Audit Report

XION and Burnt Contracts

v1.1

September 22, 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 Thames Brook Associates, LLC to perform a security audit of the XION Cosmos SDK chain and the Burnt CosmWasm smart contracts.

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 target:

Repository	https://github.com/burnt-labs/xion
Identifier	In this report, all paths pointing to this repository are prefixed with `xion:`
Commit	`432e5d73e8bfc0e88de22dda206f405542ee7c91`
Scope	All code in the repository was in scope.

Repository	https://github.com/burnt-labs/burnt-cw-std
Identifier	In this report, all paths pointing to this repository are prefixed with `cw-std:`

Commit	`ee68c1a0e9b793c5b7f1d01ca96bf774933f0c6f`
Scope	All code in the repository was in scope.

Repository	https://github.com/burnt-labs/burnt-cw-hubs
Identifier	In this report, all paths pointing to this repository are prefixed with `cw-hubs:`
Commit	`f840b6b09fe6227e11cf06007d60546cf641c57f`
Scope	All code in the repository was in scope.

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

This audit covers the functionality associated with the CosmosSDK app chain XION, and its associated hub and seat contracts along with a set of XION-specific CosmWasm standard libraries to assist smart contract developers in creating contracts on the XION app chain.

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	Medium	The complexity of the underlying code was relatively low, but the library design of the CosmWasm contracts increased the complexity rating of the codebase from low-medium to medium.
Code readability and clarity	Medium	The code was readable and clear but lacked descriptive code comments.
Level of documentation	Medium	While the Cosmos SDK modules contained some documentation, the CosmWasm library and contracts did not contain sufficient documentation to describe their functionality and expected behavior.
Test coverage	Medium	-

Summary of Findings

No	Description	Severity	Status
1	`check_ownable` is incorrectly implemented which allows attackers to list any tokens	Critical	Resolved
2	Anyone can lock any `token_id`	Critical	Resolved
3	Anyone can halt any ongoing sale	Critical	Resolved
4	`buy_item` will error when users supply the correct amount and denom	Critical	Resolved
5	Owner can be set to an invalid address	Major	Resolved
6	Primary sales can be incorrectly configured	Major	Resolved
7	Items can be bought from disabled or ended unlimited sales	Major	Resolved
8	Wrong condition leads to disabled ongoing sales appearing as active	Major	Resolved
9	`try_list` is lacking validation	Minor	Resolved
10	Incorrect funds will cause contract to panic	Informational	Resolved
11	Incorrect query endpoint	Informational	Resolved
12	Code quality could be improved	Informational	Resolved
13	Misleading telemetry information	Informational	Resolved
14	Additional funds sent to the contract are lost	Informational	Resolved
15	Incorrect error messages	Informational	Resolved

Detailed Findings

1. `check_ownable` is incorrectly implemented which allows attackers to list any tokens

Severity: Critical

In `cw-std:sellable/src/execute.rs:40` and `263`, the `check_ownable` validation function returns `Ok` in all conditions, even if the caller is not the owner of the NFT token. This means that a malicious actor is able to bypass validation measures, allowing them to list any NFT tokens at an extremely low price and subsequently purchase them.

Recommendation

We recommend fixing the `check_ownable` validation function to return an error on validation failure.

Status: Resolved

2. Anyone can lock any `token_id`

Severity: Critical

The `redeem_item` function in `cw-std:redeemable/src/execute.rs:11` lacks an authorization check to validate whether the sender is the `token_id` owner or the contract owner. This allows anyone to lock any `token_id` in `self.locked_items`.

Recommendation

We recommend adding an authorization check in `redeem_item`. Depending on the intended design, we recommend ensuring that the sender is either the token's owner or the contract owner. This function is not documented so we cannot provide exact guidance on the intended caller.

Status: Resolved

3. Anyone can halt any ongoing sale

Severity: Critical

The `halt_sale` function in `cw-std:sales/src/execute.rs:59` is permissionless. This means that any user can halt any ongoing sale. Attackers can take advantage of this to strategically halt sales or to grief users.

Recommendation

We recommend validating that `info.sender` is equal to `self.sellable.ownable` before proceeding with the halt sale functionality.

Status: Resolved

4. `buy_item` will error when users supply the correct amount and denom

Severity: Critical

The `buy_item` function in `cw-std:sales/src/execute.rs:91` contains a condition that will cause the function to error when the correct denom and amount are sent. There are a number of issues with the funds validation in this function. First, the condition in line `91` is actually inverted, it will currently return a `WrongFundError` when the correct funds are sent. This will effectively prevent any user who is correctly supplying funds from buying the item. Next, the `sale.price.contains` validation is incorrectly used. The `contains` method on a vector of coins will actually only return true for a complete match of funds. This means that it will only return true if both the denom and amount are equal.

The combination of these issues will result in a situation where users will receive a `WrongFundError` when they are correctly passing a valid `BuyItem` message. Additionally, in the current implementation, transactions that supply the wrong denom will actually pass the validation and cause a panic in line `98`.

Recommendation

We recommend updating the validation in the `buy_item` to check by denom if the `info.funds` denom is equal to the `sale.price` denom and returning an error on the other conditions. An example implementation of this could be:

```
funds_vec.iter().find(|&coin| coin.denom.eq(&price_coin[0].denom))
```

Status: Resolved

5. Owner can be set to an invalid address

Severity: Major

The `ownable` contract's `SetOwner` message in `cw-std:ownable/src/lib.rs:106` does not validate that `msg.owner` is a valid address. This is problematic because the `Addr` type always must contain valid addresses. `Addr` is intended to only be used in code after it has undergone validation. In this instance, if the address is ever set to an invalid address, the

owner value of the contract will no longer be updatable as there is an authorization check where the caller is checked and `info.sender` will never match the invalid address.

Recommendation

We recommend first validating the owner value with `addr_validate` before storing the address.

Status: Resolved

6. Primary sales can be incorrectly configured

Severity: Major

The `add_primary_sales` function in `cw-std:sales/src/execute.rs:47` will allow for overlapping primary sales to be introduced during the instantiation. While the function validates `msg.start_time`, it does not validate `msg.end_time`. The `add_primary_sales` function operates with the assumption that sales are non-overlapping and have valid durations with regard to their start and end times. If the contract is improperly instantiated, then the function would also fail to prevent additional overlapping sales from being introduced when the `PrimarySale` message is received.

Recommendation

We recommend validating during instantiation that the full period of the primary sale does not overlap, including end times.

Status: Resolved

7. Items can be bought from disabled or ended unlimited sales

Severity: Major

The `buy_item` function on line `cw-std:sales/src/execute.rs:85` contains a compound conditional statement that will allow users to buy items even when the sale is disabled and ended. If the sale has a total supply of `0`, it will override all other conditions and proceed into the function. Additionally, this function was not documented and did not contain code comments that detailed its intended valid states.

Recommendation

We recommend fixing the condition to handle the case of an unlimited but disabled or ended sale. Additionally, we recommend documenting the intended conditions to be met.

Status: Resolved

8. Wrong condition leads to disabled ongoing sales appearing as active

Severity: Major

The `active_primary_sales` function in `cw-std:sales/src/query.rs:21-22` contains a compound condition where the second condition will always override the first condition. The result of this is that as long as the sale is ongoing, it will be returned as an active sale to the query even if it is disabled. This can mislead the entity querying the contract and cause them to act with incorrect information.

This logical error is also present in `cw-std:sales/src/execute.rs:63`, which will allow the `halt_sale` function to disable an already disabled sale so long as it is currently ongoing.

Recommendation

We recommend updating this condition to correctly determine if the sale is disabled and act appropriately in both of the locations mentioned above.

Status: Resolved

9. `try_list` is lacking validation

Severity: Minor

There are a number of validations that can be implemented in `try_list` to avoid possible error states. In the `try_list` function in `cw-std:sellable/src/execute.rs:51`, there is no validation to check that the listing supplied is not already contained in the `listed_tokens`. In the case that a duplicate is supplied, it will silently overwrite the price of the existing token.

Additionally, in `cw-std:sellable/src/execute.rs:43`, listings with a price of `0` will be silently skipped and not listed. If a price of `0` is supplied, it is best practice to return an error and revert the entire transaction rather than saving a partial state that may introduce misconfigured listings.

This also applies to the `try_list` function implemented by `RSellable` in line `253`, but `RSellable` is not currently implemented.

Recommendation

We recommend implementing the fixes mentioned above in both of the `try_list` functions.

Status: Resolved

10. Incorrect funds will cause contract to panic

Severity: Informational

In the `buy_item` function in `cw-std:sales/src/execute.rs:94-99`, the function attempts to find the payment denom in `info.funds`. If the denom is not found, then the function will panic. This state is currently reachable because the validation in line `92` will not error if incorrect funds are present. See buy_item will error when users supply the correct amount and denom. Panicking goes against best practices since a panic does not provide an insightful and actionable error message to the user.

Recommendation

We recommend refactoring the code to return an error if incorrect funds are supplied.

To handle this properly, the `.find()` call should be replaced with:

```
let paying_fund = sale.price.iter().find(|coin|coin.denom==
info.funds[0].denom).ok_or(ContractError::WrongFundError)?;
```

This will return an `Err(ContractError::WrongFundError)` if no match is found, rather than panicking.

Status: Resolved

11. Incorrect query endpoint

Severity: Informational

In `xion:proto/xion/mint/v1/query.proto:15`, the `Params` gRPC querier service endpoint is incorrect. It should be `/xion/mint/v1beta1/params`.

Recommendation

We recommend correcting the gRPC querier service endpoint.

Status: Resolved

12. Code quality could be improved

Severity: Informational

Across the codebase, instances of unused and commented-out code have been found. Unused and commented-out code decreases the maintainability of a codebase.

Recommendation

We recommend improving the code quality by applying the following suggestions:

- Remove commented-out code in `xion:x/xion/module.go:63:66`
- Remove commented-out code in `xion:x/xion/module.go:76`
- Remove commented-out code in `xion:x/xion/module.go:100`
- Remove unused struct in `cw-hubs:hub/src/state.rs:21`
- Remove the `CONFIG` and `CONTRACT_INFO` states in `cw-hubs:hub/src/state.rs:196-197`, they are not used anywhere since the contract uses `cw2` to manage the contract version.
- Remove the `CONTRACT_INFO` state in `cw-hubs:seat/src/state.rs:296`, it is not used anywhere since the contract uses `cw2` to manage the contract version.
- Remove the unused `HUB_CONTRACT` in `cw-hubs:seat/src/state.rs:88`
- Remove the unused `SEAT_CONTRACT` in `cw-hubs:hub/src/state.rs:91`
- Remove the unused `set_owner` in `cw-std:ownable/src/lib.rs:74`

Status: Resolved

13. Misleading telemetry information

Severity: Informational

The telemetry information emitted by the `xion` module's `Send` message in `xion:x/xion/keeper/msg_server.go:71` emits the original amount, but the actual amount sent to the recipient has a fee deducted. This may confuse or mislead users.

Recommendation

We recommend emitting the amount after the fee has been deducted.

Status: Resolved

14. Additional funds sent to the contract are lost

Severity: Informational

In `cw-std:sellable/src/execute.rs:161` and `388`, a check is performed that ensures that in the transaction there is a `Coin` with the expected `denom` field.

This validation does not ensure however that no other native tokens have been sent, and any such additional native tokens are not returned to the user, so they will be stuck in the contract forever.

While blockchains generally do not protect users from sending funds to wrong accounts, reverting extra funds increases the user experience.

Recommendation

We recommend checking that the transaction contains only the expected `Coin` and no additional native tokens using https://docs.rs/cw-utils/latest/cw_utils/fn.must_pay.html.

Status: Resolved

15. Incorrect error messages

Severity: Informational

There are several instances of incorrect/misleading errors being returned in the codebase:

- In `cw-std:sellable/src/execute.rs:158` and `383`, `NoFundsPresent` is returned when the denomination of the token is not set in storage.
- In `cw-std:sellable/src/execute.rs:54` and `278`, `NoMetadataPresent` is returned when `token_id` does not exist in the contract.
- In `cw-std:sellable/src/execute.rs:78` and `302`, `NoMetadataPresent` is returned when `token_id` does not exist in the contract.

Recommendation

We recommend correcting these errors.

Status: Resolved