ditCraft

Report

Smart Contract Audit

Webaverse

15th of December 2020

1. Preface **2**

2. Manual Code Review **3**

2.1. Bugs & Vulnerabilities 3

2.2. Other Findings 4

3. Protocol/Logic Review **6**

4. ERC20/ERC721 Standard **8**

5. Summary **9**

1. Preface

The team of **Webaverse** contracted ditCraft to conduct a software audit of their developed smart contracts written in Solidity. They are (*among other things*) developing smart contracts to issue tokens using a Discord bot.
Their smart contracts have been reviewed by us in the course of this audit.

The following services to be provided by ditCraft were defined:

- Manual code review
- Protocol/Logic review and analysis
- Verification whether the contracts conform to the ERC20/721 standard
- A written summary of all the findings and suggestions on how to remedy them as well as the outcome of the comparison

ditCraft gained access to the code via their public GitHub repository. We based the audit on the master branch's state on December 7th 2020 (*commit hash 52da2e02689e17c5666fc28f839f0b4122cc1792*).

2. Manual Code Review

ditCraft conducted a manual code review of all the smart contracts that are contained in the repository.

The code of these contracts has been written according to the latest standards used within the Ethereum community and best practices of the Solidity community. The naming of variables is logical and comprehensible, which results in the contract being good to understand. Most of the code is well documented within the code itself.

The general documentation within the repository and the contracts could be improved.

The repository contents however were not contained in a truffle or hardhat project and also didn't make use of npm/yarn-based imports of OpenZeppelin contracts, which we would advise to do. For our audit we integrated the corresponding contracts into a local truffle project in order to also test it out in our testnet environment.

On the code level, we **found no 3 medium severity bugs or flaws**. Additionally, we did find 3 of no severity. An additional double-check with two automated reviewing tools (one of them being the highest-paid version of MythX) **did not find any additional bugs**.

2.1. Bugs & Vulnerabilities

A) WebaverseERC20Proxy.sol Line 9-15 **[NO SEVERITY]**
There is no visibility specified. While this is not a problem per se, as the default is `internal`, we suggest defining it explicitly.

B) WebaverseERC20Proxy.sol Line 47-51 **[MEDIUM SEVERITY]**

```
uint256 balanceNeeded = amount - deposits;
if (needsMint) {
    deposits += balanceNeeded;
}
deposits -= amount;
```

While SafeMath.sol is part of the repository, it is not being used here. Even if there would be no theoretical way to exploit this, we always advise to make use of SafeMath to prevent exploits.

C) WebaverseERC20Proxy.sol Line 63 **[MEDIUM SEVERITY]**

```
  deposits += amount;
```

While SafeMath.sol is part of the repository, it is not being used here. Even if there would be no theoretical way to exploit this, we always advise to make use of SafeMath to prevent exploits, especially in this case.

D) WebaverseERC721Proxy.sol Line 10-16 **[NO SEVERITY]**
There is no visibility specified. While this is not a problem per se, as the default is `internal`, we suggest defining it explicitly.

E) WebaverseERC721.sol Line 16-28 **[NO SEVERITY]**
There is no visibility specified for some variables. While this is not a problem per se, as the default is `internal`, we suggest defining it explicitly.

F) WebaverseERC721.sol Line 68 **[MEDIUM SEVERITY]**

```
tokenIdToBalance[tokenId] -= amount;
```

While SafeMath.sol is part of the repository, it is not being used here. Even if there would be no theoretical way to exploit this, we always advise to make use of SafeMath to prevent exploits.

2.2. Other Findings

A) WebaverseERC20.sol Line 21

```
require(isAllowedMinter(msg.sender));
```

As this code piece is used three times throughout the contract, it can be turned into a `modifier`.

B) WebaverseERC20Proxy.sol Line 56-57

```
address contractAddress = address(this);
parent.mint(contractAddress, balanceNeeded);
```

As `contractAddress` is not being used again, this can be simplified (*and save a bit of gas*) to:

```
parent.mint(address(this), balanceNeeded);
```

C) WebaverseERC20Proxy.sol Line 67-69

```
address from = msg.sender;
address contractAddress = address(this);
require(parent.transferFrom(from, contractAddress, amount), "transfer
failed");
```

As `from` and `contractAddress` are not being used again, this can be simplified (*and save a bit of gas*) to:

```
require(parent.transferFrom(msg.sender, address(this), amount),
"transfer failed");
```

D) WebaverseERC721Proxy.sol Line 62-64

```
address from = msg.sender;
address contractAddress = address(this);
require(parent.transferFrom(from, contractAddress, amount), "transfer
failed");
```

As `from` and `contractAddress` are not being used again, this can be simplified (*and save a bit of gas*) to:

```
require(parent.transferFrom(msg.sender, address(this), amount),
"transfer failed");
```

E) General Remark
You implemented a lot of `public` functions that are not called from within the contract. The more suitable and efficient keyword would be `external` in this case, since this only allows external calls.

F) WebaverseERC721.sol Line 61-62

```
address contractAddress = address(this);
require(erc20Contract.transferFrom(from, contractAddress, amount),
"transfer failed");
```

As `from` and `contractAddress` are not being used again, this can be simplified (*and save a bit of gas*) to:

```
require(erc20Contract.transferFrom(from, address(this), amount),
"transfer failed");
```

G) WebaverseERC721.sol Line 259-267

```
struct Token {
   uint256 id;
   uint256 hash;
   string filename;
   address minter;
   address owner;
   uint256 balance;
   uint256 totalSupply;
}
```

It is generally advised to declare variables and structs at the top of the contract.

3. Protocol/Logic Review

On the protocol level we didn’t find any alarming problems. The access control system could be a bit more granular, but this might be intended. A possible problem to lock everyone out of the contract could occur, since all minters are allowed to add and remove all of the other minters and no contract owner address is defined - thus a malicious minter could brick the contracts. Hence, we advise to be very careful with who you give these rights to. Looking into a more fine granular access system would be advisable if needed.

When deployed to our testnet and during our practical tests, we couldn’t find any additional issues.

4. ERC20/ERC721 Standard

The contracts are importing the standard ERC20 and ERC721 libraries. There are no changes being made to the predefined functions that are part of the corresponding specifications. During our tests, we were able to interact with the ERC20 token and the ERC721 tokens fine with regular wallets implementing these standards.

We didn’t find any issues corresponding to the ERC20 or ERC721 standard in the contracts.

5. Summary

During our code review (which was done manual as well as automated) we **found 3 medium severity bugs or flaws**. Additionally, we did find three of no severity.

While ultimately all of our findings are recommendations, we would like to see the medium severity ones to be addressed by the Webaverse team.

In general, we are very happy with the overall quality of the code as well as the tests that have been written. The provided documentation and the in-line code-comments could have been improved, but this is ultimately up to the Webaverse team.