HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: WeSendit
Date: October 20th, 2022

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Document

Name	Smart Contract Code Review and Security Analysis Report for WeSendit
Approved By	Evgeniy Bezuglyi | SC Audits Department Head at Hacken OU
Type	ERC20 system
Platform	EVM
Network	BSC
Language	Solidity
Methods	Manual Review, Automated Review, Architecture Review
Website	https://wesendit.io/
Timeline	03.10.2022 - 20.10.2022
Changelog	11.10.2022 - Initial Review 20.10.2022 - Second Review

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Table of contents

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Introduction

Hacken OÜ (Consultant) was contracted by WeSendit (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contracts.

Scope

The scope of the project is smart contracts in the repository:

Initial review scope
Repository:
https://github.com/wesenditmedia/contracts
Commit:
bb76815454d2860b0c7bb27a78651ae88abdccff
Documentation:
https://github.com/wesenditmedia/contracts/blob/main/README.md
https://docs.google.com/document/d/1iea6jRToGQCGSJsDU5pJDfHpHHvUHYdldZ2uoWhShdI

Integration and Unit Tests: Yes
Deployed Contracts Addresses:
Contracts:

File: ./contracts/interfaces/IDynamicFeeManager.sol
SHA3: c6322926db0cbb54387e8ee453e2b5e1754d0ad0e2c27db83995df6532e61eae

File: ./contracts/interfaces/IEmergencyGuard.sol
SHA3: 2c40babba26a1eedcf0ce5ea22514771a271972755e9715856a8b449175ae9ac

File: ./contracts/interfaces/IFeeReceiver.sol
SHA3: c843509a238a740c404138c1d99353f93167f6a5cb7bf301395a1b2a909e93f9

File: ./contracts/interfaces/IPancakeRouter.sol
SHA3: a07b8230731d408c679a08296d4a51ed1a40037b8ebb1b5c39dd1116e0b44e08

File: ./contracts/interfaces/IWeSenditToken.sol
SHA3: bf260cdfe9769b13629fd170018aec2a74c85999f317d4aeca6f197202f63729

File: ./contracts/BaseDynamicFeeManager.sol
SHA3: 1869593a3237ccce23e3c9f3a99a1a985d1f653fcff45a3ebe01c430fdb13dcb

File: ./contracts/BaseWeSenditToken.sol
SHA3: 0d45c4d14bed59cb1d4724b8dac1bdfeca453ee71d68a6b564a129c5d59dfb17

File: ./contracts/DynamicFeeManager.sol
SHA3: 74e7dfcc158f62c8d9bc636737d6e8a01ddcb7c3df7b819c358aa913610a8398

File: ./contracts/EmergencyGuard.sol
SHA3: 3fbcf99ff58e02bbdda3d74e30f1a3172a5124665bd4095d9d46eca55563e13b

File: ./contracts/WeSenditToken.sol
SHA3: 6813e196c74a50790b750359fdd0daf2d177f922b4a4cbf032897900570e7412

Second review scope

Repository:
https://github.com/wesenditmedia/contracts

Commit:
e3cbf365f1c13a1c4697aba1ea364c90e3e6c2b4

Documentation:
https://github.com/wesenditmedia/contracts/blob/main/README.md
https://docs.google.com/document/d/1iea6jRToGQCGSJsDU5pJDfHpHHvUHYdldZ2uoWhShdI

Integration and Unit Tests: Yes

Deployed Contracts Addresses:

Contracts:

File: ./contracts/interfaces/IDynamicFeeManager.sol
SHA3: 187653c39e16154afacce2c9601cd8d0125f70c7cf48e8973019023e323f1f75

File: ./contracts/interfaces/IEmergencyGuard.sol
SHA3: dd03e3354b8c124cf76246d1b0fa2e839df9dedc5b4f3959a69d25aa6415348d

File: ./contracts/interfaces/IFeeReceiver.sol
SHA3: fdab70d081f80b4baf5b75e64c50578537077acf65c751d3372413b628232f0b

File: ./contracts/interfaces/IPancakeRouter.sol
SHA3: 16e1e3679313cf4e9c4fd9fe495105e1839114eb9e28ec993aa10fbf468ae5ed

File: ./contracts/interfaces/IWeSenditToken.sol
SHA3: 12949f9d4acd9b36ce3c6504febe6ca8d2e28c691f195f2d8e299b030603f2ed

File: ./contracts/BaseDynamicFeeManager.sol
SHA3: 70dc171a2dced116bda3fe4a26cc7756b2ddb2d4bb251aaae024260f72e5059e

File: ./contracts/BaseWeSenditToken.sol
SHA3: 6ca45ffd804945cd2e8d5be362c7ced74a6568a92c3588841527351a482d2f18

File: ./contracts/DynamicFeeManager.sol
SHA3: 023f7343a81b2d85a42e925174964528801067ffe73e13cad2e665f28c22068f

File: ./contracts/EmergencyGuard.sol
SHA3: 8ac7a40c1a0ac3f209ef9e8d702ea15f2e570699b0c115eb8d4529e397459ab9

File: ./contracts/WeSenditToken.sol
SHA3: 6d3855fde3c22ea16de4850eb0279f9bf3de3758aa51340d4f14fb3977b41f1a

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to assets loss or data manipulations.
High	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g., public access to crucial functions.
Medium	Medium-level vulnerabilities are important to fix; however, they cannot lead to assets loss or data manipulations.
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that cannot have a significant impact on execution.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Executive Summary

The score measurement details can be found in the corresponding section of the scoring methodology.

Documentation quality

The total Documentation Quality score is **10** out of **10**.

- Functional requirements are provided.
- Technical description is provided.

Code quality

The total Code Quality score is **10** out of **10**.

- The development environment is configured.
- Contracts follow official Solidity guidelines.

Test coverage

Test coverage of the project is **100.00%**.

- Deployment and basic user interactions are covered with tests.
- Negative cases are covered.
- Interactions by several users are tested.

Security score

As a result of the audit, the code contains **0** issues. The security score is 10 out of **10**.

All found issues are displayed in the “Findings” section.

Summary

According to the assessment, the Customer's smart contract has the following score: **10.0**.

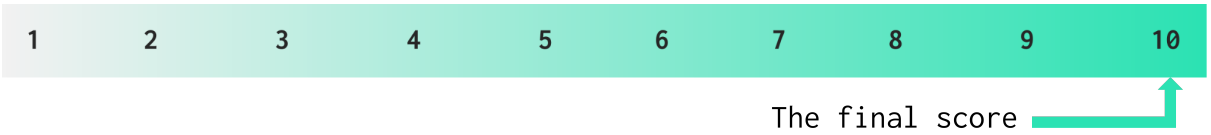

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
6 October 2022	8	4	4	2
20 October 2022	0	0	0	0

Checked Items

We have audited the Customers' smart contracts for commonly known and more specific vulnerabilities. Here are some items considered:

Item	Type	Description	Status
Default Visibility	SWC-100 SWC-108	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed
Integer Overflow and Underflow	SWC-101	If unchecked math is used, all math operations should be safe from overflows and underflows.	Passed
Outdated Compiler Version	SWC-102	It is recommended to use a recent version of the Solidity compiler.	Passed
Floating Pragma	SWC-103	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Passed
Unchecked Call Return Value	SWC-104	The return value of a message call should be checked.	Passed
Access Control & Authorization	CWE-284	Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users.	Passed
SELFDESTRUCT Instruction	SWC-106	The contract should not be self-destructible while it has funds belonging to users.	Not Relevant
Check-Effect-Interaction	SWC-107	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed
Assert Violation	SWC-110	Properly functioning code should never reach a failing assert statement.	Passed
Deprecated Solidity Functions	SWC-111	Deprecated built-in functions should never be used.	Passed
Delegatecall to Untrusted Callee	SWC-112	Delegatecalls should only be allowed to trusted addresses.	Not Relevant
DoS (Denial of Service)	SWC-113 SWC-128	Execution of the code should never be blocked by a specific contract state unless required.	Passed
Race Conditions	SWC-114	Race Conditions and Transactions Order Dependency should not be possible.	Passed

Authorization through tx.origin	SWC-115	tx.origin should not be used for authorization.	Not Relevant
Block values as a proxy for time	SWC-116	Block numbers should not be used for time calculations.	Not Relevant
Signature Unique Id	SWC-117 SWC-121 SWC-122 EIP-155	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifiers should always be used. All parameters from the signature should be used in signer recovery	Not Relevant
Shadowing State Variable	SWC-119	State variables should not be shadowed.	Passed
Weak Sources of Randomness	SWC-120	Random values should never be generated from Chain Attributes or be predictable.	Not Relevant
Incorrect Inheritance Order	SWC-125	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order.	Passed
Calls Only to Trusted Addresses	EEA-Level-2 SWC-126	All external calls should be performed only to trusted addresses.	Passed
Presence of unused variables	SWC-131	The code should not contain unused variables if this is not justified by design.	Passed
EIP standards violation	EIP	EIP standards should not be violated.	Passed
Assets integrity	**Custom**	Funds are protected and cannot be withdrawn without proper permissions.	Passed
User Balances manipulation	**Custom**	Contract owners or any other third party should not be able to access funds belonging to users.	Passed
Data Consistency	**Custom**	Smart contract data should be consistent all over the data flow.	Passed
Flashloan Attack	**Custom**	When working with exchange rates, they should be received from a trusted source and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used.	Passed
Token Supply manipulation	**Custom**	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the customer.	Passed

Gas Limit and Loops	**Custom**	Transaction execution costs should not depend dramatically on the amount of data stored on the contract. There should not be any cases when execution fails due to the block Gas limit.	Passed
Style guide violation	**Custom**	Style guides and best practices should be followed.	Passed
Requirements Compliance	**Custom**	The code should be compliant with the requirements provided by the Customer.	Passed
Environment Consistency	**Custom**	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed
Secure Oracles Usage	**Custom**	The code should have the ability to pause specific data feeds that it relies on. This should be done to protect a contract from compromised oracles.	Not Relevant
Tests Coverage	**Custom**	The code should be covered with unit tests. Test coverage should be 100%, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Passed
Stable Imports	**Custom**	The code should not reference draft contracts, that may be changed in the future.	Passed

System Overview

The WeSendit token is an essential part of the WeSendit ecosystem. The fee system introduced with token helps reduce volatility and ensure liquidity for future project growth.
The files in the scope:

- **BaseDynamicFeeManager.sol** - base contract for DynamicFeeManager.sol. Contains admin functionality.
- **DynamicFeeManager.sol** - main contract with adding fee functionality and fees calculation.
- **BaseWeSenditToken.sol** - base contract for WeSenditToken.sol. Contains admin functionality.
- **WeSenditToken.sol** - ERC20 token with fee reflection functionality implemented with DynamicFeeManager.sol.
- **EmergencyGuard.sol** - Allows contract admin to withdraw funds from WeSenditToken.sol and DynamicFeeManager.sol

Privileged roles

- DynamicFeeManager roles
  - ADMIN: allowed to do admin operations like add/remove fees, adding to fee whitelist, withdraw tokens and chain native currency from contract, grant roles, enable/disable fee for transaction
  - FEE_WHITELIST: allowed to bypass fees
  - RECEIVER_FEE_WHITELIST: allowed to receive tokens without fee
  - BYPASS_SWAP_AND_LIQUIFY: allowed to bypass swap and liquify
  - EXCLUDE_WILDCARD_FEE: allowed to bypass wildcard fees
  - owner: allowed to bypass fees
- WeSenditToken roles
  - ADMIN: allowed to grant roles, unpause contract, change DynamicFeeManager contract, withdraw tokens and chain native currency from contract
  - BYPASS_PAUSE: allowed to bypass pause
  - owner: allowed to bypass fees

Risks

- Admin of the WeSenditToken can upgrade DynamicFeeManager at any time. It can lead to contract transfers stop, increase of fee.
- In case when Pancakeswap pair address is zero or percentage is settled to 0, swap will be performed according to the settled value of fee entry and will not take into consideration pair reserves.
- Admin of the DynamicFeeManager can perform an emergency withdrawal of funds. In this case, fee entries are not updated. Fee entries should be replaced with new ones for correct work of the contract.

Findings

■■■■ Critical

1. Access Control Violation

Access control is missing for critical functionality. Anyone can change *_pancakePairBusdAddress* and *_pancakePairBnbAddress*.

This can lead to the manipulation of amounts used for swap or liquidity adding.

Path: ./contracts/BaseDynamicFeeManager.sol

Functions: setPancakePairBusdAddress, setPancakePairBnbAddress

Recommendation: restrict access for the mentioned functions.

Status: Fixed(e3cbf365f1c13a1c4697aba1ea364c90e3e6c2b4)

2. Ambiguous Third Party Integration; Undocumented Behaviour

According to NatSpec and provided documentation function *_swapAndLiquify* should transfer received LP tokens to the *destination* parameter address. It transfers received BNB from the function *swapExactTokensForETHSupportingFeeOnTransferTokens* call to the *destination* address.

In this case, liquidity will not be added for the BNB/WSI pair.

According to the comments, NatSpec *SwapTokenForBusd* event should emit the amount of received BUSD tokens on the destination address; instead, it shows a change of $WSI token balance of the destination.

Path: ./contracts/BaseDynamicFeeManager.sol

Function: _swapTokensForBnb, _swapTokensForBusd

Recommendation: change input parameter of function *_swapTokensForBnb* from *destination* to *address(this)* and make proper changes in function *_swapTokensForBusd.*

Status: Fixed(e3cbf365f1c13a1c4697aba1ea364c90e3e6c2b4)

High

1. Requirements Violation

It is possible that the sum of all fee entries will exceed *feePercentageLimit*. In this case, all user transfers will be blocked till ADMIN of the contract removes some fee entries to reach the limit.

Path: ./contracts/DynamicFeeManager.sol

Function: addFee

Recommendation: on adding the new fee entry, add the check that the total percentage of all valid fee entries is less or equal to the fee percentage limit.

Status: Fixed(e3cbf365f1c13a1c4697aba1ea364c90e3e6c2b4)

1. Race Condition

Function *reflectFees* is external and can be called by anyone with any parameters. In case the user has an approved amount of tokens for the contract DynamicFeeManager.sol an attacker can call this method with the user as *from* parameter and allowed *amount*.

This can lead to losses of user funds by draining them to cover a fee.

Path: ./contracts/DynamicFeeManager.sol

Function: reflectFees

Recommendation: add a sanitation check for the caller or describe in documentation why such behavior is needed.

Status: Fixed(e3cbf365f1c13a1c4697aba1ea364c90e3e6c2b4)

2. Invalid Calculations

In function *_reflectFee* amounts cannot be updated properly. It is possible due function *_reflectFee* updates amounts for fee entry according to *swapOrLiquifyAmount* and ignores possibility that this value can be changed in functions *_getSwapOrLiquifyAmount*, *_swapTokensForBusd* or *_swapAndLiquify*.

It can lead to not properly updating the fee entry amount.

Path: ./contracts/DynamicFeeManager.sol

Function: _reflectFee

Recommendation: add check of $WSI balance changes to functions *_swapTokensForBusd* and *_swapAndLiquify* and use them for fee entry amount update.

Status: Fixed(e3cbf365f1c13a1c4697aba1ea364c90e3e6c2b4)

3. Denial of Service Vulnerability

Function *reflectFees* loops over all fee entries and calculates plus reflect fee. In case of too many fee entries, the Gas usage of a transaction can exceed the block Gas limit.

It can lead to transaction failure and, as a result, block token transfers.

Path: ./contracts/DynamicFeeManager.sol

Function: reflectFees

Recommendation: add max limit for fee entries.

Status: Fixed(e3cbf365f1c13a1c4697aba1ea364c90e3e6c2b4)

Medium

1. Using SafeMath in Solidity ^0.8.0

Integer overflow check is built-in in Solidity ^0.8.0. Due to this, using this library is redundant.

Paths: ./contracts/BaseDynamicFeeManager.sol, ./contracts/WeSenditToken.sol

Recommendation: remove redundant functionality.

Status: Fixed(e3cbf365f1c13a1c4697aba1ea364c90e3e6c2b4)

2. Tautology of Contradiction

Require statements with checks for *uint256 >= 0* are redundant.

Path: ./contracts/BaseDynamicFeeManager.sol

Functions: setPercentageVolumeSwap, setPercentageVolumeLiquify

Recommendation: remove unnecessary checks.

Status: Fixed(e3cbf365f1c13a1c4697aba1ea364c90e3e6c2b4)

3. Missing Events Emit on Changing Important Values

The contract does not emit any events after changing important values. It is recommended to emit events after changing important values. This will make it easy for everyone to notice such changes.

Path: ./contracts/BaseDynamicFeeManager.sol

Function: setFeesEnabled

Recommendation: implement event emits after changing contract values.

Status: Fixed(e3cbf365f1c13a1c4697aba1ea364c90e3e6c2b4)

4. Unchecked Return Value

The function *addLiquidityETH* from pancakeRouter returns amounts of token used for adding liquidity and LP tokens minted. The call made to the *addLiquidityETH* does not check its return value. This means that the contract will continue its execution even if there is an erroneous situation.

This can lead to improper updating of fee entries amounts.

Paths: ./contracts/DynamicFeeManager.sol.sol, ./contracts/BaseDynamicFeeManager.sol

Functions: _reflectFee, _addLiquidity

Recommendation: implement a check for the $WSI amount used for adding liquidity and update the fee entries amounts according to the results.

Status: Fixed(e3cbf365f1c13a1c4697aba1ea364c90e3e6c2b4)

Low

1. Floating Pragma

The project`s contracts use floating pragma ^0.8.0. Contracts with unlocked pragmas may be deployed by the latest compiler, which may have higher risks of undiscovered bugs. Contracts should be deployed with the same compiler version they have been tested thoroughly.

Paths: ./contracts/ interfaces/IDynamicFeeManager.sol, ./contracts/interfaces/IEmergencyGuard.sol, ./contracts/ interfaces/IFeeReceiver.sol,

./contracts/interfaces/IPancakeRouter.sol, ./contracts/ interfaces/IWeSenditToken.sol,

./contracts/BaseDynamicFeeManager.sol,
./contracts/BaseWeSenditToken.sol,

./contracts/DynamicFeeManager.sol, ./contracts/EmergencyGuard.sol,

./contracts/WeSenditToken.sol

Recommendation: consider locking the pragma version whenever possible.

Status: Fixed(e3cbf365f1c13a1c4697aba1ea364c90e3e6c2b4)

2. Unused Function

The internal function _mint is created and overrides other implementations of this function in inherited contracts but not used in the project.

Path: ./contracts/WeSenditToken.sol

Function: _mint

Recommendation: in WeSenditToken constructor, use the internal function _mint instead of using it from inherited contracts.

Status: Fixed(e3cbf365f1c13a1c4697aba1ea364c90e3e6c2b4)

3. **Style Guide Violation**

The provided projects should follow the official guidelines.

Paths: ./contracts/BaseDynamicFeeManager.sol, ./contracts/BaseWeSenditToken.sol, ./contracts/WeSenditToken.sol

Recommendation: follow the official Solidity guidelines.

Status: Fixed(e3cbf365f1c13a1c4697aba1ea364c90e3e6c2b4)

4. **Functions that Can Be Declared External**

“public” functions that are never called by the contract should be declared “external” to save Gas.

Paths: ./contracts/BaseDynamicFeeManager.sol, ./contracts/BaseWeSenditToken.sol

Functions: getFee, getFeeAmount, feeDecreased, initialSupply

Recommendation: use the external attribute for functions never called from the contract.

Status: Fixed(e3cbf365f1c13a1c4697aba1ea364c90e3e6c2b4)

5. **Redundant Require Statement**

In function addFee require statement (`percentage <= FEE_DIVIDER`) is redundant. The maximum percentage can be 25000 (25%) and it is checked below in the mentioned require statement.

This can lead to higher Gas taxes.

Path: ./contracts/DynamicFeeManager.sol

Function: addFee

Recommendation: remove the redundant require.

Status: Fixed(e3cbf365f1c13a1c4697aba1ea364c90e3e6c2b4)

6. **Redundant Function Call**

Function `reflectFees` have redundant calls to function `_validateFeeAmount`.Function `_validateFeeAmount` consists of only two require statements. The require statement (`tTotal.add(tFees) == amount`) is redundant due to calculation of `tTotal` in main function `reflectFees` and will always bypass this require statement. Redundant calls to functions will create excessive Gas costs.

Path: ./contracts/DynamicFeeManager.sol

Functions: reflectFees , _validateFeeAmount

Recommendation: move require statement (*tTotal > 0*) to function *reflectFees* and remove function *_validateFeeAmount*.

Status: Fixed(e3cbf365f1c13a1c4697aba1ea364c90e3e6c2b4)

7. **Excessive State Variable Access**

   In function reflectFees the for loop is used to calculate the total percentage of fees. On every iteration, it checks that *i < _fees.length*.

   It will lead to higher Gas costs.

   Path: ./contracts/DynamicFeeManager.sol

   Function: reflectFees

   Recommendation: create a local variable, declare it equal to _fees.length and use it for loop iteration.

   Status: Fixed(e3cbf365f1c13a1c4697aba1ea364c90e3e6c2b4)

8. **Missing Zero Address Validation**

   Address parameters are being used without checking against the possibility of 0x0.

   This can lead to unwanted external calls to 0x0.

   Paths: ./contracts/BaseDynamicFeeManager.sol, ./contracts/BaseWeSenditToken.sol

   Functions: setPancakeRouter, setBusdAddress, setPancakePairBusdAddress, setPancakePairBnbAddress, setDynamicFeeManager

   Recommendation: Implement zero address checks.

   Status: Fixed(e3cbf365f1c13a1c4697aba1ea364c90e3e6c2b4)

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed by the best industry practices at the date of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted to and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, Consultant cannot guarantee the explicit security of the audited smart contracts.