Audit Report

One Planet Smart Contracts

v1.0

February 8, 2022

Table of Contents

Disclaimer

This audit has been performed by

Moazzam Arif
https://www.linkedin.com/in/moazzam-arif

Introduction

Purpose of This Report

The Auditor(Moazzam Arif) has been engaged by Oak Security to perform an independent security audit of the One planet 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/ProjectOnePlanet/auction-contracts

Commit hash: `5aae62df01bed315ffe4efc738e62caacfa937b3`

https://github.com/ProjectOnePlanet/contract-queuing

Commit hash: `b6fb7b84388ae590e9e87b4b61cc446ad8f92ff7`

https://github.com/ProjectOnePlanet/contract-token

Commit hash: `3134f210d22973ed1572f20b5321d8614d84c3e1`

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

The submitted code implements an NFT launchpad where NFTs are sold using different auction strategies, such as English auction, Dutch auction and random box. NFT creators can mint NFTs and sell them in the marketplace and also collect royalty on them. In addition to the NFT contract, the codebase provides modules for queuing, bidding, auction house and royalty.

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 improve 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 a security audit and vice versa.

Summary of Findings

No	Description	Severity	Status
1	Royalty contract owners can steal royalty funds by changing the funds recipient address.	Critical	Resolved
2	Bidding contract owners can abuse their authority to front-run threshold values leading to unfair auctions	Critical	Resolved
3	Contract owner could steal funds	Critical	Resolved
4	Whitelisting in queuing contract can be bypassed by directly queuering the storage contract	Critical	Resolved
5	Change of storage contract address may lead to loss of funds.	Major	Resolved
6	Cyclic dependencies are present in auction_house and royalty contracts that would break the initialization of the respective contracts.	Major	Resolved
7	The non-atomic nature of palce_bid function may lead to locked funds.	Minor	Resolved
8	Incorrect nonce management.	Minor	Resolved
9	Inefficient update of the whitelist in queuing contract.	Minor	Resolved
10	Start time of the stage should be inclusive in the if statement	Minor	Resolved
11	Startime and endtime not validated during the instantiation of the queuing contract.	Minor	Resolved
12	Contract version is missing	Informational	Resolved
13	Ambiguous naming convention	Informational	Resolved
14	Unnecessary function arguments can be avoided	Informational	Resolved
15	Unnecessary asset ordering can be avoided	Informational	Resolved
16	Repetitive code can be avoided to minimise human error	Informational	Resolved
17	Validation of address should be done before	Informational	Resolved

	storing it on chain		
18	Implement Default trait instead of random dummy values	**Informational**	**Acknowledged**
19	Incorrect spelling of one of the contract error variants.	**Minor**	**Resolved**
20	Inefficient batch mint transaction.	**Minor**	**Resolved**
21	Inefficient code which can lead up to unnecessary usage of gas	**Minor**	**Resolved**

Code Quality Criteria

Criteria	**Status**	**Comment**
Code complexity	**Medium**	-
Code readability and clarity	**Medium-High**	-
Level of documentation	**Medium**	-
Test coverage	**Low**	There are almost no test cases present for many modules, We recommend having 100% test coverage.

Detailed Findings

1. Royalty contract owners can steal royalty funds by changing the funds recipient address.

Severity: Critical

Repo - `one-planet-auction-contract`
At - `contracts/royalty/src/contract.rs:126`

The `update_royalty()` function allows updating royalty parameters, i.e recipient address and royalty rate a NFT minter wishes to charge, However, this function can also called by the royalty contract owner who can provide recipient address of its convenience in the function arguments that can used calculating the `recipient` value at `contracts/royalty/src/contract.rs:126.` This would allow the contract owner to rug pull all the royalty funds of the NFT minters.

Recommendation

We recommend not allowing the contract owner to call the `update_royalty()` function to reduce the owner obligation or fetching therecipient address directly on the basis of the NFT token address.

Status: Resolved

2. Bidding contract owners can abuse their authority to front-run threshold values leading to unfair auctions.

Severity: Critical

Repo - `one-planet-auction-contract`
At - `contracts/bidding/src/contract.rs:82, 151`

In English auctions, bid acceptance is directly proportional to the bid amount. By changing the threshold value during the auction phase using the `update_config()` function, the owner can decrease the threshold value such that his bid becomes the highest and overwrites the current highest bid using the `place_bid()` function. Even if the owner changes the threshold value during the auction without having any intention to gain an advantage, this still leads to unfair auctions as other participants may try to outbid the current highest bid compared to the previous threshold value.

Recommendation

We recommend not allowing the contract owner to change the threshold value during a running auction. Alternatively, every auction may have a static threshold value which can't be changed throughout the auction's lifespan.

Status: Resolved

3. Contract owner could steal funds

Severity: Critical

Repo - `one-planet-contract-queuing`
At - `src/execute.rs:249`

The `refund()` function can only be called by the queuing contract owner and that will allow them to remove all the CW20 tokens it holds without sending the NFT to the appropriate owner who puts their order in the queue.

Recommendation

We recommend calling the `refund()` function when there is an empty queue.

Status: Resolved

4. Whitelisting in queuing contract can be bypassed by directly queuering the storage contract

Severity: Critical

Repo - `one-planet-contract-queuing`
At - `src/execute.rs:114`

The `queue()` function checks for the whitelisted address at each stage, and allows only the whitelisted addresses to claim tokens in `claim()`. The logic that checks the whitelisting queries the `info.sender` contract directly. The contract can respond to the query with any whitelisted address.

Recommendation

We recommend validating the `info.sender`

Status: Resolved

5. Change of storage contract address may lead to loss of funds.

Severity: Major

Repo - `one-planet-auction-contract`
At - `contracts/auction_house/src/storage.rs:80`

`execute_change_storage()` changes the storage address. This may lead to loss of funds if the storage address for the given maker address already exists and the new `code_id` doesn't have the appropriate functions to claim assets out of it.

Recommendation

We recommend checking if the storage contract already exists for the a maker address in the `reply()` function.

Status: Resolved

6. Cyclic dependencies are present in auction_house and royalty contracts that would break the initialization of the respective contracts.

Severity: Major

Repo - `one-planet-auction-contract`
At - `contracts/auction_house/src/contract.rs:37`, `contracts/royalty/src/contract.rs:28`

Instationation of the `auction_house` contract requires the address of the `royalty` contract whilst instantiation of the `royalty` contract requires the address of the `auction_house` contract. Because of this cyclic interdependency, it would not be possible to deploy both of the contracts with valid params.

Recommendation

We recommend making those params optional during the instantiation and then set them using the `update_config()` function.

Status: Resolved

7. The non-atomic nature of the `place_bid` function may lead to locked funds.

Severity: Minor

Repo - `one-planet-auction-contract`

At - `contracts/storage/src/order.rs:208,209`
`contracts/bidding/src/contract.rs:99`

To place a bid, a user needs to send funds to the storage contract first and then call the `place_bid()` function of the bidding contract , which triggers the `bidding_lock()` function in the storage contract to lock funds. This is a 2-step process which may lead to locking of funds if the second transaction fails. To retrieve those funds users have to explicitly call the `claim_assets()` funciton. This process may introduce some inconsistencies and non-atomic behaviour may cause more gas consumption.

Recommendation

We recommend making this process atomic, in order to provide a better user experience and avoid any inconsistencies.

Status: Resolved

8. Incorrect nonce management

Severity: Minor

Repo - `one-planet-auction-contract`
At - `contracts/auction_house/src/contract.rs:194`

Nonces can be passed as a parameter when calling the `approve_order()` function that makes no usage no nonce as it is controlled by the user itself. This will allow the replication of order and restrict adding as new order if the same nonce gets used.

Recommendation

We recommend to use a nonce which is managed by the contract itself (auto increment), that will always be unique for every new `approve_order()` call.

Status: Resolved

9. Inefficient update of the whitelist in queuing contract.

Severity: Minor

Repo - `one-planet-contract-queuing`
At - `src/execute.rs:74`

```
// save white list
```

```
WL.save(deps.storage, &stage_name, &white_list)?;
```

The above code does not append elements to the whitelist. Instead it will overwrite the whole array. This means that the operator has to provide the whole array each time an address is added.

Recommendation

We recommend having an appending interaction instead of overwriting to provide more flexibility or use `update` instead of `save` method.

Status: Resolved

10. Start time of the stage should be inclusive in the if statement.

Severity: Minor

Repo - `one-planet-contract-queuing`
At - `src/execute.rs:95`

```
// save white list
if env.block.time.nanos() / 1_000_000 > stage.info.start_time && env.block.time.nanos()
/ 1_000_000 < stage.info.end_time {
            current_stage = Some(stage);
}
```

In the above code, the stage start time is not inclusive. If the stage starts at `x` time then user can't put a queue order at `x` time, It only works at `x+1` which seems incorrect behaviour.

Recommendation

We recommend using >= for the start time instead of > .

Status: Resolved

11. Startime and endtime not validated during the instantiation of the queuing contract.

Severity: Minor

Repo - `one-planet-contract-queuing`
At - `src/execute.rs:95`

The current code does not validate the stages startTime and endTime during the initialization so technically there can be more than one stage that can be open at the same time.

Recommendation

We recommend validating the data during initialization of the contract to avoid those edge cases.

Status: Resolved

12. Contract version is missing

Severity: Informational

Repo - `one-planet-auction-contracts`
At - `contracts/auction_house/src/contract.rs, contracts/bidding/src/contract.rs, contracts/storage/src/contract.rs`

Contract versioning is missing in the contracts

Recommendation

We recommend having contract versioning for migrations.

Status: Resolved

13. Ambiguous naming convention

Severity: Informational

Repo - `one-planet-auction-contracts`
At - `contracts/royalty/src/contract.rs:93`

```
let interest_addr = deps.api.addr_validate(&contract_addr)?;
```

It seems confusing to call the collection contract address `interest_addr`. This reduces readability less and increases the chances of error in a future upgrade.

Recommendation

We recommend changing the naming convention to something that is easy to understand.

Status: Resolved

14. Unnecessary function arguments can be avoided.

Severity: Informational

Repo - `one-planet-auction-contracts`
At - `contracts/auction_house/src/contract.rs:182`

As per the code at `contracts/auction_house/src/contract.rs:201` `market_contract` should always be equal to the `config.market_contract`, otherwise the transaction will fail. This means users should not explicitly pass the `maker_contract` as a parameter. Instead, `config.market_contract` can be used by default.

Recommendation

We recommend removing the additional function argument.

Status: Resolved

15. Unnecessary asset ordering can be avoided.

Severity: Informational

Repo - `one-planet-auction-contracts`
At - `contracts/auction_house/src/contract.rs:222, 223`

```
    // If sequence of assets changed, order hash also changed
    let maker_assets = sort_assets(maker_assets)?;
    let taker_assets = sort_assets(taker_assets)?;
```

the above code, enforces sorted order for `maker_assets` and `taker_assets`. However, the order should not matter since all `maker_assets` are always be equal to `maker_asset_contract` (ref - `contracts/auction_house/src/validate.rs:67`) and similarly all `taker_assets` are equal to `taker_asset_contract` (ref - `contracts/auction_house/src/validate.rs:71`).

Recommendation

We recommend removing line #222 and #223 to save some gas.

Status: Resolved

16. Repetitive code can be avoided to minimise human error.

Severity: Informational

Repo - `one-planet-auction-contracts`
At - `contracts/storage/src/order.rs:87, contracts/storage/src/claim.rs:140, contracts/storage/src/claim.rs:217, contracts/storage/src/contract.rs:334`

Repo - `one-planet-contract-token`
At - `src/execute.rs:269`

Recommendation

We recommend reusing as much as code possible to avoid any human error during the update of the contract.

Status: Resolved

17. Validation of address should be done before storing it on-chain.

Severity: Informational

Repo - `one-planet-contract-queuing`
At - `src/execute.rs:35`

At above mentioned line the config value gets stored directly without validating the addresses. Best practice is to validate addresses before storing them on-chain.

Recommendation

We recommend validating the addresses before storing it on-chain.

Status: Resolved

18. Implement default values instead of random dummy values.

Severity: Informational

Repo - `one-planet-contract-queuing`
At - `src/query.rs:47,65`

At line `#47` `Stage` default value could be used instead of providing a random values. The same is true for line `#65` default values can be used where default values can be used for `NumResponse`.

Recommendation

We recommend using Rust language best practices where a given type can have default values.

Status: Acknowledged

19.Incorrect spelling of one of the contract error variants.

Severity: Informational

Repo - `one-planet-auction-contract`
At - `contracts/bidding/src/error.rs:21`

Typo mistake present in the spelling of ContractError variant i.e `ExceedBidDuraion`

Recommendation

We recommend fixing it with correct spelling i.e `ExceedBidDuration.`

Status: Resolved

20. Inefficient batch mint transaction.

Severity: Informational

Repo - `one-planet-contract-token`
At - `src/error.rs:238`

In the `execute_batch_mint()` function, Multiple NFTs can be minted but if the one of the given `token_id` out of multiple `token_ids` is already minted whole transaction would fail. This is an inefficient way of doing batch minting.

Recommendation

We recommend using optimistic minting where if one of the `token_id` exists then it should hop to another one till the end of all given `token_ids` and return the array of added `token_ids` in an event so that the frontend knows which token_ids get added and which are rejected.

Status: Resolved

21. Inefficient code which can lead up to unnecessary usage of gas.

Severity: Informational

Repo - `one-planet-contract-token`
At - `src/execute.rs:242`

`execute_batch_mint()` uses the `for` loop at line #228 to update the tokens and increment the count using the `increment_tokens()` function which updates `num_tokens` incrementally. This leads to changing storage multiple times and more gas consumption.

Recommendation

We recommend incrementing `num_tokens` and writing to storage once after the for loop to save a lot of gas.

Status: Resolved