RED4SEC

Smart Contract Security Audit

Fairox

2021-02-22

Content

Introduction

Fairox iGaming Group operates a Global Online Gaming Union Membership Project that leverages Blockchain technology. This interlinks and integrates various international iGaming platforms seamlessly into its gaming ecosystem, allowing members to participate and enjoy their gaming journey from every part of the world at most secured manner.

As requested by **Fairox** and as part of the vulnerability review and management process, **Red4Sec** has been asked to perform a security code audit to **evaluate the security of** its Fairox Membership Coin smart contract.

Disclaimer

This document only represents the results of the code audit conducted by Red4Sec Cybersecurity and should not be used in any way to make investment decisions or as investment advice on a project.

Likewise, the report should not be considered neither "endorsement" nor "disapproval" of the guarantee of the correct business model of the analyzed project.

Scope

Fairox Membership Coin Smart Contract

- FairoxMembershipCoin.sol
  - SHA256: BB962E389378EF3787EFEEEE981A2086B5A49F39835A8D748434678EC7463EC8
- Remediations review
  - SHA256: 90D452E990E854F66B4ABB6ABF3DC4F20D62EC8825B1924D73FF3FD5EA7CB717

Results Overview

To this date, 22th of February 2021, the general conclusion resulting from the conducted audit is that **Fairox's smart contract is secure and does not present any known vulnerabilities** that could compromise the security of the users.

Nevertheless, Red4Sec has found some minor potential improvements, these do not pose any risk and we have classified such issues as informative only, but they will help **Fairox** to continue to improve the security and quality of its developments.

- All **issues found in the audit has been corrected** by the Fairox team and subsequently reviewed and verified by the Red4Sec team.
- A **few low impact issues** were detected and classified only as informative, but they will continue to help Fairox improve the security and quality of its developments.
- **Fairox's team has corrected the vulnerabilities detailed in this report** and the current status of the contract, after the review made by the Red4Sec team, is as follows:

Table of vulnerabilities

Id.	Vulnerability	State
FX01	Failure in the logic of burning tokens	Fixed
FX02	Improvable mining logic	Fixed
FX03	Unsecure Ownership Transfer	Fixed
FX04	Wrong timestamp variable visibility	Fixed

Recommendations

FX01 - Failure in the logic of burning tokens

The logics implemented in the contract to burn and mine tokens overlap their effects, causing the burning made by the users to be only temporary, consequently, the burned supply can be recovered under certain circumstances.

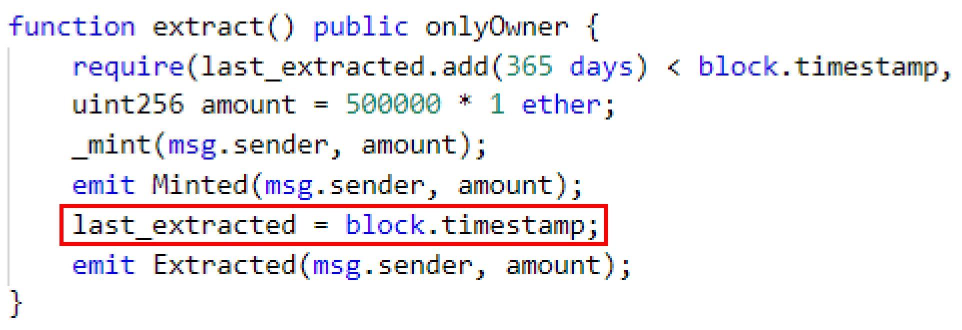
FairoxMembershipCoin establishes a maximum supply of 25000000 * 1 ether, so when a user burns tokens it waits for them to be reduced from the maximum supply and they cannot be mined again.

The problem is given because the *extract* administrative function, to mine the 2% per year that is pending, is only limited by the capped supply with the intention of being invoked in the following 10 years, so that after 10 years, tokens could continue to be mined for the owner that were burned by users, as long as they are more than 500000 * 1 ether.

FX02 - Improvable mining logic

The *last_extracted* variable is not initialized in the constructor, this enables the owner to use the *extract* function, exclusively to make the contract's deployment, since this variable will contain a 0 and will always carry out the 365 days condition.

It is also worth mentioning that the *extract* function sets the deadlines from the last time that the method was called to (not necessarily 1 year) so if the owner takes 1 month to call it, due to technical or logistics problems, this month will accumulate for future calls, being able to delay the process for more than 10 years, which was the initial estimation.



```
function extract() public onlyOwner {
    require(last_extracted.add(365 days) < block.timestamp,
    uint256 amount = 500000 * 1 ether;
    _mint(msg.sender, amount);
    emit Minted(msg.sender, amount);
    last_extracted = block.timestamp;
    emit Extracted(msg.sender, amount);
}
```

It is recommended to add 365 days to the *last_extracted* variable instead of taking the date of the current block.

FX03 - Unsecure Ownership Transfer

The modification process of an owner is a delicate process, since the governance of our contract and therefore of the project may be at risk, for this reason it is recommended to adjust the owner’s modification logic, to a logic that allows to verify that the new owner is in fact valid and does exist.

Following, we can see a standard logic of the owner’s modification where a new owner is proposed first, the owner accepts the proposal and, in this way, we make sure that there are no errors when writing the address of the new owner.

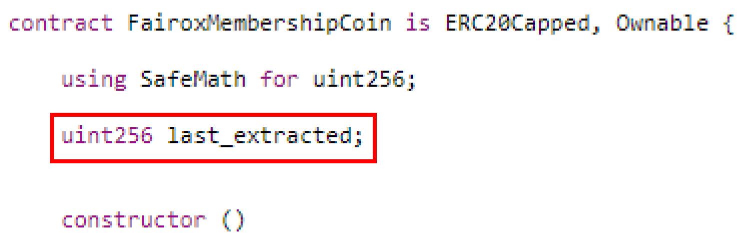
```
function proposeOwner(address _proposedOwner) public onlyOwner
{
        require(msg.sender != _proposedOwner, ERROR_CALLER_ALREADY_OWNER);
        proposedOwner = _proposedOwner;
}

function claimOwnership() public
{
        require(msg.sender == proposedOwner, ERROR_NOT_PROPOSED_OWNER);
        emit OwnershipTransferred(_owner, proposedOwner);
        _owner = proposedOwner;
        proposedOwner = address(0);
}
```

Source reference

- Owable.sol

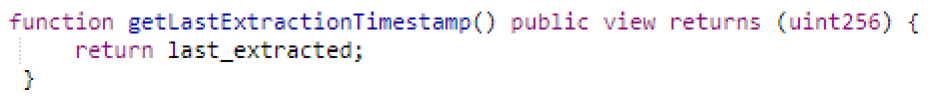
FX04 - Wrong timestamp variable visibility

In order to simplify the contract for the users, it is recommended to turn the *last_extracted* variable to public.



```
contract FairoxMembershipCoin is ERC20Capped, Ownable {

    using SafeMath for uint256;

    uint256 last_extracted;


    constructor ()
```

When initializing the *last_extracted* variable as public the *getLastExtractionTimestamp* method will no longer be necessary, and can be removed:



```
function getLastExtractionTimestamp() public view returns (uint256) {
    return last_extracted;
}
```

RED4SEC

Invest in Security, invest in your future