Audit Report

Nolus Core

v1.1

December 12, 2022

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 Nolus Platform AG to perform a security audit of the Nolus Core Cosmos SDK modules.

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 private GitLab repository:

https://github.com/Nolus-Protocol/nolus-core

Commit hash: `04b5064f1164c3a9a2cfb3f07398783f1fe1159a`

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

Nolus Protocol is a Web3 financial suite that offers an innovative approach to money markets with a novel lease solution to further develop the DeFi space.

The audit scope includes the `mint` and `tax` Cosmos SDK modules.

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	Unbounded iteration in the `tax` module's `AnteHandler` could be used by an attacker to slow down or halt the chain	Major	Resolved
2	`ValidateMinter` does not guarantee that provided parameters are adherent to the defined minting schedule	Minor	Resolved
3	Silently handled broken invariant could lead to inconsistent chain state	Minor	Resolved
4	The `feeCaps` parameter is not validated	Minor	Resolved
5	Minting cap is not enforced during genesis	Minor	Resolved
6	The `FixedMintedAmount` parameter value is incoherent with the documentation	Minor	Resolved
7	Inefficient calculation of `remainingFees`	Informational	Resolved
8	`mint` module's parameters `prev_block_timestamp` and `max_mintable_nanoseconds` should be defined as unsigned integers	Informational	Resolved
9	Inconsistent package name in `protobuf` built golang types	Informational	Resolved
10	The `tax` module implements a `CLI` for transaction commands even if it has no transactions defined	Informational	Resolved
11	The `mint` module's `BeginBlocker` could circuit break earlier when the total amount of tokens has already been minted	Informational	Resolved

Code Quality Criteria

Criteria	Status	Comment
Code complexity	**Low-Medium**	-
Code readability and clarity	**Low-Medium**	Comments are not always adherent to the implementation.
Level of documentation	**Medium**	The `mint` module had no technical documentation at the beginning of the audit but was provided during the engagement.
Test coverage	**Medium**	60.8% test coverage for the `mint` module. 55.9% test coverage for the `tax` module.

Detailed Findings

1. Unbounded iteration in the `tax` module's `AnteHandler` could be used by an attacker to slow down or halt the chain

Severity: Major

In `x/tax/keeper/taxdecorator.go:76-81`, the `ApplyTax` function called from the `tax` module's `AnteHandler` is performing an unbounded iteration over the `feeCoins` provided by users.

An attacker could craft a message with a significant number of `Coins` with the intention of slowing down the block production, which in extreme cases may lead to Tendermint's propose timeout to be surpassed.

This can prevent the node from processing further `ABCI` messages such that it has to pause and contact peers to get the latest correct blocks. If a significant number of peers hit the timeout and halt simultaneously, block production may stop.

A test case is provided in Appendix 1.

Recommendation

We recommend implementing a guard which returns an error if duplicated denom `Coins` are provided and defining a maximum number of `Coins` that can be supplied.

Status: Resolved

2. `ValidateMinter` does not guarantee that provided parameters are adherent to the defined minting schedule

Severity: Minor

In `x/mint/types/minter.go:43`, the `ValidateMinter` function does not verify that the parameters passed on genesis are conformant with the equation that defines the minting schedule. There is a risk that one or more of them are not correct, which would lead to a flawed minting schedule.

Once this happens, there is no straightforward way to return to the intended trajectory. It is important to note, nevertheless, that there are checks to ensure that the total minted amount does not exceed a predefined threshold.

Recommendation

We recommend ensuring that the `Minter`'s parameters define an equation that converges to the total amount to be minted in the defined period.

Status: Resolved

3. Silently handled broken invariant could lead to inconsistent chain state

Severity: Minor

In `x/mint/abci.go:41-44`, during the calculation of the time interval between two subsequent blocks, the guard that checks that the timestamp of the current block is greater than the previous one does not panic or return an error. Instead, it assigns blocks with the same timestamp, letting the execution flow continue.

Since the code should always enforce the invariant

$$nsecBetweenBlocks > 0$$

incorrect subsequent block order or simultaneous blocks should result in an error and eventually halt the chain.

We classify this as a minor issue since `Tendermint` is already enforcing this invariant.

Recommendation

We recommend reworking the guard mentioned above to panic in case of broken invariants.

Status: Resolved

4. The `feeCaps` parameter is not validated

Severity: Minor

In `x/tax/types/params.go:103-113`, the `validateFeeCaps` function that is invoked by `ValidateBasic` is not validating the `feeCaps` parameter. Instead, a `TODO` comment is present as a reminder to implement the validation logic.

Consequently, every string value could be provided as `feeCaps` parameter.

Recommendation

We recommend implementing the `feeCaps` parameter validation.

Status: Resolved

5. Minting cap is not enforced during genesis

Severity: Minor

In `x/mint/types/minter.go:48`, the `ValidateMinter` function ensures that the `TotalMinted` parameter is positive, but it does not validate that the value is less than the `MintingCap`.

This implies that during genesis, a `TotalMinted` parameter value that exceeds the `MintingCap` amount is accepted and stored.

Recommendation

We recommend enforcing the `TotalMinted` parameter value to be less than the defined `MintingCap`.

Status: Resolved

6. The `FixedMintedAmount` parameter value is incoherent with the documentation

Severity: Minor

In `x/mint/types/minter.go:11`, `FixedMintedAmount` is set to `103125` which implies that 0.0825% of the total amount of tokens is minted per year.

However, the docs define this number to be 0.08% which is incoherent with the implementation and could lead to an unintended minting schedule.

Recommendation

We recommend either updating the docs to match the implementation or replace the existing `FixedMintedAmount` with `100000`.

Status: Resolved

7. Inefficient calculation of `remainingFees`

Severity: Informational

In `x/tax/keeper/taxdecorator.go:81`, the `SafeSub` method, which has `O(n)` asymptotic complexity, is used to calculate `remainingFees`.

However, since this variable is not used and `Coins` cannot be negative because the value of `feeRate` is ensured to be in the `[0,100)` range, this calculation is not needed.

Recommendation

We recommend removing the statement in `x/tax/keeper/taxdecorator.go:81` in order to improve code efficiency.

Status: Resolved

8. `mint` module's parameters `prev_block_timestamp` and `max_mintable_nanoseconds` should be defined as unsigned integers

Severity: Informational

In `proto/nolus/mint/v1beta1/mint.proto` the `prev_block_timestamp` and `max_mintable_nanoseconds` parameters are defined of type `int64`.

Since they contain UNIX timestamp information that is represented only by positive values, it is advisable to directly define them as `uint`.

Recommendation

We recommend modifying the type of `prev_block_timestamp` and `max_mintable_nanoseconds` parameters to `uint`.

Status: Resolved

9. Inconsistent package name in `protobuf` built golang types

Severity: Informational

In the following `protobuf` built files:

- `x/tax/types/genesis.pb.go:72`
- `x/tax/types/params.pb.go:87`
- `x/tax/types/query.pb.go:117-118`
- `x/tax/types/query.pb.go:172`
- `x/tax/types/query.pb.go:207`
- `x/tax/types/query.pb.go:216`
- `x/tax/types/tx.pb.go:75`

the code is referring to the `veselin.venus` package instead of the correct `nomo.cosmzone` one.

Since the defined package does not exist in the project, it cannot be reached.

Recommendation

We recommend rebuilding the `protobuf` in order to let them refer to the correct package name and rework the build pipeline if needed.

Status: Resolved

10. The `tax` module implements a `CLI` for transaction commands even if it has no transactions defined

Severity: Informational

In `x/tax/client/cli/tx.go`, the `CLI` command handler for submitting transactions is defined. Since the `tax` module does not define and support transactions, this is not needed and may be confusing for users.

Recommendation

We recommend removing `CLI` transaction commands from the `tax` module.

Status: Resolved

11. The `mint` module's `BeginBlocker` could circuit break earlier when the total amount of tokens has already been minted

Severity: Informational

In `x/mint/abci.go:106`, the `BeginBlocker` could avoid performing extra computations when the module already has minted the total amount of tokens.

Since the function has early access to the `Minter`, it is possible to perform a validation as soon as it retrieves from the storage. Then, `calcTokens` will not be called unnecessarily.

Recommendation

We recommend adding an early check that returns if the total minted amount is equal to the `MintingCap`.

Status: Resolved

Appendix

1. Test case for "Unbounded iteration in the `tax` module's `AnteHandler` could be used by an attacker to slow down or halt the chain"

```
func (suite *KeeperTestSuite) TestTaxesOverload() {
	suite.SetupTest(true) // setup
	suite.txBuilder = suite.clientCtx.TxConfig.NewTxBuilder()

	// keys and addresses
	priv1, _, addr1 := sdktestutil.KeyTestPubAddr()

	// msg and signatures
	msg := sdktestutil.NewTestMsg(addr1)

	fmt.Println("Preparing...")
	var feeAmount sdk.Coins
	for i := 0; i < 1000000000; i++ {
		denom := "atom" + strconv.FormatInt(int64(i), 10)
		feeAmount = feeAmount.Add(sdk.NewCoin(denom, sdk.NewInt(1)))
		err := simapp.FundAccount(suite.app.BankKeeper, suite.ctx,
addr1, sdk.NewCoins(sdk.NewCoin(denom, sdk.NewInt(10))))
		suite.Require().NoError(err)
	}
	fmt.Println("Sending transaction...")
	gasLimit := sdktestutil.NewTestGasLimit()
	suite.Require().NoError(suite.txBuilder.SetMsgs(msg))
	suite.txBuilder.SetFeeAmount(feeAmount)
	suite.txBuilder.SetGasLimit(gasLimit)

	privs, accNums, accSeqs := []cryptotypes.PrivKey{priv1},
[]uint64{0}, []uint64{0}
	tx, err := suite.CreateTestTx(privs, accNums, accSeqs,
suite.ctx.ChainID())
	suite.Require().NoError(err)

	acc := suite.app.AccountKeeper.NewAccountWithAddress(suite.ctx,
addr1)
	suite.app.AccountKeeper.SetAccount(suite.ctx, acc)

	dfd := ante.NewDeductFeeDecorator(suite.app.AccountKeeper,
suite.app.BankKeeper, nil)
```

```
	dtd := keeper.NewDeductTaxDecorator(suite.app.AccountKeeper,
suite.app.BankKeeper, suite.app.TaxKeeper)
	antehandler := sdk.ChainAnteDecorators(dfd, dtd)

	_, err = antehandler(suite.ctx, tx, false)
	fmt.Println(err)
	suite.Require().NoError(err)
}
```