Contract audit report

0. Preface

For making code repo more readable and easier to understand we separated provided code files with all code in one to individual files with imports. No code was changed during this process.

1. Critical issues

1.1 Contract “ERC1155Sale”

1.1.1 Function “Buy”

Require statements:

```
require(total + buyerFeeValue == msg.value, "msg.value is incorrect");
```

&

```
require(msg.value == price + buyerFeeValue, "msg.value is incorrect");
```

If somebody manages to put such “price” to make the next equation:
`price(or total) + buyerFeeValue == 2^256`.

Then due to overflow, it becomes zero which makes it possible to pass the requirement with

```
msg.value == 0
```

1.1.2 Solution

Recommended: Add “SafeMath” library to those “require” to make them fail in case of overflow.

The fixed code would look next:

```
require(total.add(buyerFeeValue) == msg.value, "msg.value is incorrect");
```

```
require(msg.value == price.add(buyerFeeValue), "msg.value is incorrect");
```

1.2 Contract “ERC721Sale”

1.2.1 Function “Buy”

Require statements:

```
require(total + buyerFeeValue == msg.value, "msg.value is incorrect");
```

&

```
require(msg.value == price + buyerFeeValue, "msg.value is incorrect");
```

If somebody manages to put such “price” to make the next equation:
`price(or total) + buyerFeeValue == 2^256`.

Then due to overflow, it becomes zero which makes it possible to pass the requirement with

```
msg.value == 0
```

1.2.2 Solution

Recommended: Add “SafeMath” library to those “require” to make them fail in case of overflow.

The fixed code would look next:

```
require(total.add(buyerFeeValue) == msg.value, "msg.value is incorrect");
```

```
require(msg.value == price.add(buyerFeeValue), "msg.value is incorrect");
```

2. Non critical issues

2.1 Contract “./ERC1155/ERC1155UniqueOneToken.sol”

2.1.1 Function “mint”

Contract ERC1155UniqueOneToken does not have “onlyOwner” restriction on “mint” function. It can be a threat if contract “ERC1155UniqueOneToken” is deployed itself and not using “ERC1155UniqueOneUserToken”, in this case, anybody will be able to use “mint” functionality without owner access restrictions.

2.1.2 Solution

Recommended:

1. Change visibility modifier of “mint” function of “ERC1155UniqueOneToken” to internal, it will make it possible to call it only from contract “ERC1155UniqueOneUserToken”.

2. Change constructor visibility of “ERC1155UniqueOneToken” to “internal” to make it impossible to be deployed separately from “ERC1155UniqueOneUserToken”.

2.2 Contract “./ERC1155/ERC1155UniqueOneUserToken.sol”

2.1.1 Function “mint”

Contract ERC1155UniqueOneToken does not have “onlyOwner” restriction on “mint” function. It can be a threat if contract “ERC1155UniqueOneToken” is deployed itself and not using “ERC1155UniqueOneUserToken”, in this case, anybody will be able to use “mint” functionality without owner access restrictions.

2.1.2 Solution

Recommended:

1. Change visibility modifier of “mint” function of “ERC1155UniqueOneToken” to internal, it will make it possible to call it only from contract “ERC1155UniqueOneUserToken”.

2. Change constructor visibility of “ERC1155UniqueOneToken” to “internal” to make it impossible to be deployed separately from “ERC1155UniqueOneUserToken”.

2.3 Contract “./ERC20/RareToken.sol”

2.3.1 Function “setGovernance”

Due to the non-existence of any “require” statements “governance” can be passed to any address including address(0) and “dead” address.

2.3.2 Solution

Recommended:

1. add require statement to make sure it is impossible to transfer to address(0) or/and add claiming functionality for new “governance” address.

2.4 Contract “AbstractSale”

2.4.1 Function “setBuyerFee”

We suppose that the “buyerFee” variable should be not more than 10000 because it represents the percentage as we can understand from functions “buy”, where parameter “buyerFee” is multiplied to some value and divided to 10000. That means for a 100% fee we need to assign a “buyerFee” value of 10000. From this point, if the owner makes a mistake and puts more than 10000 fee amount and total amount in “buy” functions can become higher than expected.

2.4.2 Solution

Recommended:

1. In the function “setBuyerFee” add a required statement that will not pass the input value more than 10000.

3 Additional recommendations

1. Contract “AbstractSale” function “strConcat” the same as from string library, should be replaced with the library using.
2. Contract “AbstractSale” and “ERC1155Sale”. All over the code magical constant “10000” is used, should be placed to a constant variable with the corresponding name to make more understanding of what it is and making it is harder to mistake in case of changing this number.
3. Fix inheritance tree:
    a. Contract “ERC721Base” is inherited from “ERC721” and from “ERC721Enumerable”. But “ERC721Enumerable” is already inherited from “ERC721”, meaning that “ERC721Base” should be inherited from “ERC721Enumerable” only.
    b. Contract “ERC1155Base” is inherited from “Ownable”, “ERC1155UniqueOneToken” is inherited from “ERC1155Base” and along with it from “Ownable” too, no need to repeat inheritance. (Same for “AbstractSale” and two sale contracts, “ERC1155Sale”, “ERC721Sale).

3.1 Code style recommendations

1. Contract “AbstractSale”, line 20, no need to assign just defined variable to 0 it is by default.
2. Code style is broken in lots of places in the contracts, it is recommended to use code style form https://docs.soliditylang.org/en/v0.8.0/style-guide.html.