Audit Report

Noble

v1.0

March 13, 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 Strangelove Crypto, Inc. to perform a security audit of Noble.

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 repository:

https://github.com/strangelove-ventures/noble

Commit hash: `b6697f36245b536888ac3424ea1e68f832f73a95`

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

Noble is a blockchain built using Cosmos SDK and Tendermint. This blockchain is expected to be deployed using Interchain Security provided by the Cosmos Hub. The core module is the `TokenFactory` module that allows generic assets to be minted and controlled by privileged accounts.

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.

Code Quality Criteria

The auditor team assesses the codebase's code quality criteria as follows:

Criteria	Status	Comment
Code complexity	Low-Medium	-
Code readability and clarity	Medium-High	Readability is generally in line with most Cosmos SDK based blockchain applications.
Level of documentation	Medium-High	-
Test coverage	Low-Medium	Go test reports a 35.5% test coverage excluding simulation packages and automatically generated files.

Summary of Findings

No	Description	Severity	Status
1	Blacklist policy is not enforced for other chain addresses	Major	Resolved
2	Minting denom can be transacted through IBC during a paused state	Major	Resolved
3	IBC middleware does not parse denom's trace when receiving ICS-20 packet data	Major	Resolved
4	Incorrect `minterController` genesis state validation	Major	Resolved
5	Deviation from specification: A controller can set up any number of minters	Major	Resolved
6	Separation of privileged addresses is not enforced	Minor	Resolved
7	Modifying minting denom allows blacklisted users to transact the old denom	Minor	Resolved
8	Minters can be added during a paused state	Minor	Resolved
9	`Amino` codec must be registered to support hardware devices like Ledger	Minor	Resolved
10	Incomplete genesis validation	Minor	Resolved
11	Missing validation for existing blacklisted address	Minor	Resolved
12	Missing blacklist validation during minter configuration	Minor	Acknowledged
13	Tokenfactory does not perform two-step ownership transfer	Minor	Resolved
14	Succinct event emission restricts information available to indexers	Minor	Acknowledged
15	`Amount` and `Allowance` message attributes are not validated to be greater than zero	Minor	Resolved
16	Incorrect field numbers	Informational	Resolved
17	`InitChainer` and `BeginBlocker` are set twice	Informational	Resolved
18	Unnecessary validation of `From` address	Informational	Acknowledged

19	Duplicated `ValidateBasic` invocation in CLI	Informational	Resolved
20	Inefficient use of `GetDenomMetaData`	Informational	Acknowledged
21	Miscellaneous Comments	Informational	Resolved

Detailed Findings

1. Blacklist policy is not enforced for other chain addresses

Severity: Major

In `x/tokenfactory/blockibc_middleware.go:111-115`, the sender address is checked to not be blacklisted when performing an IBC token transfer. Since ICS-20 token transfers involve escrowing USDC to other chains, the blacklist policy should ensure that other blacklisted chain addresses cannot redeem the locked USDC with their IBC vouchers.

However, this is not possible due to the `AccAddressFromBech32` validation in `x/tokenfactory/types/message_blacklist.go:45-48`, which can only validate native `Bech32` prefix addresses. This implies that addresses on other chains can redeem USDC independent of a blacklist, since they cannot be blocked by the blacklist.

For example, the following command fails when trying to blacklist a Juno address:

```
nobled --home $CHAINDIR/$CHAINID tx tokenfactory blacklist
juno1t8ehvswxjfn3ejzkjtntcyrqwvmvuknzy3ajxy --from $(nobled keys
show blacklister -a) -y
```

```
Error: invalid blacklist address (invalid Bech32 prefix; expected
noble, got juno): invalid address
```

This issue also affects the receiver blacklist policy in `app/ante.go:89-93` when initiating an ICS-20 token transfer. Consequently, the blacklister will not be able to prevent users from transferring USDC to malicious addresses on other chains.

We classify this issue as major because the blacklist policy is considered an important security feature in Noble's architecture.

Recommendation

We recommend implementing a blacklist that works on public keys rather than `Bech32` encoded addresses. This allows all addresses generated from one public key to be blocked, rather than individual encodings that are chain-specific.

Status: Resolved

2. Minting denom can be transacted through IBC during a paused state

Severity: Major

In `x/tokenfactory/blockibc_middleware.go:92-114`, the `OnRecvPacket` handler does not validate the keeper's paused state when receiving ICS-20 fungible token transfer packets. When the keeper's paused state is `true`, all internal and external transfers are disallowed, as seen in `app/ante.go:39-43`. However, other chains can still transact USDC tokens to the native chain, effectively bypassing the paused state.

We classify this issue as major because the paused state is considered an important security feature in Noble's architecture.

Recommendation

We recommend verifying the chain is not in a paused state when receiving IBC USDC packet data.

Status: Resolved

3. IBC middleware does not parse denom's trace when receiving ICS-20 packet data

Severity: Major

In `x/tokenfactory/blockibc_middleware.go:101-103`, the blacklist security measure will be skipped if the packet's denom is not the configured minting denom. As mentioned in lines `83-85`, this is to ensure that the sender and receiver are not blacklisted if they were to transact an IBC voucher representation of USDC.

However, this approach will never work because the receiving packet's denomination is constructed with a record of channels for which they have been transferred across chains, such as `transfer/channel-40/uusdc`. Consequently, the condition in line `101` will always evaluate as `true`, effectively bypassing the blacklist policy in lines `105-115`.

Recommendation

We recommend parsing the denom's trace to compare the base denom with the minting denom to ensure the blacklist policy works as expected.

Status: Resolved

4. Incorrect `minterController` genesis state validation

Severity: Major

In `x/tokenfactory/types/genesis.go:54`, the `MinterControllerKey` function is used to derive the minter controller key from a minter controller address. However, the minter controller address is passed as an `elem.Minter` in the current implementation. This sets an incorrect minter controller key to validate a duplicated index. Therefore, as the specified key does not contain any data, accessing data onchain would not be possible.

Recommendation

We recommend passing `elem.Controller` instead of `elem.Minter` in `x/tokenfactory/types/genesis.go:54`.

Status: Resolved

5. Deviation from specification: A controller can set up any number of minters

Severity: Major

According to the specification, the `ConfigureMinter` function in `x/tokenfactory/keeper/msg_server_configure_minter.go:12` should only be called by the `minterController` and set its corresponding minter. However, the current implementation of `ConfigureMinter` does not validate that the minter address being configured matches the specified address that the `MinterController` controls. The same issue exists in the `RemoveMinter` function in `x/tokenfactory/keeper/msg_server_remove_minter.go:12`. Because of that, the relationship between `MultiController` and minter becomes one-to-many instead of one-to-one. This deviates from the specification.

Recommendation

We recommend adding a check in both `ConfigureMinter` and `RemoveMinter` functions to validate `msg.Address == minterController.minter`.

Status: Resolved

6. Separation of privileged addresses is not enforced

Severity: Minor

While there has been an extensive effort made to develop privileged accessed roles within the Noble chain, privilege separation of these roles is not properly enforced. Currently, there is no validation to ensure that these privileges cannot be shared by a single address. In the unlikely case of a compromised account, account sharing may have severe implications.

For example, the Owner can assign the Blacklister, Master Minter, and Pauser roles to itself.

While new addresses can easily be generated and all roles can technically be owned by the same entity using different addresses, it is best practice to restrict address reuse.

Another step that can be taken to reduce the impact of a compromised account is to implement timelocks to prevent frequent updates of addresses.

Recommendation

We recommend adding validation to prevent privileged address reuse and consider implementing timelocks following the update of these privileged addresses.

Status: Resolved

7. Modifying minting denom allows blacklisted users to transact the old denom

Severity: Minor

In `x/tokenfactory/genesis.go:45-47`, supplying a different `MintingDenom` will cause the `SetMintingDenom` function to modify the mint denom. This is problematic since the blacklist functionality only works based on the current minting denom, as the decorator in `app/ante.go:67-98` only validates that the addresses are not blacklisted if the transacted denom is the configured minting denom.

Consequently, blacklisted users who were previously unable to transact the old minting denom can now transact that denom like other users. Additionally, all minters would need to update their old allowances' denom to continue minting new tokens.

We classify this issue as minor because only the governance can cause it, which requires cooperation among impacted parties.

Recommendation

We recommend ensuring the minting denom can only be set once.

Status: Resolved

8. Minters can be added during a paused state

Severity: Minor

In `x/tokenfactory/keeper/msg_server_configure_minter_controller.go:12` and `x/tokenfactory/keeper/msg_server_configure_minter.go:12`, the

`ConfigureMinterController` and `ConfigureMinter` keeper functions can be executed even if the keeper's `paused` state is set to `true`. This is inconsistent with the pausing section of the token design specification.

As mentioned in that specification, adding minters should be prevented while the contract is paused.

Recommendation

We recommend adding a check to ensure that the chain is not paused in the `ConfigureMinterController` and `ConfigureMinter` keeper functions.

Status: Resolved

9. `Amino` codec must be registered to support end users with hardware devices like Ledger

Severity: Minor

In `x/tokenfactory/types/codec.go:77`, `Amino` should be used to register all interfaces and concrete types for the `tokenfactory` module. This is necessary for JSON serialization to support hardware devices like Ledger since these devices do not support proto transaction signing.

Recommendation

We recommend registering all interfaces and concrete types for the `tokenfactory` module using `Amino` to support JSON serialization.

Status: Resolved

10. Incomplete genesis validation

Severity: Minor

In `x/tokenfactory/types/genesis.go:29-63`, the `Validate` function is not validating all the provided `GenesisState` attributes.

There is no logic in place that validates `Pauser`, `Owner` and `Blacklister` addresses and that enforces `Denom` to be a non-empty string.

Recommendation

We recommend implementing validation of all the `GenesisState` attributes.

Status: Resolved

11. Missing validation for existing blacklisted address

Severity: Minor

In `x/tokenfactory/keeper/msg_server_blacklist.go`, the `Blacklist` function is used for adding new addresses, however in its current implementation, it is not verified whether a new address is already on a blacklist or not. As a result, the operation succeeds without an error, which may give indexers or other off-chain services the misleading impression that a specific address has not yet been blacklisted.

Recommendation

We recommend adding validation that will revert the transaction if the address already exists within the store.

Status: Resolved

12. Missing blacklist validation during minter configuration

Severity: Minor

In `x/tokenfactory/keeper/msg_server_configure_minter.go`, the `ConfigureMinter` method is used to add a minter. However, its current implementation does not check if the provided `msg.Address` is on a blacklist. This allows adding a blacklisted address as a minter.

Recommendation

We recommend adding a check such as the following:

```
_, found := k.GetBlacklisted(ctx, msg.Address)
if found {
  return nil, sdkerrors.Wrapf(types.ErrUnauthorized, "Provided
minter is blacklisted")
}
```

Status: Acknowledged

13. Tokenfactory does not perform two-step ownership transfer

Severity: Minor

The `UpdateOwner` function in `x/tokenfactory/keeper/msg_server_update_owner.go:12` does not perform a

two-step ownership transfer. In the unlikely event that the new owner is set to an incorrect address, then the owner will not be able to be reset, and the owner functionality will be lost.

Recommendation

We recommend implementing a two-step ownership transfer process where the current owner proposes a new owner. Then that new owner passes a separate message to accept ownership. Only after the ownership has been accepted is the state change actually made with `SetOwner`.

Status: Resolved

14. Succinct event emission restricts information available to indexers

Severity: Minor

All defined messages emit `TypedEvent` events containing the submitted message. However, those events contain only limited detail and may not represent enough information for indexers and other off-chain services.

Recommendation

We recommend emitting more detailed events.

Status: Acknowledged

15. `Amount` and `Allowance` message attributes are not validated to be greater than zero

Severity: Minor

In the `ValidateBasic` functions in

- `x/tokenfactory/types/message_burn.go:45`,
- `x/tokenfactory/types/message_configure_minter.go:41`, and
- `x/tokenfactory/types/message_mint.go:41`,

the `Amount` and `Allowance` attributes are not validated to be greater than zero.

This implies that messages that contain invalid values can pass the `ValidateBasic` validation without triggering an error.

Recommendation

We recommend adding validation for `Amount` and `Allowance` attributes.

Status: Resolved

16. Incorrect field numbers

Severity: Informational

The `GenesisState` message in `proto/tokenfactory/genesis.proto` uses an incorrect field number starting from `9`. The `minterControllerList` should have `9`, and `mintingDenom` should have `10` field numbers.

Recommendation

We recommend correcting these field numbers.

Status: Resolved

17. `InitChainer` and `BeginBlocker` are set twice

Severity: Informational

In `app/app.go:571-572` and `app/app.go:593-594`, the `SetInitChainer` and `SetBeginBlocker` methods are called twice, which is inefficient.

Recommendation

We recommend removing the duplicated method invocations.

Status: Resolved

18. Unnecessary validation of `From` address

Severity: Informational

Additional validation for the `From` address in `ValidateBasic` functions in the lines below are not necessary since that address belongs to the transaction sender address and is already validated:

- `x/tokenfactory/types/message_blacklist.go:41`
- `x/tokenfactory/types/message_burn.go:41`
- `x/tokenfactory/types/message_configure_minter_controller.go:42`
- `x/tokenfactory/types/message_configure_minter.go:42`
- `x/tokenfactory/types/message_mint.go:42`
- `x/tokenfactory/types/message_pause.go:40`
- `x/tokenfactory/types/message_remove_minter_controller.go:41`
- `x/tokenfactory/types/message_remove_minter.go:41`
- `x/tokenfactory/types/message_unblacklist.go:41`

- x/tokenfactory/types/message_unpause.go:40
- x/tokenfactory/types/message_update_blacklister.go:41
- x/tokenfactory/types/message_update_master_minter.go:41
- x/tokenfactory/types/message_update_owner.go:41
- x/tokenfactory/types/message_update_pauser.go:41

Recommendation

We recommend removing the `From` address validation in these `ValidateBasic` functions.

Status: Acknowledged

19. Duplicated `ValidateBasic` invocation in `CLI`

Severity: Informational

All transaction CLI commands registered in the `GetTxCmd` function in `x/tokenfactory/client/cli/tx.go` call the `msg.ValidateBasic` function before calling `GenerateOrBroadcastTxCLI`. As `msg.ValidateBasic` is already called inside the `GenerateOrBroadcastTxCLI`, this is an unnecessary and duplicated invocation.

Recommendation

We recommend removing the duplicated `msg.ValidateBasic` invocation.

Status: Resolved

20. Inefficient use of `GetDenomMetaData`

Severity: Informational

In `x/tokenfactory/keeper/minting_denom.go:14`, the `GetDenomMetaData` function is used to validate whether the given denom is registered with the `bank` module or not. That function call not only proves that data exists though, but also returns unmarshalled data, which is inefficient, since that data is not required. In `bank v0.46.0`, the more efficient `HasDenomMetaData` function got introduced..

Recommendation

We recommend upgrading the bank module to `v0.46.0` and using `HasDenomMetaData` instead of `GetDenomMetadata`.

Status: Acknowledged

21. Miscellaneous Comments

Severity: Informational

The following are some recommendations to improve the overall code quality and readability:

- Remove the unused function `IsProposalWhitelisted` in `app/proposals_whitelisting.go:12`.
- Remove the unused function `RemoveBlacklister` in `x/tokenfactory/keeper/blacklister.go:30`.
- Remove the unused function `RemoveMintingDenom` in `x/tokenfactory/keeper/minting_denom.go:37`.
- Remove the unused code in `app/proposals_whitelisting.go:169` and `x/tokenfactory/client/cli/tx.go:14-21`.
- Remove unused imports in `x/tokenfactory/client/cli/query.go` and `x/tokenfactory/client/cli/tx.go`.
- Remove the unused file `x/tokenfactory/types/types.go`.
- Remove the unhandled error in `x/tokenfactory/keeper/msg_server_burn.go:41`.
- Fix the typos in the comments in `x/tokenfactory/blockibc_middleware.go:17` and `x/tokenfactory/blockibc_middleware.go:85`.
- Fix the inaccurate comments that refer to 'all' in the `ExportGenesis` function in `x/tokenfactory/genesis.go:53`.
- Fix the incorrect returning error in `x/tokenfactory/keeper/msg_server_burn.go`. The error should return `types.ErrBurn`.
- Fix the incorrect spelling in `x/tokenfactory/keeper/grpc_query_minters.go:31`.

Status: Resolved