	86.00	90.32	84.41	80.34	
interfaces/IApeFactory.sol	93.31	79.80	93.10	93.47	
interfaces/IApePair.sol	86.04	92.52	84.87	82.02	
interfaces/IERC20.sol	87.23	90.80	88.34	89.03	
libraries/Math.sol	89. 84	90.32	87.64	81.68	
libraries/SafeMath.sol	80.87	94.34	80.35	89.67	
libraries/UQ112x112.sol	86.42	90.32	82.48	83.46	

We are thankful for the opportunity to collaborate with the ApeSwap team.

This document's statements should not be taken as investment or legal advice, and the authors cannot be held responsible for any decisions made based on them.

It is suggested by the Tetragaze Security Team that the ApeSwap team implement a bug bounty program in order to encourage external parties to scrutinize the smart contract further.