Audit Report

Injective wasmx and CosmWasm bindings

v1.0

January 20, 2023

Table of Contents

License

Disclaimer

This audit has been performed by

Oak Security

https://oaksecurity.io/
info@oaksecurity.io

Introduction

Purpose of This Report

Oak Security has been engaged by Injective Labs to perform a security audit of the Injective core wasmx module and CosmWasm bindings.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine smart contract bugs, which might lead to unexpected behavior.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebase Submitted for the Audit

The audit has been performed on the following GitHub repositories:

https://github.com/InjectiveLabs/injective-core (referred to as `injective-core` below)

Commit hash: `2d2cdc763904e1c587afe79aeffa05cd6f7268f7`

https://github.com/InjectiveLabs/cw-injective (referred to as `cw-injective` below)

Commit hash: `fdc9fdd64b240462ddbce5011ec6a645ffa01b4f`

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line by line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
4. Report preparation

Functionality Overview

Injective is an open, interoperable smart contract platform built for decentralized financial applications.

The audit scope comprehends the `wasmx` Cosmos SDK module, the `registry` CosmWasm smart contract, CosmWasm bindings and the `MsgExec` implementation in the `exchange` Cosmos SDK module.

How to Read This Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending,** **Acknowledged**, or **Resolved**.

Note that audits are an important step to improving the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than in a security audit and vice versa.

Summary of Findings

No	Description	Severity	Status
1	Contract creators can update the gas price into invalid integer value in order to disable the `wasmx`'s `BeginBlocker` execution	Critical	Resolved
2	Valid positions may be rejected prior the application of PnL	Major	Resolved
3	Governance is not able to effectively remove a contract from `registry`	Major	Resolved
4	`registry` contract and `wasmx` module are not aware of registered contract migrations	Major	Acknowledged
5	Integer gas price type allows negative values	Minor	Resolved
6	Consider validating max contract gas limit is higher than the minimum execution gas limit	Minor	Resolved
7	Too many contracts registered would cause `GetContracts` and `GetActiveContracts` query messages to fail	Minor	Resolved
8	Custom calls between Cosmos SDK modules and CosmWasm contracts requires third party developers to implement their own authorization logic	Minor	Resolved
9	Name validation can be bypassed with whitespace	Informational	Acknowledged
10	Consider deduping batch contract addresses in `ValidateBasic`	Informational	Resolved
11	Outstanding `TODO` comments present in the codebase	Informational	Resolved
12	A `dev-dependency` is vulnerable to two `CVE`s	Informational	Resolved

Code Quality Criteria

Criteria	Status	Comment
Code complexity	Medium	-
Code readability and clarity	Medium-High	-
Level of documentation	Medium	-
Test coverage	Low	17.4% test coverage for wasmx Cosmos SDK module. 1.2% test coverage for registry CosmWasm contract.

Detailed Findings

1. Contract creators can update the gas price into invalid integer value in order to disable the `wasmx`'s `BeginBlocker` execution

Severity: Critical

In `cw-injective:contracts/registry/src/contract.rs:149`, the contract creator can update the gas price to an invalid string.

Since gas prices are supposed to be denominated in integer values, this would cause the `ToContractExecutionParams` message to fail in `injective-core:injective-chain/modules/wasmx/types/exec_msgs.go:127` due to an invalid gas price.

Consequently, the `ExecuteContracts` functionality in `BeginBlocker` will always fail due to the `FetchRegisteredContractExecutionList` function returning an error in `injective-core:injective-chain/modules/wasmx/keeper/wasm.go:50`.

A malicious user can use this behavior to inhibit the `BeginBlocker` execution of the `wasmx` module.

Recommendation

We recommend modifying the `gas_price` type in `cw-injective:contracts/registry/src/contract.rs:136` to force the user input to be a valid unsigned integer instead of a string.

Status: Resolved

2. Valid positions may be rejected prior the application of PnL

Severity: Major

During the handling of synthetic trades in `injective-core:injective-chain/modules/exchange/keeper/wasm_privileged_action.go`, the initial margin requirements of positions are verified twice: It is verified before and after the application of PnL in lines `128` and `139`, respectively.

Consequently, if a position does not fulfill the initial margin requirement before the application of position delta but does fulfill it afterwards, an error is thrown. This implies that valid positions can be rejected.

Recommendation

We recommend verifying that initial margin requirements are fulfilled, using the function `ensurePositionAboveInitialMarginRatio`, post application of any position delta in `injective-core:injective-chain/modules/exchange/keeper/wasm_privileged_action.go:139`.

Status: Resolved

3. Governance is not able to effectively remove a contract from `registry`

Severity: Major

In `injective-core:injective-chain/modules/wasmx/proposal_handler.go:13-24` no proposal is defined that allows removal of a contract from the `registry`.

The `wasmx` module can disable a contract, but the contract's creator could simply re-activate it by interacting with that contract.

This implies that a compromised contract, a contract that has been migrated to a malicious code, or one that has lost its reputation, cannot be removed from the `registry`.

Recommendation

We recommend implementing a new proposal and a message in the `registry` contract in order to let governance remove a contract from the `registry`.

Status: Resolved

4. `registry` contract and `wasmx` module are not aware of registered contract migrations

Severity: Major

The `ContractRegistrationRequestProposal` proposal specifies a contract address that should be added in the `registry`.

The contract address identifies the contract instance, but not its deployed `wasm` code. In fact, a malicious contract creator could migrate a registered contract to a new `wasm` binary in order to steal user funds or to execute malicious logic that can impact the entire chain, for example, a high resource computation in the `begin_blocker` handler.

Recommendation

We recommend using a `(address, code_id)` tuple in order to identify unequivocally the contract instance and its code version.

Status: Acknowledged

5. Integer gas price type allows negative values

Severity: Minor

The `Int` type is used to represent the gas price in the `ContractRegistrationRequest` struct defined in `injective-core:injective-chain/modules/wasmx/types/wasmx.pb.go`.

As an integer type can hold both positive and negative numbers, a negative gas price would trigger an error during the execution of a `wasmx`'s registered contract.

Recommendation

We recommend using an unsigned integer type for gas prices throughout the code base and specifically in the `ContractRegistrationRequest` struct defined in `injective-core:injective-chain/modules/wasmx/types/wasmx.pb.go`.

Status: Resolved

6. Max contract gas limit can be set lower than minimum execution gas limit, causing contract registration to fail

Severity: Minor

In `injective-core:injective-chain/modules/wasmx/types/params.go:111-121`, there is no validation to ensure the `MaxContractGasLimit` value is higher than `keeper.MinExecutionGasLimit`, which represents the minimum gas limit (`90396` in `injective-core:injective-chain/modules/wasmx/keeper/wasm.go:17`).

As a result, a misconfiguration of the maximum contract gas limit to be lower than `90396` would cause the validation in `injective-core:injective-chain/modules/wasmx/proposal_handler.go:30` to always fail, affecting the `handleContractRegistration` functionality.

Recommendation

We recommend verifying the `MaxContractGasLimit` value to be greater than `keeper.MinExecutionGasLimit`.

Status: Resolved

7. Too many contracts registered would cause `GetContracts` and `GetActiveContracts` query messages to fail

Severity: Minor

In `cw-injective:contracts/registry/src/contract.rs:220` and `237`, both `query_contracts` and `query_active_contracts` messages attempt to fetch all contracts registered in the storage and return them to the caller. This is problematic because the query could run out of gas in an execution context if too many contracts are stored.

Recommendation

We recommend implementing a pagination mechanism for `query_contracts` and `query_active_contracts` to prevent out-of-gas issues.

Status: Resolved

8. Custom calls between Cosmos SDK modules and CosmWasm contracts require third-party developers to implement their own authorization logic

Severity: Minor

The `wasmx` and `exchange` modules are performing authorized calls to particular CosmWasm contract handlers using custom logic.

In order to do so, Cosmos SDK modules are performing `Execute` functions from `wasm` module impersonating the receiver contract. From the CosmWasm side, it is as if the message comes from the contract itself.

This means that on the CosmWasm contract, contract developers need to implement custom authorization logic in order to restrict the access of a particular entrypoint only to Injective Cosmos SDK modules. Delegating the authorization of the interaction between modules and smart contracts to third-party developers is risky since they could not implement it in the right way or not implement it at all.

Additionally, the workaround to use the receiver contract address to call modules' reserved handlers has the side effect that the contract can execute calls that should be reserved to `wasmx` and `exchange` modules by itself.

Recommendation

In CosmWasm version 1.0, the concept of `Sudo` messages got introduced, which are designed for this specific use case.

We recommend using such `Sudo` messages to perform calls from Cosmos SDK modules to CosmWasm smart contracts.

Status: Resolved

9. Name validation can be bypassed with whitespace

Severity: Informational

In `injective-core:injective-chain/modules/exchange/types/msgs.go:1288`, the execution will revert if the data name provided is an empty string. This validation can be bypassed by providing whitespace. Ideally, whitespaces should be prohibited since they are also invalid names.

Recommendation

We recommend trimming whitespaces before validating that the name input is not an empty string.

Status: Acknowledged

10. Consider deduping batch contract addresses in `ValidateBasic`

Severity: Informational

In `injective-core:injective-chain/modules/wasmx/types/proposal.go:83`, the `ValidateBasic` function does not filter duplicate contract addresses contained in the `ContractRegistrationRequests` array. Such duplicate contract addresses would cause the execution to fail for each duplicated contract address in `injective-core:injective-chain/modules/wasmx/proposal_handler.go:60` since the contract is already registered.

Recommendation

We recommend deduping contract addresses during the `ValidateBasic` functionality in `injective-core:injective-chain/modules/wasmx/types/proposal.go:83`.

Status: Resolved

11. Outstanding `TODO` comments present in the codebase

Severity: Informational

`TODO` comments were found in the following code lines:

- `injective-core:injective-chain/modules/wasmx/keeper/wasm.go:106`
- `injective-core:injective-chain/modules/wasmx/keeper/wasm.go:240`
- `injective-core:injective-chain/modules/wasmx/keeper/wasm.go:269`
- `cw-injective:packages/injective-cosmwasm/src/subaccount.rs:12`

Recommendation

We recommend resolving the `TODO` comments in the codebase.

Status: Resolved

12. A `dev-dependency` is vulnerable to two `CVEs`

Severity: Informational

The `plotters` crate defined as a `dev-dependency` in `cw-injective:packages/injective-math/Cargo.toml:27` is vulnerable to two `CVEs`:

- RUSTSEC-2020-0159: chrono
- RUSTSEC-2020-0071 - Potential segfault in the time crate

Recommendation

We recommend updating or removing `plotters` from the dependencies.

Status: Resolved