SECURE BE.

BUILD TRUST IN YOUR PROJECT WITH OUR AUDIT

Table Of Content

1 Summary

2 Overview

3 — 4 Vulnerability Check

5 — 6 Owner Privileges

Summary

This report has been prepared for World Pay Token to discover issues and vulnerabilities in the source code of the World Pay Token project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilising Static Analysis and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from Medium to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

Overview

PROJECT NAME	World Pay Token (WPAY)
ADDRESS	0x1c2E57525cec4a791FeDD360475d921f1911f104
NETWORK	Binance Smart Chain
TOTAL SUPPLY	1,000,000,000 WPAY
DECIMALS	18
COMPILER VER.	v0.8.19+commit.7dd6d404
LANGUAGE	Solidity

BUY FEE 10%

●	0-5%
●	5-10%
●	<10%

SELL FEE 10%

✓ **Optimization With 200 Runs**

Wallets

OWNER	0x034cb55be505ab072c6cfed435964ee119d77e0a
MARKETING	0x0e7D4Bd3A238358fdF25b48978DEB37b069d3162
TREASURY	0x7BEA22e5a3433bD0BF6a645e3a509078365D3037

Vulnerability Summary

SECURITY SCORING: 90 / 100

Vulnerability Check

Code Review

DESIGN LOGIC	PASSED
COMPILER WARNINGS	PASSED
PRIVATE USER DATA LEAKS	PASSED
TIMESTAMP DEPENDENCE	PASSED
INTEGER OVERFLOW AND UNDERFLOW	PASSED
RACE CONDITION REENTRANCY	PASSED
POSSIBLE DELAYS IN DATA DELIVERY	PASSED
ORACLE CALLS	PASSED
FRONT RUNNING	PASSED
DOS WITH BLOCK GAS LIMIT	PASSED
DOS WITH REVERT	PASSED
METHODS EXECUTION PERMISSIONS	PASSED
ECONOMY MODEL	PASSED
IMPACT OF THE EXCHANGE RATE	PASSED
MALICIOUS EVENT LOG	PASSED
SCOPING AND DECLARATIONS	PASSED
UNINITIALIZED STORAGE POINTERS	PASSED
ARITHMETIC ACCURACY	PASSED
CROSS FUNCTION RACE CONDITIONS	PASSED
SAFE ZEPPELIN MODULE	PASSED
FALLBACK FUNCTION SECURITY	PASSED

Vulnerability Check

Function Review

BUSINESS LOGICS REVIEW FUNCTIONALITY CHECKS	PASSED
ACCESS CONTROL & AUTHORIZATION	PASSED
ESCROW MANIPULATION	PASSED
TOKEN SUPPLY MANIPULATION	PASSED
ASSETS INTEGRITY	PASSED
USER BALANCES MANIPULATION	PASSED
DATA CONSISTENCY MANIPULATION	PASSED
KILL - SWITCH MECHANISM OPERATION TRAILS & EVENT GENERATION	PASSED

Owner Privileges

```
  function enableTrading(uint256 deadBlocks) external onlyOwner {
     require(!tradingActive, "Cannot reenable trading");
     tradingActive = true;
     swapEnabled = true;
     tradingActiveBlock = block.number;
     blockForPenaltyEnd = tradingActiveBlock + deadBlocks;
     emit EnabledTrading();
  }
```

```
   function updateBuyFees(uint256 _TreasuryFee, uint256 _liquidityFee,
      uint256 _MarketingFee, uint256 _burnFee) external onlyOwner {
      buyTreasuryFee = _TreasuryFee;
      buyLiquidityFee = _liquidityFee;
      buyMarketingFee = _MarketingFee;
      buyBurnFee = _burnFee;
                      buyTotalFees   =   buyTreasuryFee   +   buyLiquidityFee   +
buyMarketingFee + buyBurnFee;
      require(buyTotalFees <= 10, "Must keep fees at 10% or less");
   }

   function updateSellFees(uint256 _TreasuryFee, uint256 _liquidityFee,
      uint256 _MarketingFee, uint256 _burnFee) external onlyOwner {
      sellTreasuryFee = _TreasuryFee;
      sellLiquidityFee = _liquidityFee;
      sellMarketingFee = _MarketingFee;
      sellBurnFee = _burnFee;
       sellTotalFees = sellTreasuryFee + sellLiquidityFee + sellMarketingFee
+ sellBurnFee;
      require(sellTotalFees <= 10, "Must keep fees at 10% or less");
```

```
  function BoughtEarly(address[] calldata wallets, bool flag) external
      onlyOwner {
      for(uint256 i = 0; i < wallets.length; i++){
         boughtEarly[wallets[i]] = flag;
      }
   }
```

Owner Privileges

```
function manageBoughtEarly(address wallet, bool flag) external
   onlyOwner {
   boughtEarly[wallet] = flag;
}
```

CONCLUSION

✓ Owner can't mint tokens	✓ Owner can't stop the contract
✓ Owner can't limit transactions	✗ Owner can't stop trading
✓ Owner can't set fees <25%	✗ Owner can't block wallets