WadzPay

WadzPayToken.sol

Smart Contract Audit
Final Report

December 17, 2021

Introduction

1. About WadzPay

WadzPay is building an interoperable and agnostic blockchain-based payments ecosystem. The company was founded in 2018 in Singapore and is currently operating in South East Asia, South Asia, the Middle East, and Africa.

The potential for CBDC and Digital Assets leading the next revolution in the payments industry: by enabling faster payments, improvements in security, cost efficiency with optionality.

WadzPay is working with large international payment companies, banks, and other large global companies to enable digital asset-based transaction processing and settlement.

Visit https://wadzpay.com/ to know more about it.

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 105+ 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 WadzPay team has provided the following doc for the purpose of audit:

1. Short write-up over the email.

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: WadzPay
- Contracts Name: WadzPayToken.sol
- Languages: Solidity(Smart contract)
- Github commit for initial audit: 9e3ee82e900c7e6a6f6fa32396ab848dd9f5365a
- Github commit for final audit: 299cfbddf3f918cf41a277623f280c55d562be66
- 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:

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	-	-	1
Closed	1	2	2

High Severity Issues

1. **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 pushing the empty object first then using the space.

   Recommendation:

```
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

Amended (December 17th, 2021): The issue was fixed by the **WadzPay** team and is no longer present in commit 299cfbddf3f918cf41a277623f280c55d562be66

Medium Severity Issues

1. **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.

   Amended (December 17th, 2021): The issue was fixed by the **WadzPay** team and is no longer present in commit 299cfbddf3f918cf41a277623f280c55d562be66

2. **The 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 implementing the function body or just removing the function if there is no use of it.

   Amended (December 17th, 2021): The issue was fixed by the **WadzPay** team and is no longer present in commit 299cfbddf3f918cf41a277623f280c55d562be66

Low Severity Issues

1. **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

2. **Unnecessary use of safeMath**

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

   Amended (December 17th, 2021): The issue was fixed by the **WadzPay** team and is no longer present in commit 299cfbddf3f918cf41a277623f280c55d562be66

3. **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

   Amended (December 17th, 2021): The issue was fixed by the **WadzPay** team and is no longer present in commit 299cfbddf3f918cf41a277623f280c55d562be66

Recommendations / Informational

1. **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

   Amended (December 17th, 2021): The issue was fixed by the **WadzPay** team and is no longer present in commit [299cfbddf3f918cf41a277623f280c55d562be66]

2. **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 using the local variable and storing the _tgeWhitelistRounds.length such that read operation gets minimized.

```
uint256 _tgeWhitelistRoundLength = _tgeWhitelistRounds.length
for (uint256 i = 0; i < _tgeWhitelistRoundLength; i++) {

    WhitelistRound storage wlRound = _tgeWhitelistRounds[i];

    wlCloseTimestampLast = wlCloseTimestampLast.add(wlRound.duration);
    if(block.timestamp <= wlCloseTimestampLast)
        return (i.add(1), wlRound.duration, wlCloseTimestampLast, wlRound.amountMax, wlRound.addresses[_msgSender()], wlRound.purchased[_msgSender()]);
}
```

Automated Audit Result

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

Concluding Remarks

While conducting the audits of the WadzPay smart contract, it was observed that the contracts contain High, Medium, and Low severity issues.

Our auditors suggest that High, Medium, and Low severity issues should be resolved by the developers. 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 WadzPay platform or its product nor 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