Blaize.Security

September 22nd, 2023 / V. 1.0

COINSENDER
SMART CONTRACT AUDIT

TABLE OF CONTENTS

AUDIT RATING

SCORE **9.9** /10

The scope of the project includes **CosmWasm** contracts

src/contract.rs
src/error.rs
src/lib.rs
src/msg.rs

Repository:
https://github.com/Megadev-OU/cosmwasm-contracts

Branch: main

Initial commit:

- 800f348f942289bfa52ef85fd23d07b5ee7dffa7

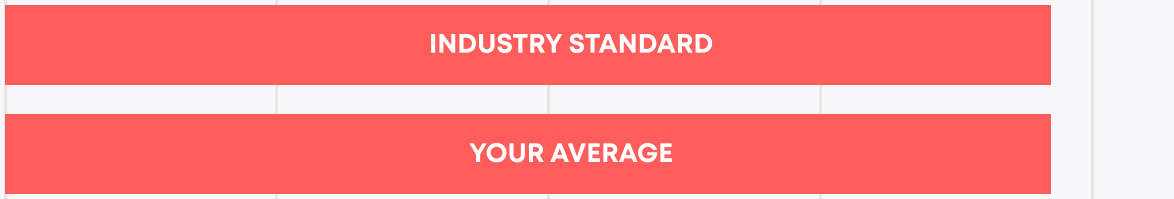
Final commit:

- 6cd9ba07f7b0272b355722e24020068da8be4f8c

TECHNICAL SUMMARY

During the audit, we examined the security of smart contracts for the CoinSender protocol. Our task was to find and describe any security issues in the smart contracts of the platform. This report presents the findings of the security audit of the **CoinSender** smart contracts conducted between **September 6th, 2023** and **September 22nd, 2023.**

Testable code



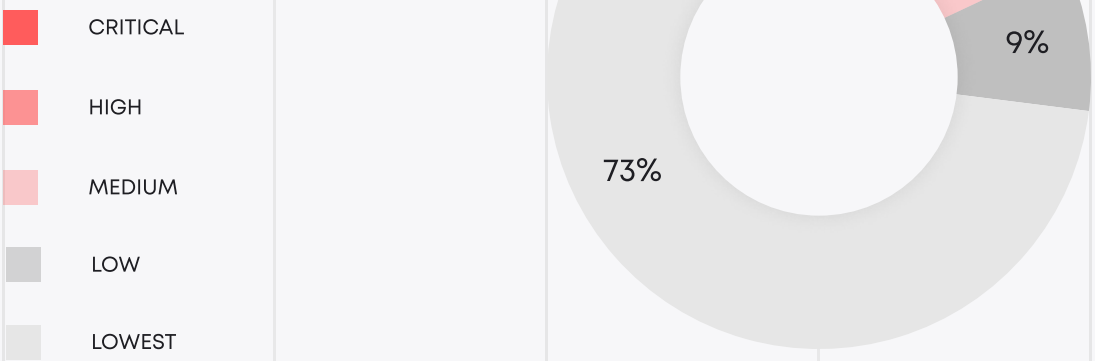
Auditors approved code as testable within the industry standard.

The audit scope includes all tests and scripts, documentation, and requirements presented by the **CoinSender** team. The coverage is calculated based on the of CosmWasm testing framework and scripts from additional testing strategies, and includes testable code from manual and exploratory rounds.

However, to ensure the security of the contract, the **Blaize.Security** team suggests that the **CoinSender** team follow post-audit steps:

1. launch **active protection** over the deployed contracts to have a system of early detection and alerts for malicious activity. We recommend the AI-powered threat prevention platform **VigiLens**, by the **CyVers** team.
2. launch a **bug bounty program** to encourage further active analysis of the smart contracts.

THE GRAPH OF VULNERABILITIES DISTRIBUTION:



The table below shows the number of the detected issues and their severity. A total of 11 problems were found. 11 issues were fixed or verified by the CoinSender team.

	FOUND	FIXED/VERIFIED
Critical	0	0
High	0	0
Medium	2	2
Low	1	1
Lowest	8	8

SEVERITY DEFINITION

Critical

The system contains several issues ranked as very serious and dangerous for users and the secure work of the system. Requires immediate fixes and a further check.

High

The system contains a couple of serious issues, which lead to unreliable work of the system and migh cause a huge data or financial leak. Requires immediate fixes and a further check.

Medium

The system contains issues that may lead to medium financial loss or users' private information leak. Requires immediate fixes and a further check.

Low

The system contains several risks ranked as relatively small with the low impact on the users' information and financial security. Requires fixes.

Lowest

The system does not contain any issues critical to the secure work of the system, yet is relevant for best practices

AUDITING STRATEGY AND TECHNIQUES APPLIED/PROCEDURE

Blaize.Security auditors start the audit by developing an **auditing strategy** - an individual plan where the team plans methods, techniques, approaches for the audited components. That includes a list of activities:

Manual audit stage

- Manual line-by-line code by at least 2 security auditors with crosschecks and validation from the security lead;
- Protocol decomposition and components analysis with building an interaction scheme, depicting internal flows between the components and sequence diagrams;
- Business logic inspection for potential loopholes, deadlocks, backdoors;
- Math operations and calculations analysis, formula modeling;
- Access control review, roles structure, analysis of user and admin capabilities and behavior;
- Review of dependencies, 3rd parties, and integrations;
- Review with automated tools and static analysis;
- Vulnerabilities analysis against several checklists, including internal Blaize.Security checklist;
- Storage usage review;
- Gas (or tx weight or cross-contract calls or another analog) optimization;
- Code quality, documentation, and consistency review.

For advanced components:

- Cryptographical elements and keys storage/usage audit (if applicable);
- Review against OWASP recommendations (if applicable);
- Blockchain interacting components and transactions flow (if applicable);
- Review against CCSSA (C4) checklist and recommendations (if applicable);

Testing stage:

- Development of edge cases based on manual stage results for false positives validation;
- Integration tests for checking connections with 3rd parties;
- Manual exploratory tests over the locally deployed protocol;
- Checking the existing set of tests and performing additional unit testing;
- Fuzzy and mutation tests (by request or necessity);
- End-to-end testing of complex systems;

In case of any issues found during audit activities, the team provides detailed recommendations for all findings.

EXECUTIVE SUMMARY

CoinSender contract is a cosmwasm-based tool that simplifies batch transactions, allowing users to transfer tokens to multiple accounts in one transaction effortlessly. This tool will be useful in many scenarios, including the distribution of tokens in airdrops, employee rewards, and payments to various suppliers.

Auditors provide an in-depth review of the contract, validate its deployment flow, initialization, presence of necessary validations, and several more crucial places. During the testing stage, auditors check the full flow of the contract - including validation of the tx initiation, funds distribution, fees calculation, and sending to the collector account. There were no critical findings, found issues are related to missing validations, missed edge cases processing (found during the testing stage), and code quality connected problems. All issues were resolved or verified by the CoinSender team.

However, the CoinSender team applied a major update to the code during the audit, which completely excluded admin-related logic and changed the fee system. Auditors checked the update and verified the contract's security. Now, the contract is fully decentralized. One of the important features added to this contract is a user-specified transaction fee that can be variated from 0.1% to 5%, which is equally applied to all transactions, regardless of whether they involve native tokens or others. Despite the contract needs an extended fee structure description (or natspec comments), the user flow is quite transparent, and the contract is easy to use.

The code is well-organized and self-declaring, with good native test coverage. Therefore, the contract is verified to be secure for the usage.

	RATING
Security	10
Logic optimization	10
Code quality	9.8
Test coverage**	9.8
Total	9.9

** Contract has quite high native coverage which was checked by Blaize Security team.

PROTOCOL OVERVIEW

Roles & Responsibilitie

1. **Deployer**
    a. **Deployment:** The deployer deploys the smart contract by calling the `instantiate()` function. During deployment, the owner sets the initial value bank's address
    b. **Flow:**
        i. There is no deployment script, contract is deployed manually via wasmd and CLI interface.
2. After the contract update performed by CoinSender team during the audit, the contract became completely decentralized, with no admin/ owner roles. The only varied value is fee, which is provided by the dApp
3. Users:
    a. **Role:** Users are individuals or entities interacting with the deployed smart contract to send tokens to recipients.
    b. **Responsibilities:**
        i. **Transaction Execution:** Users can call the execute() function with the TokenSender action to send tokens to one or more recipients.
        ii. **Query Information:** Users can call the query() function to retrieve information about the owner's address, bank's address, or the current fee percentage.
    c. **Flow:**
        i. Execute transactions by calling execute() with the TokenSender action.
        ii. Query information as needed using the query() function.

Settings

Settings that must be set during deployment:

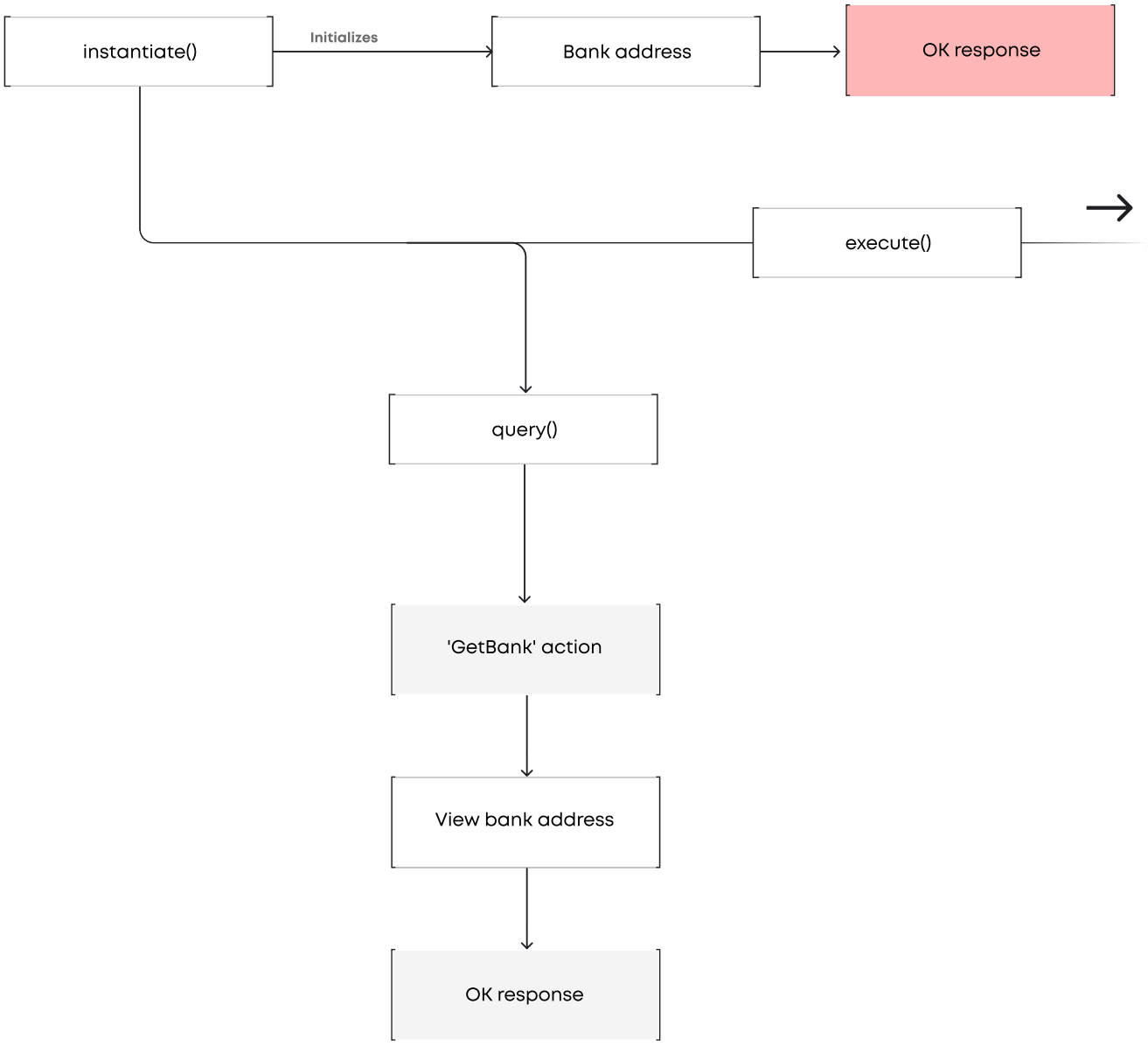
1. **Bank Address:** The bank's address is where transaction fees will be sent. This address should be set during deployment to determine where collected fees are routed. Once set it cannot be changed. So in case of bank address compromise, the whole contract needs to be re-deployed.
2. Max fee: Constant representing the maximum fee which can be deducted. It is set to 5%. Also, it needs to be noticed, that the minimum fee is 0.1%, implemented via the the check against non-zero fee.

Deployment

No deployment script provided

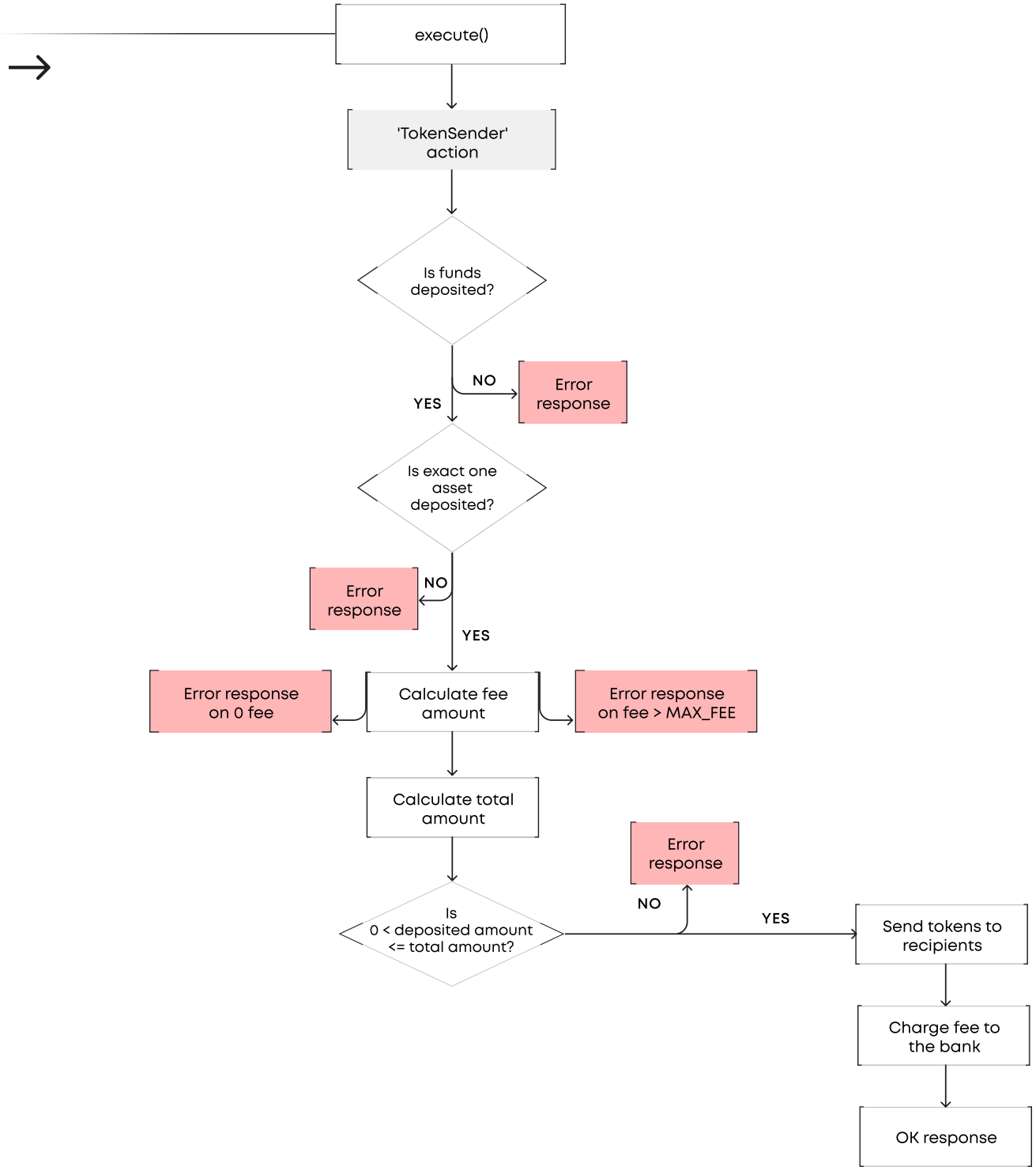
WORKFLOW

Basic Workflow



WORKFLOW

Basic Workflow



COMPLETE ANALYSIS

MEDIUM-1	✔ Resolved

No validation for deposited amount to exceed the total amount
src/contract.rs.
In the 'TokenSender' action, If the deposited amount is greater than the total amount required, the excess tokens would not be returned to the sender, and furthermore, would not be sent to the bank account and would not be sent to the owner of the contract. Funds will be stuck at the contract address.

Recommendation:

Add validation to the 'TokenSender' action, that ensures that the deposited amount is exactly equal to the required total amount **OR** add functionality to return excess tokens to the sender.

MEDIUM-2	✔ Resolved

Missing owner address validation.
src/contract.rs. line 28, line 105.
There is no validation for the provided owner address in the 'instantiate' function and the 'ChangeOwner' action. It may cause contract ownership loss, and consequently, the contract management mechanism will be unreachable.
That means the function may take invalid text strings as input, and ownership will be lost. The issue is marked as **Medium**, as it is under the owner's control and has a minor impact on security. Still, it increases the risk of human mistakes and further impacts the protocol.

Recommendation:

Add the owner address validation(native cosmwasm 'addr_validate') to the **'instantiate'** function and to the **'ChangeOwner'** action, to avoid possible contract ownership loss.

Post-audit:

Admin functionality was completely removed

LOW-1	✔ Resolved

Unclear fee calculation

src/contract.rs. line 66.
In the 'TokenSender' action, a fee amount calculation represents the amount of tokens users should deposit to pay for the execution. Since the fee amount is calculated using the 'get_fee()' function that returns the fee value as is, it is necessary to divide the denominator value by 'PERCENT_PRECISION.' However, several tests seem to have incorrect fee representation (refer to Info-5), and the calculation example has no comment/natspec.
So, such tests and missing comments may be misleading during the fee update and cause an incorrectly high amount set.

Recommendation:

Validate the fee calculation (the 'PERCENT_PRECISION' constant usage) **AND** verify native tests **AND** add the natspec comment with fee example calculation (e.g. 10 = 1%, 500 = 50%, 1000 = 100%)

From client:

With removal of admin, fee percent will be passed as argument of 'TokenSender' action, fee calculation description will be added.

Post-audit:

Fee percentage is passed as an argument of 'TokenSender' action, fee calculation description was added. Fee validation was added with the fee variated from 0.1% to 5%.

LOWEST-1	✔ Resolved

Unnecessary validation.

src/contract.rs. line 44.
In the ‘TokenSender’ action, verifying deposited funds is unnecessary because the next verification is for depositing one asset only. If no asset is deposited, it will not meet the condition of the ‘exact one asset deposited’ validation and will throw an error. This validation makes sense only if adding a custom error message for this scenario is important.

Recommendation:

Remove unnecessary validation **OR** confirm this validation is needed because of a custom error message.

LOWEST-2	✔ Verified

Missing validation for the same recipient address as the sender.

src/contract.rs.
In the ‘TokenSender’ action, there is no validation for the sender trying to send tokens to himself. In general, it’s unnecessary but will make the user experience more pleasant.

Recommendation:

Add validation for the same sender and recipient address **OR** confirm that the current logic is correct.

LOWEST-3 ✔ **Resolved**

Missing fee validation.

src/contract.rs. line 22, line 124.
There is no validation for the fee provided in the ‘instantiate’ function and the ‘ChangeFee’ action. It may cause a scenario where the fee value provided may be from ‘0’ to ‘u128::MAX’ and, respectively, the bank account may lose its purpose, or the user could lose a lot of money just because of inattention because the fee amount may be set to 100% and even much more than 100%

Recommendation:

Add fee validation functionality against the max value. It is recommended to add a threshold constant for the fee, which cannot be exceeded.

From client:

With removal of admin, fee percent will be passed as argument of ‘TokenSender’ action, fee calculation description will be added.

Post-audit:

Fee percentage is passed as an argument of ‘TokenSender’ action, fee calculation description was added. Fee validation was added with the fee variated from 0.1% to 5%.

LOWEST-4 ✔ **Resolved**

Unused variable.

src/contract.rs. line 17.
A constant variable is never used. This variable does not affect anything, but may be needed in the future (even though it's not a good practice).

Recommendation:

Delete an unnecessary variable **OR** confirm the need for that variable in the future

LOWEST-5	✔ Resolved

Issues with native tests.

1) Incorrect assertion in native test
tests/multisend/test_multisend.rs line 103.
The owner executes the 'TokenSender' action, but in the assertion, there is a 'user' balance checked. Furthermore, the fee amount should be removed, not added to the sender's address.

2) Incorrect fee representation
tests/test_utility/test_change_fee.rs line 19, line 35.
tests/utils.rs line 72.
In test_change_fee.rs, there is a comment that describes that the fee provided to the function is 10%, but for such a percentage, according to the 'TokenSender' calculations, the provided value should be just '100'; since then, it will be divided for 'PERCENT_PRECISION' value. Analogically, in utils.rs, to set a 1% fee value, the provided value should be just '10' for the same reasons. Fee calculations should be changed per **Low-1** fixes.

Recommendation:

Fix native tests.

LOWEST-6	✔ Resolved

Missing bank address validation.

src/contract.rs. line 29, line 119
There is no validation for the provided bank address in the 'instantiate' function and the 'ChangeBank' action. This value is checked in the 'TokenSender' action, and tokens can't be transferred to the wrong address, but still, it's essential to disallow any incorrect addresses to be instantiated in the contract.

Recommendation:

Add the bank address validation(native cosmwasm 'addr_validate') to the 'instantiate' function and to the 'ChangeBank' action, to avoid wrong address instantiation.

LOWEST-7	✔ Resolved

Owner can change the fee, bank address and owner address at any time

src/contract.rs.

The owner can change the core contract values without limitations by calling 'ChangeOwner,' 'ChangeFee,' and 'ChangeBank' actions. It may cause a scenario where the user expected one thing (using query methods) but got another (since the owner may change some values before 'TokenSender' was executed by the user).

Such behavior means an overpowered owner and generally requires either rights limitation or implementation of multisig-like access with timelock functionality. However, auditors understand that such logic may be bound to the business case and desired architecture. Either way, such issues should be present in the report and confirmed by the team.

Recommendation:

Confirm that it is correct logic **OR** add limitations to the 'ChangeOwner', 'ChangeFee' and 'ChangeBank' actions.

LOWEST-8	✔ Resolved

File 'cosmwasm_contracts.wasm' is not generated

src/contract.rs.

To optimize compilation by using rust-optimizer and further deployment, it is required that the 'cosmwasm_contracts.wasm' file be generated after the' cargo wasm' command executed, but it isn't.

Recommendation:

In order to generate *.wasm file of the contract, request a crate-type of 'cdylib' in 'cargo.toml' by providing file following lines:

[lib]

crate-type = ["cdylib"].

CODE COVERAGE AND TEST RESULTS FOR ALL FILES, PREPARED BY BLAIZE SECURITY TEAM

multisend::test_multisend::tests::test_repeated_recepients
multisend::test_multisend::tests::test_excess_deposited_funds
multisend::test_multisend::tests::test_instantiate_invalid_bank
multisend::test_multisend::tests::test_instantiate_invalid_owner
multisend::test_multisend::tests::test_properly_provided_fee
multisend::test_multisend::tests::test_invalid_token

Obsolete tests from previous iteration:
multisend::test_multisend::tests::test_invalid_bank_address
multisend::test_multisend::tests::test_instantiate_invalid_owner
test_utility::test_change_bank::tests::test_change_bank_invalid_addr
test_utility::test_change_fee::tests::test_change_fee_to_extreme_max_value
test_utility::test_change_fee::tests::test_change_fee_to_extreme_min_value
test_utility::test_change_owner::tests::test_change_owner_invalid_addr

CODE COVERAGE AND TEST RESULTS FOR ALL FILES, PREPARED BY COINSENDER TEAM

multisend::test_multisend::tests::test_success ... ok
multisend::test_multisend::tests::test_fail_not_enough_deposited - should panic ... ok
test multisend::test_multisend::tests::test_fail_fee_too_big - should panic ... ok

Obsolete tests from previous iteration:
test_utility::test_change_bank::tests::test_change_bank_success ... ok
test_utility::test_change_bank::tests::test_change_bank_fail - should panic ... ok
test_utility::test_change_fee::tests::test_change_fee_success ... ok
test_utility::test_change_fee::tests::test_change_fee_fail - should panic ... ok
test_utility::test_change_owner::tests::test_change_owner_success ... ok
test_utility::test_change_owner::tests::test_change_owner_fail - should panic ... ok

DISCLAIMER

The information presented in this report is an intellectual property of the customer, including all the presented documentation, code databases, labels, titles, ways of usage, as well as the information about potential vulnerabilities and methods of their exploitation. This audit report does not give any warranties on the absolute security of the code. Blaize.Security is not responsible for how you use this product and does not constitute any investment advice.

Blaize.Security does not provide any warranty that the working product will be compatible with any software, system, protocol or service and operate without interruption. We do not claim the investigated product is able to meet your or anyone else's requirements and be fully secure, complete, accurate, and free of any errors and code inconsistency.

We are not responsible for all subsequent changes, deletions, and relocations of the code within the contracts that are the subjects of this report.

You should perceive Blaize.Security as a tool, which helps to investigate and detect the weaknesses and vulnerable parts that may accelerate the technology improvements and faster error elimination.