CD SECURITY

AUDIT REPORT

Garage Sale
December 2023

Introduction

A time-boxed security review of the **Garage Sale** protocol was done by **CDSecurity**, with a focus on the security aspects of the application's implementation.

Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource, and expertise-bound effort where we try to find as many vulnerabilities as possible. We can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs, and on-chain monitoring are strongly recommended.

About **Garage Sale**

`GarageSale` is a single smart contract that allows users to sell their old NFTs and avoid paying taxes for them. Then it groups the tokens in bundles of 4 and sells them in an auction. Here is how it works:

The `controller` sets an `offer` which is a price that users who want to sell their ERC721 or ERC1155 tokens get paid by the contract once the tokens are transferred in it. The contract is funded by calling `fund()` with some `msg.value`. All the tokens that are sold to the contract are stored in the `inventory` array. At consistent 15-minute intervals, GarageSale conducts a Dutch auction. This auction model progressively decreases the price until the auction's conclusion. Each auction curates bundles containing 4 different tokens. Users possess the privilege to review the ERC721/ERC1155 tokens scheduled for sale in an upcoming auction via the `preview()` function.Then if he is interested in buying, he just calls `buy()`, which calculates the price that has to be paid based on the time elapsed from 15 min till the moment of calling the function, the user pays the price and gets the bundle of tokens.

Threat Model

Privileged Roles & Actors

- `Owner` - responsible for setting the `controller` and able to `withdraw` the native tokens.
- `Controller` - a role that can set the parameters for an auction, update the offer and the tokens.
- `Users` - the normal users that can sell and buy tokens through the contract.

Security Interview

Q: What in the protocol has value in the market?

A: The ERC721/ERC1155 tokens and the native tokens stored in the contract.

Q: In what case can the protocol/users lose money?

A: If the balance of the contract is drained or the NFTs are lost.

Q: What are some ways that an attacker achieves his goals?

A: To gain ownership over the contract and `withdraw` all of the funds.

Severity classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Impact - the technical, economic and reputation damage of a successful attack

Likelihood - the chance that a particular vulnerability gets discovered and exploited

Severity - the overall criticality of the risk

Security Assessment Summary

***review commit hash* - 92cf4e90ef392251c0813499d40c08f1697263f2**

Scope

The following smart contracts were in scope of the audit:

- `GarageSale.sol`

The following number of issues were found, categorized by their severity:

- Critical & High: 1 issues
- Medium: 1 issues
- Low: 1 issues

Findings Summary

ID	Title	Severity
[H-01]	An off-by-one error leads to stuck funds and unexpected errors	High
[M-01]	Insufficient input validation	Medium
[L-01]	Single-step ownership transfer pattern is dangerous	Low
[I-01]	Missing event emission	Informational
[I-02]	Redundant import	Informational

ID	Title	Severity
[I-03]	Prefer Solidity Custom Errors over `require` statements with strings	Informational
[I-04]	Using the latest pragma version is problematic for some chains	Informational

Detailed Findings

[H-01] An off-by-one error leads to stuck funds and unexpected errors

Severity

Impact: High because a funds will be stuck in the contract

Likelihood: Medium because only a small amount will be left

Description

The use of a wrong comparison operator will make it impossible to `withdraw` the whole balance of the contract:

```
    function withdraw(uint256 amount) external onlyOwner {
        require(amount > 0, "withdraw amount is zero");
        require(amount < address(this).balance, "insufficient balance");
        emit Withdrawn(amount);
        (bool sent, ) = payable(msg.sender).call{value: amount}("");
        require(sent, "ether withdraw failed");
    }
```

As it can be seen from the second `require` statement, the `amount` should be strictly less than `address(this).balance`. This will lead to leaving funds in the contract forever as there is no other way to get the native tokens out of the contract.

The same issue is present in the `onERC721Received`, `onERC1155Received` and `onERC1155BatchReceived` functions where it is required the balance of the contract to be strictly more than the `offer`:

```
require(address(this).balance > offer_, "insufficient funds");
```

This means that even if there is enough tokens in the contracts to buy the users' tokens, the call will revert with an error message. This could be quite frustrating for users. For example, if the `offer` is `0.0001 ether` and there is the exact amount, the call will revert.

Recommendations

Change the above instances to `>=` and `<=`.

Client

Fixed

[M-01] Insufficient input validation

Severity

Impact: Medium, because a protocol can be broken and the code could give a false calculations

Likelihood: Medium, as it can be gamed but it needs compromised / malicious owner

Description

In `setAuction()` we have couple of params that require a proper validation in case of a error from the side of the owner or a malicious/compromised one. You have perfectly validated the input of the `min` param but the same is lacking for `max` & `duration` which can be problematic in some cases.

Recommendation

Write reasonable checks for those two params in order to avoid further issues.

Client

Fixed

[L-01] Single-step ownership transfer pattern is dangerous

We can see that the contract is inheriting from OpenZeppelin's `Ownable` contract which means a single-step ownership transfer pattern is used.

If the `owner` provides an incorrect address for the new owner this will result in none of the `onlyOwner` marked methods being callable again.

The better way to do this is to use a two-step ownership transfer approach, where the new owner should first claim its new rights before they are transferred. Use OpenZeppelin's `Ownable2Step` instead of `Ownable`.

Client

Fixed

[I-01] Missing event emission

It's a best practice to emit events on every state changing method for off-chain monitoring. This is missing in `bump` function.

Client

Fixed

[I-02] Redundant import

The contract is importing the `IERC165`. However, this is unnecessary as the `IERC721` and `IERC1155` are inheriting it. Consider removing it to make the got more optimized.

Client

Acknowledged. Leaving as is for cleaner code pattern (can use IERC165 api for either token type). Not as concerned with gas cost on admin functions.

[I-03] Prefer Solidity Custom Errors over `require` statements with strings

Using Solidity Custom Errors has the benefits of less gas spent in reverted transactions, better interoperability of the protocol as clients of it can catch the errors easily on-chain, as well as you can give descriptive names of the errors without having a bigger bytecode or transaction gas spending, which will result in a better UX as well. Consider replacing the `require` statements with custom errors.

Client

Acknowledged. Leaving as is for simplicity.

[I-04] Using the latest pragma version is problematic for some chains

The contract is using a stable `0.8.23` version which will lead to problems if the contract is deployed on some L2s. For example, the `PUSH0` opcode which is introduced in the `0.8.20` version is not yet supported on Arbitrum - check here. The same applies for Optimism. Consider changing it the version to `0.8.19` as it is battle-tested on all chains.

Client

Fixed