SOLIDIFIED

Audit Report for SuperWorld Inc. - May 19, 2021

Summary

Audit Report prepared by Solidified covering the NFTSalon smart contract.

Process and Delivery

Three (3) independent Solidified experts performed an unbiased and isolated audit of the code below. The final debrief took place on April 23, 2021, and the results are presented here.

Some fixes were received on 7 May.

NOTE: The fixes supplied on 7 May only partially address the findings, further increase the complexity of the codebase, and introduce at least 3 new issues. Due to the low readability of the code and the limited testing, it is likely that further issues are present.

The audit team strongly advises against the deployment of the current codebase and recommends a redesign.

Audited Files

The source code has been supplied in a private source code repository:

https://github.com/kole-swapnil/CryptoArt/blob/master/contracts/NFTSalon.sol

Final commit number: `e55ed6029045f73c2759562990413e9f25876f2a`

Intended Behavior

The smart contract implements a non-fungible ERC-721 token, in addition to token batches that can be minted individually or in groups and can be sold directly or auctioned.

Findings

Smart contract 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 a smart contract 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**.

Note, that high complexity or lower 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.

Criteria	Status	Comment
Code complexity	High	Considering the functionality provided and the design chosen, the code complexity is excessively complex.
Code readability and clarity	Low	The design decision to include token collections, auctions, and sale functionality into a single ERC-721 contract decreases the readability of the codebase significantly.
Level of Documentation	Low	No documentation has been submitted. Source code commenting does not comply with best-practice guidelines (NatSpeec format) and is inconsistent.
Test Coverage	Low	No tests have been submitted to the audit and the number of issues found indicates that little or no testing has been performed.

SOLIDIFIED

Audit Report for SuperWorld Inc. - May 19, 2021

Solidified found that the NFTSalon contract contains 4 critical issues, 4 major issues, 3 minor issues, and two informational issues. Two end-user warnings have been noted.

We recommend issues are amended, while informational notes are up to the team's discretion, as they refer to best practices.

Issue #	Description	Severity	Status
1	Overall architecture may imply major security risks	Warning	-
2	No test coverage	Warning	-
2a	NEW ISSUE AFTER FIXES: Reentrancy in several places	Critical	Pending
2b	NEW ISSUE AFTER FIXES: Users cannot withdraw their funds	Critical	Pending
3	Anyone can mint tokens from any batch at an arbitrary or zero price	Critical	Resolved
4	Any bidder can block subsequent bids from being accepted	Critical	Partially Resolved
5	closeBid() does not close bidding	Critical	Resolved
6	Sale and Auction at the same time can cause bidder's funds to be locked forever	Critical	Pending
6a	NEW ISSUE AFTER FIXES: toString(uint256 _i) function always returns "0"	Major	Pending
7	Sellers can be left without any payment if fees are set high	Major	Partially Resolved
8	The contract owner can withdraw all funds at any time	Major	Resolved
9	Bids can be added even after auction end time	Major	Resolved
10	Token owner will lose funds for closing an auction with no bids	Major	Pending
11	Auction end time can be set in the past	Minor	Resolved

12	Duplicate over-/underflow protection	Minor	Resolved
13	Use call() instead of transfer()	Minor	Partially Resolved
14	Inefficient data-structures	Note	-
15	Include revert messages	Note	-

Warnings

1. Overall architecture may imply major security risks

The design of the system is based on an architecture that groups a large number of functionalities into a single smart contract. The decision to add token collection, auction, and sale functionality into the token smart contract, which also holds Ether value, increases the attack surface considerably. This design increases the likelihood of undetected issues.

Recommendation
Consider re-designing the protocol in a more modular fashion.

Team Reply
“Going ahead with the same structure”

2. No test coverage

The codebase lacks evidence of test coverage and the number of issues found indicates that testing has been very limited. This adds a risk of additional issues since there are a number of vulnerability types that are easy to detect in unit testing but harder to spot in a security audit.

Recommendation
Consider providing unit tests covering all functions and branches.

Team Reply
“Tested on our end from remix on most cases”

Critical Issues

2a. NEW ISSUE AFTER FIXES: Reentrancy in several places

The fixes supplied to issues 4 and 13 introduce reentrancy vulnerabilities in the codebase. For example:

The contract owner can take all ETH funds stored in the contract by using the `closeBid()` function.
The contract owner could bid for a token and then use re-entrancy to drain ETH funds from the contract.

```
(bool success, ) = (auctions[_tokenId].bidder).call{value:
auctions[_tokenId].bidPrice}("");
```

These reentrancy issues are due to state changes after external calls.

Recommendation
Use a reentrancy guard or avoid state changes after external calls

2b. NEW ISSUE AFTER FIXES: Users cannot withdraw their funds

The function `withdrawUserBalance()` has the following `require` statement

```
require(!success,"Transfer failed");
```

Thus, successful `msg.sender.call{value: userBalance[msg.sender]}("")` will always revert.

Recommendation
Enable users to withdraw funds.

3. Anyone can mint tokens from any batch at an arbitrary or zero price

The `mintToken()` function allows anyone to mint unclaimed tokens, even if the caller is not the batch creator. In addition, the payment for this operation is set by the caller and can be set as desired, even to zero. The fee deduction process indicates that minting should be payable operation, however, no minimum price is enforced.

Recommendation
Clarify the mint functionality and either enforce a minimum price or restrict this operation to the batch creator.

Team Reply
"When openCloseMint is true, the caller can do open minting when they enter a value greater or equal to the price set by the batch creator. If openCloseMint is false, caller cannot mint because they are not the creator. Also have a require to put sufficient money to openmint."

4. Any bidder can block subsequent bids from being accepted

A bidder can block subsequent bids from being accepted with a Denial Service attack, winning the auction by default. This is due to the `transfer()` call in `addBid()` that pushes refunds out to the previous highest bidder. The bidder can call the `addBid()` function from a smart contract. This contract can be implemented to revert when receiving the refund payment, causing the whole transaction to fail. This will block any other users from placing a higher bid.
In a similar manner, a bidder may block an auction to be closed. However, this is less critical, since there would not be an incentive to do so.

Recommendation
Do not push out refund payments. Instead, a pull pattern is recommended in which refunds have to be claimed in a separate transaction.

5. closeBid() does not close bidding

The `closeBid()` function never sets the `bidding` flag to `false`, meaning that auctions remain open. This allows anyone to create a new bid for that auction for as low as 1 wei and call the `closeBidBuyer()` method to claim the token.

Recommendation
Set `bidding` to `false`.

6. Sale and Auction at the same time can cause bidder's funds to be locked forever

A token on sale can be auctioned and vice-versa. This locks in the last bidder funds forever if an item on auction is sold.

Recommendation
Check the auction or sale status before initiating the other operation

Major Issues

6a. NEW ISSUE AFTER FIXES: toString(uint256 _i) function always returns "0"

```
if (_i == 0){
      }
      return "0";
```

The return statement is misplaced causing the function to always return 0

Recommendation
Place the return statement in the correct place.

7. Sellers can be left without any payment if fees are set high

There is nothing that prevents the total fees (royalties and operator fees) to be set to 100%, meaning sellers only receive any payment. This is even the case with the percentages used as examples in the comments in the code.

Recommendation
Limit fees.

Team Reply
"We did not want to limit the seller to any royalty fee but we will be showing the royalties percentage to the buyers. After a sale or bid happens, token creators cannot change the royalties."

8. The contract owner can withdraw all funds at any time

The contract owner can withdraw funds at any time, leaving bidders and sellers without any means of receiving their proceeds.

Recommendation
Limit the owner's privileges to reduce funds.

9. Bids can be added even after auction end time

The function `addBid()` does not check the auction end time and allows anyone to add a bid even after the set time.

Recommendation
Verify the bid end time.

10. Token owner will lose funds for closing an auction with no bids

In order for the owner to close a bid using the `closeBid()` method, it requires transferring the `bidPrice` to the bidder. In the case of an auction with no bidders, the owner can close the bid only if the specified `bidPrice` is transferred to zero address.

Recommendation
Avoid the transfer if no bidders are present for an auction.

Minor Issues

11. Auction end time can be set in the past

The `startBid()` function does not check the start timestamp when setting up an auction, allowing for any time to be used, including start times in the past. In such an auction, the first bidder could just immediately close the auction and win it. In addition, there is no bound on the maximum duration of an auction,

Recommendation

Enforce auction end times to be in the future and consider adding a maximum duration for an auction.

12. Duplicate over-/underflow protection

The code uses the `safeMath` library for the protection of arithmetic operations. However, it also enforces Solidity version 0.8.0 or higher, which already provides overflow protection. This causes unnecessary code execution, resulting in a higher gas cost.

Recommendation
Remove `safeMath` or disable built-in overflow protection.

13. Use call() instead of transfer()

The codebase uses the `transfer()` for ETH payments. However, since the Istanbul hard fork, the use of `transfer()` is discouraged, since the gas stipend forwarded may not be enough anymore for certain basic operations in smart contract wallets.

Recommendation
Consider using `call()` for ETH transfers. Additional reentrancy protection needs to be considered.

Informational Notes

14. Inefficient data-structures

The code uses mappings that map keys to hatches. However, the hashes could act themselves as keys, resulting in unnecessary storage and indirections.

Recommendation
Consider refactoring the data-structure design.

15. Include revert messages

It is recommended to add revert messages to all possible revert conditions for better usability.

Disclaimer

Solidified audit is not a security warranty, investment advice, or an endorsement of SuperWorld Inc. or its products. This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

The individual audit reports are anonymized and combined during a debrief process, in order to provide an unbiased delivery and protect the auditors of Solidified platform from legal and financial liability.

Solidified Technologies Inc.