Cho Saigon Market Audit Conclusion

by Patrizio Stavola

30th September 2022

I found no critical bugs, but have discovered some highly impacting issues that disrupt smart contract logic and functionality.

Findings

ID	Severity	Subject
CV-1	High	Storage memory not updated
CV-2	High	Storage memory not updated
CV-3	High	Modifier condition never met
CV-4	Medium	Unused State Variable
CV-5	Low	State Variables declaration on top
CV-6	Low	State Variables name start with _ (underscore)

Contents

1 Introduction

The audit goal is a general review of the smart contract structure, critical/major bugs detection and issuing the general recommendations.

A project outline was provided, but there was no mention of any working UI.

Mocha javascript unit test cases were provided, of which 4 out of 12 were failing because of smart contract issues that are analyzed in this document. After fixing the smart contract issues 2 test cases are still failing because of errors in javascript test files. These last 2 errors are not part of this analysis.

For this analysis the following files were audited:

- contracts/SaigonMarket.sol;
- contracts/SaigonNftFactory.sol;

1.1 Methodology

The methodology is not a strict formal procedure, but rather a collection of methods and tactics. In current audit I use:

- **General Code Assessment**. The code is reviewed for clarity, consistency, style, and for whether it follows code best practices applicable to the particular programming language used. I check indentation, naming convention, commented code blocks, code duplication, confusing names, confusing, irrelevant, or missing comments etc. At this phase we also understand overall code structure.
- **Code Logic Analysis**. The code logic of particular functions is analysed for correctness and efficiency. I check that code actually does what it is supposed to do, that algorithms are optimal and correct, and that proper data types are used. At this phase I also understand data structures used and the purposes they are used for.

2 Detailed Results

2.1 CV-1 Storage memory not updated

- **Severity** High
- **Source** SaigonMarket.sol

Description

createMarketSale function updates current market sale details into temporary memory location instead of storage memory location.

```
321    Listing memory listing = listings[_nftAddress][_listingId];
```

Impact

Market listing results as still open while the item has already been sold and transferred. This leads to unwanted behavior since smart contract state is never updated and then market sales are never stored.

Recommendation

Instantiate variable by using “storage” keyword to update smart contract state.

```
321    Listing storage listing = listings[_nftAddress][_listingId];
```

2.2 CV-2 Storage memory not updated

- **Severity** High
- **Source** SaigonMarket.sol

Description

resellNFT function updates listing details into temporary memory location instead of storage memory location.

```
290    Listing memory listing = listings[_nftAddress][_listingId];
```

Impact

Market listing is never updated with the new price. This leads to unwanted behavior since smart contract state is never updated and then new listing detail is never stored.

Recommendation

Instantiate variable by using “storage” keyword to update smart contract state.

```
290    Listing storage listing = listings[_nftAddress][_listingId];
```

2.3 CV-3 Modifier condition never met

- **Severity** High
- **Source** SaigonMarket.sol

Description

Inside logic of modifier ‘isListed’ the ‘&&’ condition is impossible to be met since input uint value will never be <= 0 AND > counter current value at the same time (counter starts at one).

```
147     if (listingId <= 0 && listingId > _listingIds.current()) {
```

Impact

Invoking createMarketSale for a not listed NFT will never be reverted by this modifier, it will actually trigger a function call to a not-existing contract address hence an EVM error.

Recommendation

Use || operator to trigger revert in both “<= 0” and “> counter” cases. Additionally, as this modifier is used only in one function it can be considered to move this check inside function body.

```
147     if (listingId <= 0 || listingId > _listingIds.current()) {
```

2.4 CV-4 Unused State Variable

- **Severity** Medium
- **Source** SaigonMarket.sol

Description

State variables has been correctly defined but never used.

```
172     bytes4 private constant INTERFACE_ID_ERC1155 = 0xd9b67a26;
```

Impact

Contract state bloating and increased gas usage for contract calls.

Recommendation

Unused state variables should be deleted or commented out.

```
172     //bytes4 private constant INTERFACE_ID_ERC1155 = 0xd9b67a26;
```

2.5 CV-5 State Variables declarations on top

- **Severity** Low
- **Source** SaigonMarket.sol

Description

According to common design patterns you should have state variables declared on top, right before events declarations.

```
167 /**********************/
168 /*** State Variables ***/
169 /**********************/
```

Impact

No tangible impacts but following best practices improve clarity and security.

Recommendation

Move declarations on top.

```
42 /**********************/
43 /*** State Variables ***/
44 /**********************/
```

2.6 CV-6 State Variables names start with _ (underscore)

- **Severity** Low
- **Source** SaigonMarket.sol

Description

According to common design patterns, variables with initial letter _ are used for function parameters and not state variables, which follow the usual mixed case convention.

```
39 Counters.Counter public _listingIds;
40 Counters.Counter private _listingsSold;
```

Impact

No tangible impacts but following best practices improve clarity and security.

Recommendation

Remove _

```
39 Counters.Counter public listingIds;
40 Counters.Counter private listingsSold;
```