Audit Report

Stargaze Marketplace

v1.0

August 1, 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 Public Awesome LLC to perform a security audit of the Stargaze Marketplace 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 GitHub repository:

https://github.com/public-awesome/marketplace

Commit hash: `07ad739453965f1322c21d95478df601d78738d9`

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

Stargaze is a Cosmos dedicated zone for NFTs that supports CosmWasm smart contracts. One of the main pillars of this chain is its marketplace.

Stargaze marketplace is a protocol owned by the chain governance that allows users to sell and buy NFTs with both auctions and direct offers.

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	`payout` function may panic due to lack of fee and royalty validation	Major	Resolved
2	Misconfiguring stale bid duration causes operators to be unable to remove stale bids	Major	Resolved
3	Bidder can specify unchecked finders fee	Major	Resolved
4	Percentage and basis point values lack maximum value validation	Minor	Resolved
5	Sellers can update the price for inactive ask requests	Minor	Resolved
6	Bidders may set themselves as finder and receive finders fee	Minor	Resolved
7	Existing operators will be overwritten	Minor	Resolved
8	Large number of hooks may cause out of gas errors	Minor	Acknowledged
9	Release candidate dependencies	Minor	Resolved
10	`reserved_for` should only be used when the sale type is not an auction	Informational	Resolved
11	`map_validate` does not ensure the address vector is deduplicated	Informational	Resolved
12	Overflow checks not enabled for release profile	Informational	Resolved
13	Potentially unexpected hook behavior	Informational	Resolved

Code Quality Criteria

Criteria	Status	Comment
Code complexity	**Medium**	-
Code readability and clarity	**Medium-High**	-
Level of documentation	**Medium**	While the code was straightforward and easy to understand, the project did not have in-depth documentation and specification on the marketplace functionality and design.
Test coverage	**High**	cargo-tarpaulin reports a test coverage of 95.18%

Detailed Findings

1. `payout` function may panic due to lack of fee and royalty validation

Severity: Major

The `payout` function in `contracts/marketplace/src/execute.rs:890-946` handles the distribution of funds when a sale is being finalized. There are a number of parties that may be paid during the sale of an NFT, such as the seller, finder, artist, and the network. Each of these parties has a defined percentage they receive which is defined in various configuration parameters. The `payout` function does properly validate that the sum of each of these payment parameters is not greater than `100%`.

`finders_fee`, `network_fee`, and `royalty.share` should be validated to ensure that their sum is not greater than the payment. The amounts of `seller_share_msg`, `royalty`, `finders_fee`, and `network_fee` are all independently calculated off of the original payment value with no respect to their sum.

An example of a situation that would cause a panic is if a certain collection had an abnormally high `royalty.share`. For example, let's say the `royalty.share` is `95%`. The subtraction of `royalty.share`, `network_fee`, and `finders_fee` in line `931` would be greater than the original payment and cause a panic due to an underflow (the workspace has `overflow-checks` enabled, which causes underflows to panic).

Recommendation

We recommend ensuring that the sum of `finders_fee`, `network_fee`, and `royalty.share` is smaller than `100%`. At `100%`, the seller would receive nothing for sale.

Status: Resolved

2. Misconfiguring stale bid duration causes operators to be unable to remove stale bids

Severity: Major

During the contract instantiation process in `contracts/marketplace/src/execute.rs:49`, `msg.stale_bid_duration` is not validated to have the input cast as `Duration::Time`, which is inconsistent with updating of parameters during sudo message execution in `contracts/marketplace/src/sudo.rs:96-98`. As a result, a misconfiguration of initializing the stale bid duration as `Duration::Height` would cause the execution to fail in lines `745` and `796` because the `Duration` enum does not allow addition of time and height.

Recommendation

We recommend modifying the stale bid duration instantiation process to follow the `sudo_update_params` approach, where the input is of type `u64` instead of a `Duration` enum.

Status: Resolved

3. Bidder can specify unchecked finders fee

Severity: Major

`execute_accept_bid` and `execute_accept_collection_bid` in lines `contracts/marketplace/src/execute.rs:513` and `669` use the `bid.finders_fee_bps` if there is no existing ask when a seller accepts a bid. This is problematic because nothing stops a bidder from specifying a very high value. While the finder's fee would still go to the seller's specified finder, it may be a proportion that is higher than the seller expected.

When creating an ask, the finders fee is validated to ensure that it is less than `params.max_finders_fee_percent`, but this check doesn't occur in the cases mentioned above.

Example:

1. A user creates a bid with a very high finders fee of `90%` (high but not enough to error). They would need to create a bid on an NFT that doesn't currently have an ask.
2. Owner would accept the bid
3. If there is no ask, then `bid.finders_fee_bps` is used
4. The seller's specified finder would get nearly all of the value from the NFT sale, and the seller would not.

It is important to note that the caller still specifies the finder's address when accepting a bid. If this finder value was also pulled from the bid, this would be a critical issue allowing bidders to pay virtually nothing for NFTs.

Recommendation

We recommend implementing a check to ensure that the bidder specified `finders_fee_bps` is less than `params.max_finders_fee_percent` (see `contracts/marketplace/src/execute.rs:249-253`).

Status: Resolved

4. Percentage and basis point values lack maximum value validation

Severity: Minor

In `contracts/marketplace/src/execute.rs:47` and `50`, the values of `max_finders_fee_percent` and `bid_removal_reward_percent` are converted to `Decimal::percent` values but are not checked to ensure that they do not exceed `100%`. If either of these values were to be set to greater than `100%`, it would be harmful to the protocol. It is best practice to enforce a hard limit on percent values.

During the payout calculation, the finders fee could be greater than `100%` which will cause the calculations to underflow, but this would panic because of enable `overflow-checks`. Additionally, in `execute_remove_stale_bid`, a `bid_removal_reward_percent` greater than `100%` would result in a reward greater than the value of the original bid.

This situation is also present in the `sudo_update_params` function in `contracts/marketplace/src/sudo.rs:58`. `trading_fee_percent`, `max_finders_fee_percent`, and `bid_removal_reward_bps` should all be validated to ensure that they are not greater than `100%`.

Recommendation

We recommend performing validation on the parameters mentioned above to ensure that they cannot be set to values greater than `100%`.

Status: Resolved

5. Sellers can update the price for inactive ask requests

Severity: Minor

The `execute_update_ask_price` functionality in `contracts/marketplace/src/execute.rs:306` allows the seller to update their NFT asking price without validating whether the associated ask request is expired or inactive. This might cause a misconception to the seller that their ask request is still valid and available for bidders to bid, which may not be true.

Recommendation

We recommend validating that the asking request is not expired and still active before allowing the seller to update their NFT asking price.

Status: Resolved

6. Bidders may set themselves as finder and receive finders fee

Severity: Minor

In the `execute_set_bid` function in `contracts/marketplace/src/execute.rs:334`, when a bid is placed on an NFT with a `FixedPrice` auction, the finder is specified by the `set_bid` caller. That finder is then passed to the `finalize_sale` function and in the `payout` function, the address is then paid the finders fee. This function does contain any logic to verify that the finder address specified is not `info.sender`. This means that the function does not prevent bidders from specifying themselves as the finder and receiving the illegitimate finder fee.

Recommendation

We recommend performing validation on the finder address specified by the bidder in `execute_set_bid` to determine that it is not the same as `info.sender`. While this does not prevent the sender to specify an alternative account they own, it at least increases the barrier of gaming the system. To fully prevent misuse of the finder system, finders could be whitelisted and managed through governance, similar to how operators are managed.

Status: Resolved

7. Existing operators will be overwritten

Severity: Minor

The `sudo_update_params` function in `contracts/marketplace/src/sudo.rs:86-88` does not incrementally add operators, instead, it will overwrite the existing vector with the new operators. This may become problematic as the number of operators grows.

Recommendation

We recommend implementing functionality to specifically add and remove operators rather than overwriting the existing operator set.

Status: Resolved

8. A large number of hooks may cause out of gas errors

Severity: Minor

The `sudo_add_sale_hook`, `sudo_add_ask_hook`, and `sudo_add_bid_hook` functions in `contracts/marketplace/src/sudo.rs` do not impose a maximum number of hooks that may be added. As the marketplace scales, the number of hooks may begin to grow and cause out-of-gas errors. Note that this is recoverable because the contract contains hook removal functionality.

It is best practice to impose a maximum number of hooks as they increase the gas cost of the call.

Recommendation

We recommend implementing a maximum number of hooks and validating that the addition of new hooks does not exceed this value.

Status: Acknowledged

9. Release candidate dependencies

Severity: Minor

In `contract/marketplace/Cargo.toml:36-37`, the `cosmwasm-std` and `cosmwasm-storage` crates are specified at a release candidate version.

As `rc` software is by convention still in development we suggest not using it in production.

Recommendation

We recommend using `cosmwasm-std` and `cosmwasm-storage` in the latest stable version.

Status: Resolved

10. `reserved_for` should only be used when the sale type is not an auction

Severity: Informational

When bidders attempt to bid on an existing ask request via `execute_set_bid` functionality, the `reserved_for` value would be validated to ensure only the seller specified bidder can bid on the asked request as seen in `contracts/marketplace/src/execute.rs:378-382`. While this functionality is intended for sellers to reserve the sales for a specific bidder, it would defeat the purpose of the auction if the sales type is not fixed price.

Recommendation

We recommend only validating `reserved_for` to be the bidder if the sales type is fixed price and allowing any bidders to bid for an auction sales type.

Status: Resolved

11. `map_validate` does not ensure the address vector is deduplicated

Severity: Informational

The `map_validate` function in `contracts/marketplace/src/helpers.rs:28-33` does not ensure that the provided vector of addresses is deduplicated, which is inefficient.

Recommendation

We recommend ensuring that the `map_validate` properly ensures that the vector of addresses only contains one entry per address.

Status: Resolved

12. Overflow checks not enabled for release profile

Severity: Informational

While there are overflow checks at the workspace level, `contracts/marketplace/Cargo.toml` does not explicitly enable `overflow-checks` for the release profile.

While enabled implicitly through the workspace manifest, a future refactoring might break this assumption.

Recommendation

We recommend enabling overflow checks in all packages, including those that do not currently perform calculations, to prevent unintended consequences if changes are added in future releases or during refactoring. Note that enabling overflow checks in packages other than the workspace manifest will lead to compiler warnings.

Status: Resolved

13. Potentially unexpected hook behavior

Severity: Informational

There are several scenarios throughout the marketplace contract where actions may be performed without triggering the expected hooks. While this is not inherently a vulnerability, it may lead to unintended consequences if these scenarios are not explicitly considered when implementing hooks.

For example, a seller can call `SetAsk` multiple times to effectively update/overwrite their current ask. This will trigger the `HookAction::Create` hook multiple times, but not the

`HookAction::Delete` hook, which may be unexpected. The same issue is also found when setting bids.

Recommendation

While this behavior is not a vulnerability per se, we recommend clearly documenting it to ensure that governance and hook creators are fully aware of potential implications before creating hooks.

Status: Resolved