Tetragaze

Smart contract audit

Pulse Protocol™

Tetragaze

June 6, 2022 | v 2.0

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.

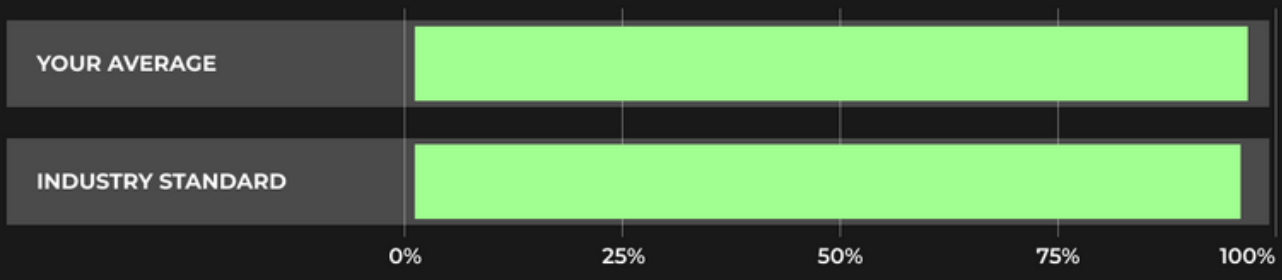
Technical summary

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

Contract Status

There were no critical issues found during the audit.

Testable Code



Testable code is 96%, 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 Ethereum network's fast-paced and rapidly changing environment, we at Tetragaze recommend that the Pulse Protocol 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

A smart contract analysis identified a minor issue related to the authority of the contract owner to transfer funds to the team's wallet. To increase security against potential hacks, a multi-wallet signing pattern can be implemented. Another option is to temporarily lock the contract or renounce ownership, which would eliminate all threats to the contract. The contract also has a maximum fee limit of 25%.

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

Unlimited Liquidity to Team Wallet

Low

The contract owner has the power to transfer unlimited amounts of funds from the contract's collected fees to the team wallet by utilizing the withdrawAllToTreasury() method.

```
function withdrawAllToTreasury() external swapping onlyOwner {

    uint256 amountToSwap =
_gonBalances[address(this)].div(_gonsPerFragment);
    require( amountToSwap > 0,"There is no Pulse token deposited
in token contract");
    address[] memory path = new address[](2);
    path[0] = address(this);
    path[1] = router.WETH();
    router.swapExactTokensForETHSupportingFeeOnTransferTokens(
        amountToSwap,
        0,
        path,
        treasuryReceiver,
        block.timestamp
    );
}
```

Recomendation:

The contract may include a limit on the amount of funds that can be exchanged to prevent instability in the token's price. It is important for the team to properly secure the owner's account by implementing a strong security system and preventing a single user from gaining access to contract admin functions. To mitigate this risk, the contract can be temporarily locked or ownership can be renounced.

Complete Analysis

Manipulate Total Supply

Low

The owner has the ability to alter the total supply of the token, which will subsequently affect the token price and market capitalization.

```
for (uint256 i = 0; i < times; i++) {
    _totalSupply = _totalSupply
        .mul((10**RATE_DECIMALS).add(rebaseRate))
        .div(10**RATE_DECIMALS);
}
```

Recomendation:

The contract owner should pay careful attention to managing the circulating supply of the token by adjusting it according to the price fluctuations of the token.

Complete Analysis

Public Function could be Declared External

Low

To save gas, public functions that are never called by the contract should be declared as external.

```
getLiquidityBacking
updatePancakeSwapRouter
decimals
symbol
name
transferOwnership
renounceOwnership
owner
```

Recomendation:

Functions that are not called within the contract should be marked with the external attribute.

Complete Analysis

State Variables could be Declared Constant

Low

Declaring constant state variables as constant will help to conserve gas.

```
totalFee
swapEnabled
feeDenominator
dividendsPerShareAccuracyFactor
```

Recomendation:

State variables that never change should be given the constant attribute.

Complete Analysis

Conformance to Solidity Naming Conventions

Low

A naming convention is defined by Solidity, which should be adhered to. However, there are some exceptions to this rule:

- Constant variables, symbols, and decimals may be written in lowercase.
- Private variables and unused parameters may begin with an underscore (_) in mixed_case.

```
_totalSupply
_lastAddLiquidityTime
_lastRebasedTime
_initRebaseStartTime
_autoAddLiquidity
_autoRebase
burnPitFee
sellFee
pulseDividendFee
...
```

Recomendation:

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

Complete Analysis

Missing Events Arithmetic

Low

There are certain critical arithmetic parameters for which events have not been detected, making it difficult to track changes made outside of the blockchain. These functions do not emit any events, making it challenging to accurately track the changes.

```
minPeriod = _minPeriod
```

Recomendation:

Emit an event for critical parameter changes.

Divide before Multiply Operation

Low

If divisions are performed before multiplications, it may result in a loss of accuracy in predictions.

```
liquidityBalance = _gonBalances[pair].div(_gonsPerFragment)
times = deltaTime.div(900)
BUSD.transfer(addr_taxes,amount / 100 * 40)
BUSD.transfer(shareholder,amount / 100 * 60)
```

Recomendation:

The multiplications should be prior to the divisions.

We are thankful for the opportunity to collaborate with the Pulse Protocol 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 Pulse Protocol team implement a bug bounty program in order to encourage external parties to scrutinize the smart contract further.