WadzPayToken Contract Final Audit Report

Overview

Scope of Audit

The scope of this audit was to analyze and document the staking smart contract codebase for quality, security, and correctness.

Check Vulnerabilities

- Re-entrancy
- Timestamp Dependence

- Gas Limit and Loops
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Byte array
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Redundant fallback function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- Unsafe type inference
- Implicit visibility level

Techniques and Methods

Throughout the audit of smart contracts, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behavior.
- Token distribution and calculations are as per the intended behavior mentioned in the whitepaper.
- Implementation of ERC-20 token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods, and tools were used to review all the smart contracts.

Structural Analysis

In this step, we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

A static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual Analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behavior of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit

Remix IDE, Truffle, Truffle Team, Solhint, Mythril, Slither, Solidity statistic analysis.

Issue Categories

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below.

High Severity Issues

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract’s performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium Severity Issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low Severity Issues

Low-level severity issues can cause minor impact and or are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational Issues

These are four severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Number of security issues per severity.

TYPE	HIGH	MEDIUM	LOW	INFORMATIONAL
Open	1	0	0	0
Acknowledged	0	0	0	0
Closed	0	2	3	2

Introduction

During the period of **November 14, 2021, to November 16, 2021** - ImmuneBytes Team performed a security audit for **WadzPayToken** smart contract.

Issues Found – Code Review / Manual Testing

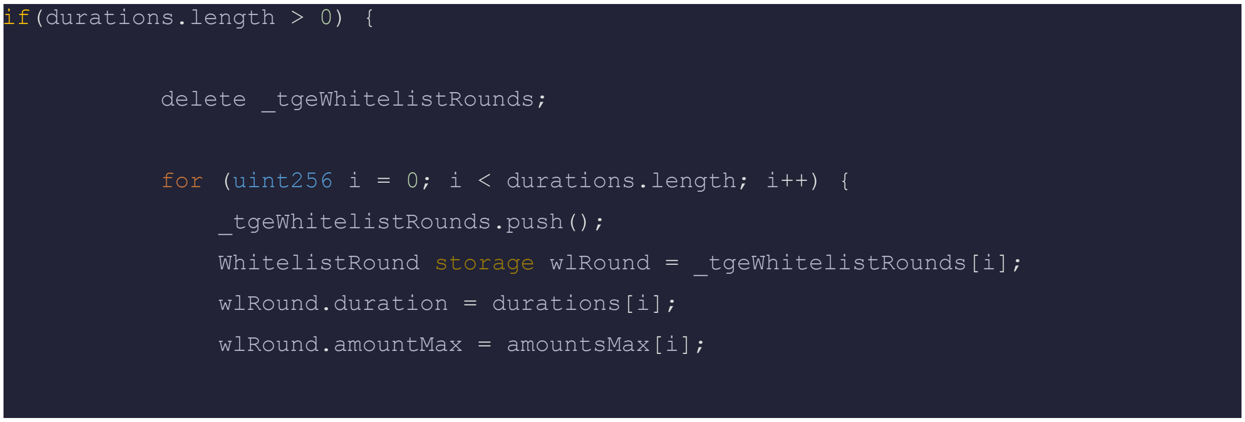
High Severity Issues

- **[H1] createTGEWhitelist has access to unassigned variables.**

  Online no 869 the function access the non assigned value of _tgeWhitelistRounds.
  It throws the VM expectation when we try to execute createTGEWhitelist.

  _tgeWhitelistRounds can't be accessed directly. We recommend to push the empty object first then use the space.

Recsommendation:



```
if(durations.length > 0) {

        delete _tgeWhitelistRounds;

        for (uint256 i = 0; i < durations.length; i++) {
            _tgeWhitelistRounds.push();
            WhitelistRound storage wlRound = _tgeWhitelistRounds[i];
            wlRound.duration = durations[i];
            wlRound.amountMax = amountsMax[i];
```

```
        }

    }
```

Recommendation transaction KOVAN

Status: open

Medium Severity Issues

- **[M1] Make the deployer address to be MultiSig**

  We recommend that the deployer address used should be MultiSig. As this remove the centralization nature from the contract.

  Status: closed

- **[M2] Function implementation is missing**

  The ***_beforeTokenTransfer*** function implementation is missing.
  As there are calls for the *_beforeTokenTransfer* and the function body is missing.

  We recommend to implement the function body or just remove the function if there is no use of it.

  Status: closed

Low Severity Issues

- **[L1] Invalid percent assignment**

  The default value of maxTxPercent should be 100 to make it more readable and consistent. The same change should be made on line no 709 and the division should be mad ebay 100 instead of 1000

  Status: Closed

- **[L2] Unnecessary use of safeMath**

  The SafeMath library is not needed as the library in already been incorporated in the 0.8 compiler onwards. We recommend to remove the same library from the code base.

  Status: Closed

- **[L3] Used locked pragma version**

  The pragma versions used in the contract are not locked. Consider using the latest versions among 0.8.10 for deploying the contracts and libraries as it does not compile for any other version and can be confusing for a developer. Solidity source files indicate the versions of the compiler they can be compiled with.

  pragma solidity ^0.8.0; // bad: compiles between 0.8.0 and 0.8.10
  pragma solidity 0.8.0; // good : compiles w 0.8.0 only but not the latest version
  pragma solidity 0.8.10; // best: compiles w 0.8.10

  Status: Closed

Informational Issues

- **[INF1] use mixedCase for naming conventions**

  mixedCase is used for the naming conventions. Some of the functions are missing that. We recommend using the below functions.

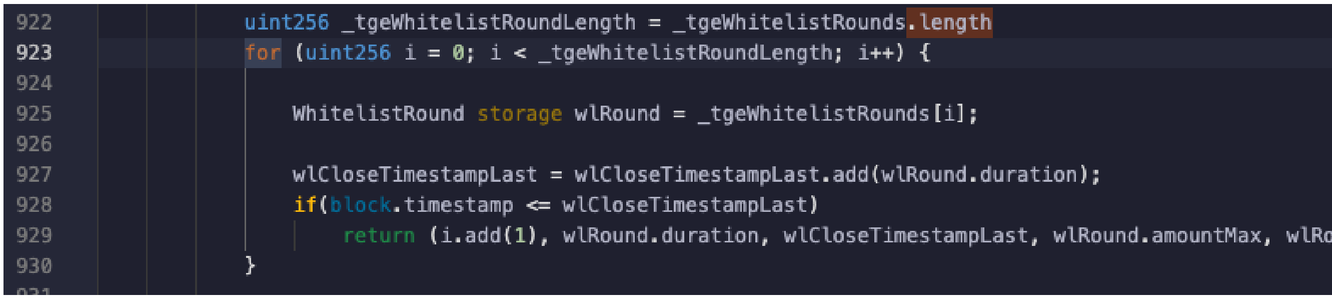
  Function **-** addBlacklist, removeBlacklist

  Status: closed

- **[INF2] Use local variable declaration**

  On line no 923 the _tgeWhitelistRounds.length is used in the for loop so by using this the storage read operation will be performed again and again.

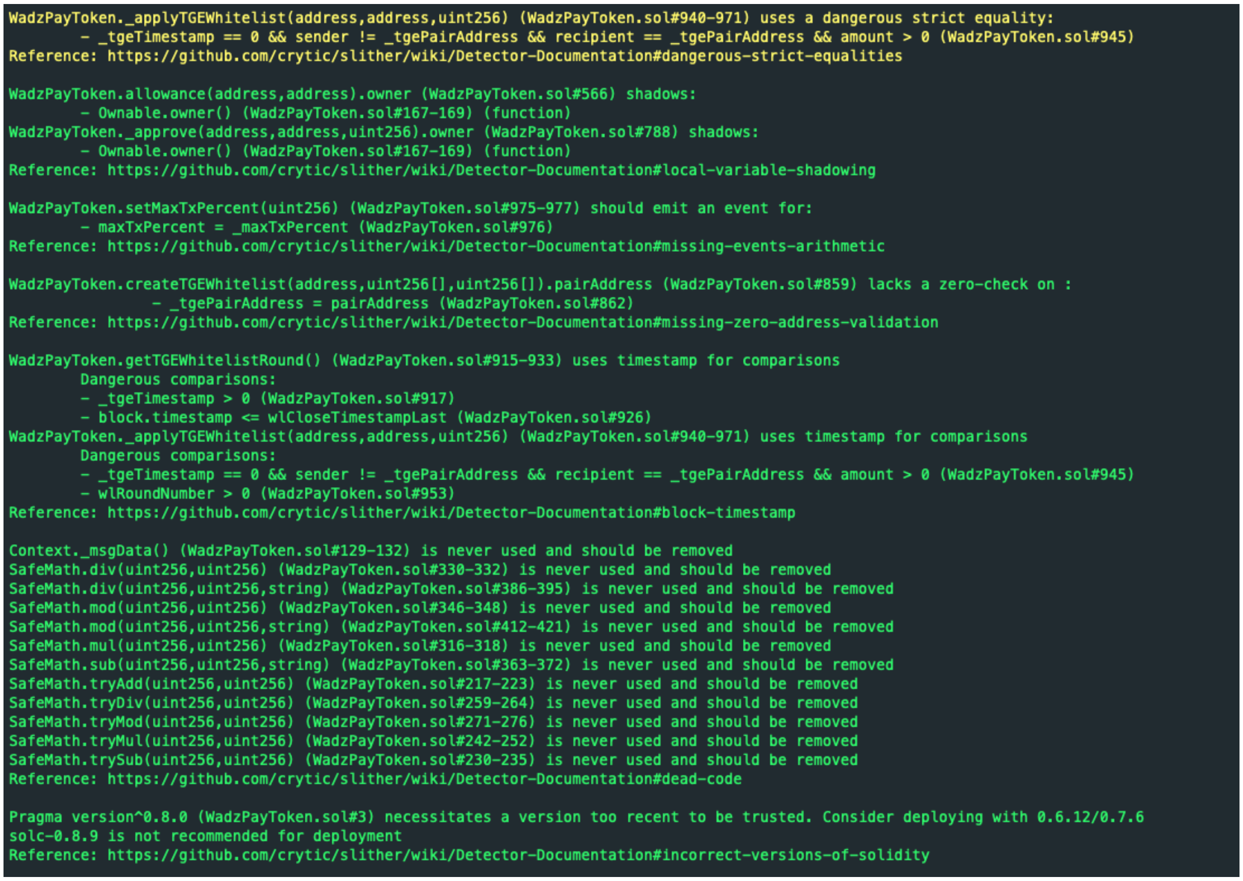
  We recommend to use the local variable and store the _tgeWhitelistRounds.length such that read operation gets minimized.



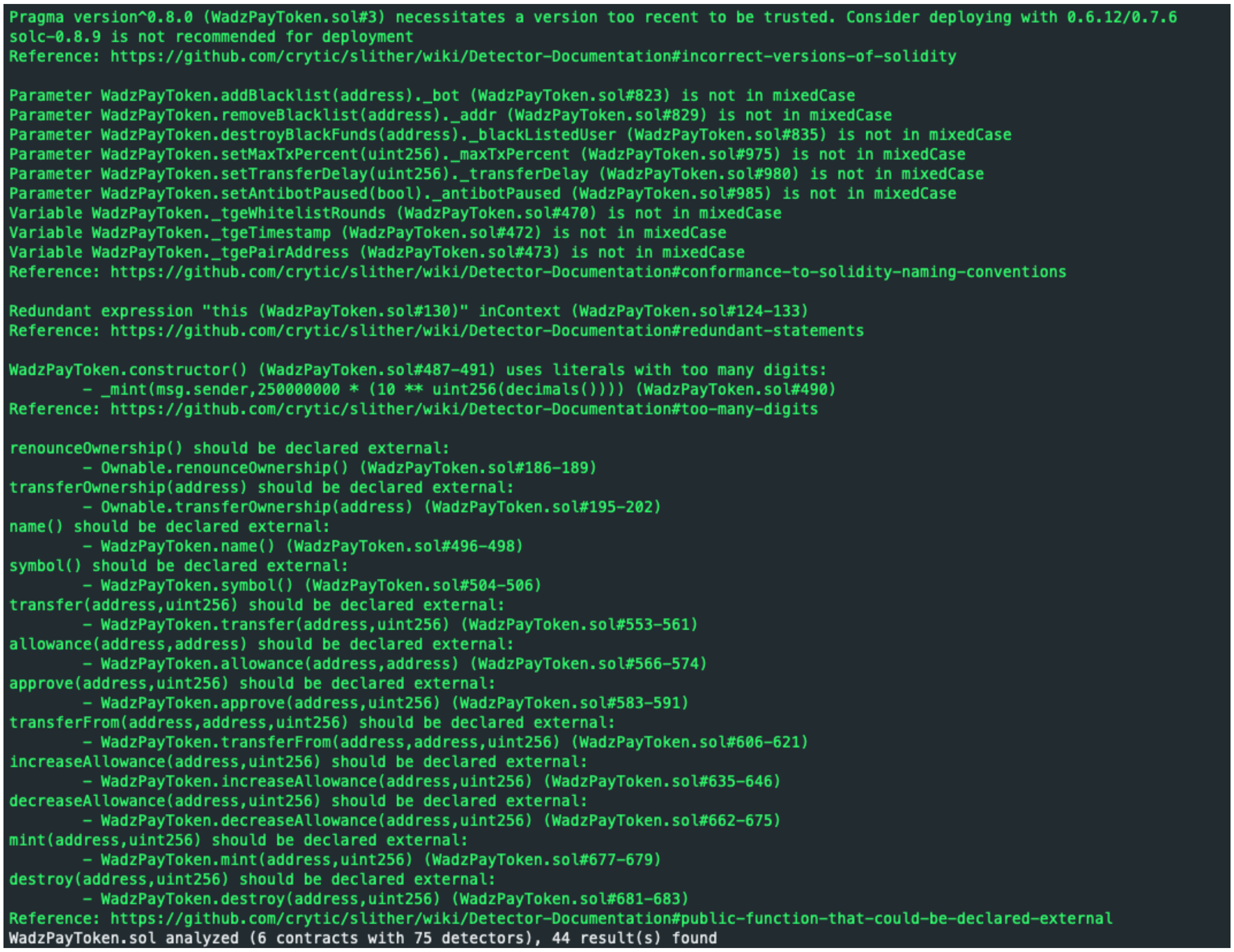
```
922         uint256 _tgeWhitelistRoundLength = _tgeWhitelistRounds.length
923         for (uint256 i = 0; i < _tgeWhitelistRoundLength; i++) {
924
925             WhitelistRound storage wlRound = _tgeWhitelistRounds[i];
926
927             wlCloseTimestampLast = wlCloseTimestampLast.add(wlRound.duration);
928             if(block.timestamp <= wlCloseTimestampLast)
929                 return (i.add(1), wlRound.duration, wlCloseTimestampLast, wlRound.amountMax, wlRo
930         }
```

  Status: Closed

Slither Report



```
WadzPayToken._applyTGEWhitelist(address,address,uint256) (WadzPayToken.sol#940-971) uses a dangerous strict equality:
        - _tgeTimestamp == 0 && sender != _tgePairAddress && recipient == _tgePairAddress && amount > 0 (WadzPayToken.sol#945)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities

WadzPayToken.allowance(address,address).owner (WadzPayToken.sol#566) shadows:
        - Ownable.owner() (WadzPayToken.sol#167-169) (function)
WadzPayToken._approve(address,address,uint256).owner (WadzPayToken.sol#788) shadows:
        - Ownable.owner() (WadzPayToken.sol#167-169) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

WadzPayToken.setMaxTxPercent(uint256) (WadzPayToken.sol#975-977) should emit an event for:
        - maxTxPercent = _maxTxPercent (WadzPayToken.sol#976)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic

WadzPayToken.createTGEWhitelist(address,uint256[],uint256[]).pairAddress (WadzPayToken.sol#859) lacks a zero-check on :
                - _tgePairAddress = pairAddress (WadzPayToken.sol#862)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

WadzPayToken.getTGEWhitelistRound() (WadzPayToken.sol#915-933) uses timestamp for comparisons
        Dangerous comparisons:
        - _tgeTimestamp > 0 (WadzPayToken.sol#917)
        - block.timestamp <= wlCloseTimestampLast (WadzPayToken.sol#926)
WadzPayToken._applyTGEWhitelist(address,address,uint256) (WadzPayToken.sol#940-971) uses timestamp for comparisons
        Dangerous comparisons:
        - _tgeTimestamp == 0 && sender != _tgePairAddress && recipient == _tgePairAddress && amount > 0 (WadzPayToken.sol#945)
        - wlRoundNumber > 0 (WadzPayToken.sol#953)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

Context._msgData() (WadzPayToken.sol#129-132) is never used and should be removed
SafeMath.div(uint256,uint256) (WadzPayToken.sol#330-332) is never used and should be removed
SafeMath.div(uint256,uint256,string) (WadzPayToken.sol#386-395) is never used and should be removed
SafeMath.mod(uint256,uint256) (WadzPayToken.sol#346-348) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (WadzPayToken.sol#412-421) is never used and should be removed
SafeMath.mul(uint256,uint256) (WadzPayToken.sol#316-318) is never used and should be removed
SafeMath.sub(uint256,uint256,string) (WadzPayToken.sol#363-372) is never used and should be removed
SafeMath.tryAdd(uint256,uint256) (WadzPayToken.sol#217-223) is never used and should be removed
SafeMath.tryDiv(uint256,uint256) (WadzPayToken.sol#259-264) is never used and should be removed
SafeMath.tryMod(uint256,uint256) (WadzPayToken.sol#271-276) is never used and should be removed
SafeMath.tryMul(uint256,uint256) (WadzPayToken.sol#242-252) is never used and should be removed
SafeMath.trySub(uint256,uint256) (WadzPayToken.sol#230-235) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.8.0 (WadzPayToken.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.9 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```



```
Pragma version^0.8.0 (WadzPayToken.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.9 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Parameter WadzPayToken.addBlacklist(address)._bot (WadzPayToken.sol#823) is not in mixedCase
Parameter WadzPayToken.removeBlacklist(address)._addr (WadzPayToken.sol#829) is not in mixedCase
Parameter WadzPayToken.destroyBlackFunds(address)._blackListedUser (WadzPayToken.sol#835) is not in mixedCase
Parameter WadzPayToken.setMaxTxPercent(uint256)._maxTxPercent (WadzPayToken.sol#975) is not in mixedCase
Parameter WadzPayToken.setTransferDelay(uint256)._transferDelay (WadzPayToken.sol#980) is not in mixedCase
Parameter WadzPayToken.setAntibotPaused(bool)._antibotPaused (WadzPayToken.sol#985) is not in mixedCase
Variable WadzPayToken._tgeWhitelistRounds (WadzPayToken.sol#470) is not in mixedCase
Variable WadzPayToken._tgeTimestamp (WadzPayToken.sol#472) is not in mixedCase
Variable WadzPayToken._tgePairAddress (WadzPayToken.sol#473) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Redundant expression "this (WadzPayToken.sol#130)" inContext (WadzPayToken.sol#124-133)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements

WadzPayToken.constructor() (WadzPayToken.sol#487-491) uses literals with too many digits:
        - _mint(msg.sender,250000000 * (10 ** uint256(decimals()))) (WadzPayToken.sol#490)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits

renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (WadzPayToken.sol#186-189)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (WadzPayToken.sol#195-202)
name() should be declared external:
        - WadzPayToken.name() (WadzPayToken.sol#496-498)
symbol() should be declared external:
        - WadzPayToken.symbol() (WadzPayToken.sol#504-506)
transfer(address,uint256) should be declared external:
        - WadzPayToken.transfer(address,uint256) (WadzPayToken.sol#553-561)
allowance(address,address) should be declared external:
        - WadzPayToken.allowance(address,address) (WadzPayToken.sol#566-574)
approve(address,uint256) should be declared external:
        - WadzPayToken.approve(address,uint256) (WadzPayToken.sol#583-591)
transferFrom(address,address,uint256) should be declared external:
        - WadzPayToken.transferFrom(address,address,uint256) (WadzPayToken.sol#606-621)
increaseAllowance(address,uint256) should be declared external:
        - WadzPayToken.increaseAllowance(address,uint256) (WadzPayToken.sol#635-646)
decreaseAllowance(address,uint256) should be declared external:
        - WadzPayToken.decreaseAllowance(address,uint256) (WadzPayToken.sol#662-675)
mint(address,uint256) should be declared external:
        - WadzPayToken.mint(address,uint256) (WadzPayToken.sol#677-679)
destroy(address,uint256) should be declared external:
        - WadzPayToken.destroy(address,uint256) (WadzPayToken.sol#681-683)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
WadzPayToken.sol analyzed (6 contracts with 75 detectors), 44 result(s) found
```

All issues raised by slither are covered in the manual audit or are not relevant.

Kovan Test Contracts

WadzPayToken.sol: 0x6D83E4620D2C86D3B8969CC3E66C0CE4dF40DAB5

Test Transactions

createTGEWhitelist() - FAIL
0x3a40a3480a0c6e4d0dcde44153c6f99d2aa460155f28b527714eebdb5aed6d37

_tgeWhitelistRounds(0) - FAIL
VM execution error

approve (1000) to owner address and transfer 200 to other address - PASS

approve() 0x8f176e4082d4b6a657b2937eb51f8f39fb6f94076172870b262ea5fc46ef0918
transferFrom()
0x2e31978684e66febcedc6c31e4dd922ba963f648f0377a497b1b1f3909c21836

addBlackList() - PASS
blacKList(address) - true
0xf5dec428ac68afe64e717e35fe4c0d3d40b62104c7dae4c4126af5f25d178080

removeBlacklist() - PASS
blacKList(address) - false
0x2ddeaa0de5daf12ecc149b1d2c1a45ff99c67041406f161c3cf3707aeef4c9a0

Closing Summary

Overall, smart contracts are well written and adhere to guidelines.
A High, Medium and low severity have been reported and fixed.
We recommend that other than fixing the issues we should also focus on improving the indentation of the code and the naming conventions.

Apart from the Audit we suggest the **WadzPayToken** team to write the Unit Test cases and maintain the coverage for around 100%.

Disclaimer

ImmuneBytes audit is not a security warranty, investment advice, or an endorsement of the **WadzPayToken**. This audit does not provide a security or correctness guarantee of the audited smart contracts. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.