PLUTEUS

Smart Contract Audit Report

June 11, 2021

IMMUNE BYTES

Introduction

1. About Pluteus

PLUTEUS Token (PLUTEUS)

BEP-20 deflationary token on Binance Smart Chain

Initial supply: 1 Trillion tokens

FAIR LAUNCH
Fair Launch on DxSale.com - team keeps no portion of tokens and needs to participate on Fair Launch as well
Trading pair set up on PancakeSwap with Locked Liquidity

BURNS
Initial burn: 25% of Initial supply
Manual burns throughout the initial phase of the project in order to boost awareness and value for holders
Burn of tokens - irreversible and publicly available. Burnt tokens are subtracted from Total supply

TOKENOMICS
There is a 9% fee on each transaction as an anti-dump`n pump measure encouraging investors to long-term hold.

The 9% fee is consisting of:

1) 4% Reflection to PLUTEUS holder wallets based on their percentage portion of Total supply
2) 4% Liquidity pool creation:
- half of this amount will be converted by contract to BNB
- other half will be locked together with the first half on PancakeSwap Liquidity pool
- leftover portion created due to price decrease after BNB purchase will be sent to Pluteus Fund
3) 1% Pluteus Fund - held on Multisig wallet with 48-hours lock on outgoing transactions. The funds will be splitted as follows:
0,5% of funds received to be donated to partnering charity projects
0,5% of funds received to be used to cover team, development and marketing costs

SECURITY
Smart contract of PLUTEUS token was designed based on state-of-the-art technology to provide as much security as possible and stay scalable at the same time.

Liquidity sent to PancakeSwap will be locked for a minimum of 6 months.

Contract owner wallet will be kept secured and in form of a Multisignature wallet with 48-hours timelock on outgoing transactions in order to prevent unauthorised spending, leakage and hacking. The wallet transactions will be publicly available.

Whale-proof - the smart contract will decline any transaction where one wallet would exceed the 1% of Total supply threshold

Rugpull-proof - locked Liquidity, transparent use of funds and time-locks provide top-notch transparency against rugpulls

2. About ImmuneBytes

ImmuneBytes is a security start-up to provide professional services in the blockchain space. The team has hands-on experience in conducting smart contract audits, penetration testing, and security consulting. ImmuneBytes's security auditors have worked on various A-league projects and have a great understanding of DeFi projects like AAVE, Compound, 0x Protocol, Uniswap, dydx.

The team has been able to secure 15+ blockchain projects by providing security services on different frameworks. ImmuneBytes team helps start-up with a detailed analysis of the system ensuring security and managing the overall project.

Visit http://immunebytes.com/ to know more about the services.

Documentation Details

The team has provided a description of the Smart Contract over the chat on UpWork.

Audit Process & Methodology

ImmuneBytes team has performed thorough testing of the project starting with analyzing the code design patterns in which we reviewed the smart contract architecture to ensure it is structured and safe use of third-party smart contracts and libraries.

Our team then performed a formal line-by-line inspection of the Smart Contract in order to find any potential issues like Signature Replay Attacks, Unchecked External Calls, External Contract Referencing, Variable Shadowing, Race conditions, Transaction-ordering dependence, timestamp dependence, DoS attacks, and others.

In the Unit testing phase, we run unit tests written by the developer in order to verify the functions work as intended. In Automated Testing, we tested the Smart Contract with our in-house developed tools to identify vulnerabilities and security flaws.

The code was audited by a team of independent auditors which includes -

1. Testing the functionality of the Smart Contract to determine proper logic has been followed throughout.
2. Analyzing the complexity of the code by thorough, manual review of the code, line-by-line.
3. Deploying the code on testnet using multiple clients to run live tests.
4. Analyzing failure preparations to check how the Smart Contract performs in case of bugs and vulnerabilities.
5. Checking whether all the libraries used in the code are on the latest version.
6. Analyzing the security of the on-chain data.

Audit Details

- Project Name: Pluteus
- Languages: Solidity(Smart contract)
- Github commit hash for audit: 6499d42f3de8414829ec076fcb07077983c905a6
- Platforms and Tools: Remix IDE, Truffle, Truffle Team, Ganache, Solhint, VScode, Contract Library, Slither, SmartCheck

Audit Goals

The focus of the audit was to verify that the smart contract system is secure, resilient, and working according to its specifications. The audit activities can be grouped into the following three categories:

1. Security: Identifying security-related issues within each contract and within the system of contracts.
2. Sound Architecture: Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.
3. Code Correctness and Quality: A full review of the contract source code. The primary areas of focus include:
    a. Correctness
    b. Readability
    c. Sections of code with high complexity
    d. Quantity and quality of test coverage

Security Level References

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

Admin/Owner Privileges can be misused either intentionally or unintentionally.

High severity issues will 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.

Issues	High	Medium	Low
Open	2	-	3
Closed	-	-	-

Admin/Owner Privileges

The **admin/owner** of **Pluteus** smart contract has various privileges over the smart contract. These privileges can be misused either intentionally or unintentionally (in case admin's private key gets hacked). We assume that these extra rights will always be used appropriately. Some of these admin rights are listed below.

1. **In the Pluteus contract the owner address can exclude any ethereum address from rewards.**
   The **Pluteus** contract contains an **excludeFromReward()** function by which the **owner** account can exclude any ethereum address from rewards.

2. **In the Pluteus contract the owner address can include any ethereum address to receive rewards.**
   The **Pluteus** contract contains an **includeInReward()** function by which the **owner** account can include any ethereum address for rewards.

3. **In the Pluteus contract the owner address can include and exclude any ethereum address from fee deduction.**
   The **Pluteus** contract contains **excludeFromFee**() and **includeInFee**() functions by which the **owner** account can include and exclude any ethereum address from fee deductions.

4. **The owner of Pluteus contract can anytime update the fee, liquidityFee and max allowed transfer amount.**
   The **Pluteus** contract contains **setTaxFeePercent**(), **setLiquidityFeePercent**() and **setMaxTxPercent**() functions by which the **owner** account can update the **_taxFee**, **_liquidityFee** and **_maxTxAmount** respectively.

5. **The owner of Pluteus contract can anytime update the *swapAndLiquifyEnabled* status.**
   The **Pluteus** contract contains the **setSwapAndLiquifyEnabled**() function by which the **owner** account can update the **swapAndLiquifyEnabled** boolean status.

Recommendation:
Consider hardcoding predefined ranges or validations for input variables in privileged access functions. Also consider adding some governance for admin rights for smart contracts or use a multi-sig wallet as admin/owner address.

High Severity Issues

1. **Non standard interface of ERC20 Token used.**
   The **Pluteus** smart contract inherits a non standard interface for **ERC20** or **BEP20** standard. The **transferFrom**() function of **Pluteus** token returns an **address** value as opposed to a **boolean** value of a standard **ERC20**.

```
function transferFrom(address sender, address recipient, uint256 amount) external returns (address);
```

   The implementation of **transferFrom**() function returns an **address** as well due to which any external smart contract (like Uniswap's) won't be able to interact with **Pluteus** token.

   Recommendation:
   Consider following the ERC20 token standard described here https://eips.ethereum.org/EIPS/eip-20.

2. **Incorrect PancakeSwap interfaces used.**
   The **Pluteus** smart contract uses incorrect pancakeswap smart contract interfaces which creates multiple issues.
   Since the incorrect interfaces are used in **Pluteus** contract's **constructor**() for creation of a new LP pair, the contract deployment transaction fails. So the contract cannot be deployed on blockchain.
   The incorrect interfaces and functions are also used in other functions of **Pluteus** contract like **swapTokensForBnb**() and **addLiquidity**(). So transactions to these functions will also fail.
   The incorrect use of PancakeSwap interfaces makes the **Pluteus** contract non deployable and non usable.

   Recommendation:
   Consider using the interfaces provided here
   https://github.com/pancakeswap/pancake-swap-core/tree/master/contracts/interfaces
   https://github.com/pancakeswap/pancake-swap-periphery/tree/master/contracts/interfaces

Low Severity Issues

1. **Incorrect parameters used in the OwnershipTransferred event.**
   In the **lock**() function of **Ownable** smart contract incorrect parameters are used in emitting the **OwnershipTransferred** event.

```
        _previousOwner = _owner;
        _owner = address(0);
        _lockTime = block.timestamp + time;
```

   Here the **_owner** variable used in **OwnershipTransferred** is already set to **zero** address. So it will always seem like the ownership was transferred from **zero** address to **zero** address which is incorrect.

   Recommendation:
   Consider emitting the event before updating the **_owner** variable.

2. **lock() and unlock() functions can be called several times by the owner.**
   In **Pluteus** smart contract, after the initial (planned) lock and unlock period these functions can still be called one after the other by the owner.
   Also after the lock period when the contract gets unlocked the **unlock**() function can still be called any number of times by the contract owner. The preconditions in **unlock**() function only checks that the caller is the **_previousOwner** of the contract which is always true.

   Recommendation:
   Consider reseting the **_previousOwner** address to **address(0)** in the **unlock**() function.

3. **Spelling mistake in geUnlockTime() function's name.**
   The **geUnlockTime**() function in **Ownable** smart contract is spelled incorrectly. This function is declared as a public function so it could be used by other smart contract and off-chain scripts.

   Recommendation:
   Consider correcting the name of the function.

Automated Audit

Solhint Linting Violations

Solhint is an open-source project for linting solidity code, providing both security and style guide validations. It integrates seamlessly into most mainstream IDEs. We used Solhint as a plugin within our VScode for this analysis. Multiple Linting violations were detected by Solhint, it is recommended to use Solhint's npm package to lint the contracts.

Slither

Slither, an open-source static analysis framework. This tool provides rich information about Ethereum smart contracts and has critical properties. While Slither is built as a security-oriented static analysis framework, it is also used to enhance the user's understanding of smart contracts, assist in code reviews, and detect missing optimizations.

The concerns slither raises have already been covered in the manual audit.

```
Compilation warnings/errors on ./Pluteus.sol:
Warning: Contract code size exceeds 24576 bytes (a limit introduced in Spurious Dragon). This contract may not be dep
loyable on mainnet. Consider enabling the optimizer (with a low "runs" value!), turning off revert strings, or using
libraries.
   --> ./Pluteus.sol:680:1:
    |
680 | contract Pluteus is Context, IERC20, Ownable, ITimelockable {
    | ^ (Relevant source part starts here and spans across multiple lines).


INFO:Detectors:
Pluteus.withdrawLeftoverBnb(address) (Pluteus.sol#1211-1215) sends eth to arbitrary user
        Dangerous calls:
        - address(to).transfer(leftover) (Pluteus.sol#1213)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to-arbitrary-desti
nations
INFO:Detectors:
Reentrancy in Pluteus._transfer(address,address,uint256) (Pluteus.sol#1066-1102):
        External calls:
        - swapAndLiquify(contractTokenBalance) (Pluteus.sol#1092)
                - pancakeswapV2Router.addLiquidityBNB{value: bnbAmount}(address(this),tokenAmount,0,0,address(this),b
lock.timestamp) (Pluteus.sol#1141-1148)
                - pancakeswapV2Router.swapExactTokensForBNBSupportingFeeOnTransferTokens(tokenAmount,0,path,address(t
his),block.timestamp) (Pluteus.sol#1121-1127)
        External calls sending eth:
        - swapAndLiquify(contractTokenBalance) (Pluteus.sol#1092)
                - pancakeswapV2Router.addLiquidityBNB{value: bnbAmount}(address(this),tokenAmount,0,0,address(this),b
lock.timestamp) (Pluteus.sol#1141-1148)
        State variables written after the call(s):
        - _tokenTransfer(from,to,amount,takeFee) (Pluteus.sol#1101)
                - _rOwned[PLUTEUS_ADDRESS] = _rOwned[PLUTEUS_ADDRESS] + rPluteus (Pluteus.sol#859)
                - _rOwned[address(this)] = _rOwned[address(this)].add(rLiquidity) (Pluteus.sol#1015)
                - _rOwned[sender] = _rOwned[sender].sub(rAmount) (Pluteus.sol#1181)
                - _rOwned[sender] = _rOwned[sender].sub(rAmount) (Pluteus.sol#1171)
                - _rOwned[sender] = _rOwned[sender].sub(rAmount) (Pluteus.sol#1193)
                - _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount) (Pluteus.sol#1172)
                - _rOwned[sender] = _rOwned[sender].sub(rAmount) (Pluteus.sol#924)
                - _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount) (Pluteus.sol#1183)
```

Concluding Remarks

While conducting the audits of the Pluteus smart contract, it was observed that the contracts contain High, and Low severity issues, along with several areas of recommendations.

Our auditors suggest that High and Low severity issues should be resolved by the developers. Resolving the areas of recommendations are up to the team's discretion. The recommendations given will improve the operations of the smart contract.

Disclaimer

ImmuneBytes's audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Our team does not endorse the Pluteus platform or its product; neither this audit is investment advice.
Notes:

- Please make sure contracts deployed on the mainnet are the ones audited.
- Check for the code refactor by the team on critical issues.

ImmuneBytes Pvt Ltd.