SALUS SECURITY

JULY 2023

CODE SECURITY ASSESSMENT

SIRIUS

Overview

Project Summary

- Name: Sirius - NFT
- Platform: EVM-compatible Chains
- Language: Solidity
- Audit Scope: See Appendix - 1

Project Dashboard

Application Summary

Name	Sirius - NFT
Version	1
Type	Solidity
Dates	July 25 2023
Logs	July 25 2023

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	0
Total Low-Severity issues	1
Total informational issues	10
Total	11

Contact

E-mail: support@salusec.io

SALUS

Risk Level Description

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for clients' reputations or serious financial implications for clients and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to a moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or defense in depth.

SALUS

Content

SALUS

Introduction

1.1 About SALUS

At Salus Security, we are in the business of trust.

We are dedicated to tackling the toughest security challenges facing the industry today. By building foundational trust in technology and infrastructure through security, we help clients to lead their respective industries and unlock their full Web3 potential.

Our team of security experts employ industry-leading proof-of-concept (PoC) methodology for demonstrating smart contract vulnerabilities, coupled with advanced red teaming capabilities and a stereoscopic vulnerability detection service, to deliver comprehensive security assessments that allow clients to stay ahead of the curve.

In addition to smart contract audits and red teaming, our Rapid Detection Service for smart contracts aims to make security accessible to all. This high calibre, yet cost-efficient, security tool has been designed to support a wide range of business needs including investment due diligence, security and code quality assessments, and code optimisation.

We are reachable on Telegram (https://t.me/salusec), Twitter (https://twitter.com/salus_sec), or Email (support@salusec.io).

1.2 Audit Breakdown

The objective was to evaluate the repository for security-related issues, code quality, and adherence to specifications and best practices. Possible issues we looked for included (but are not limited to):

- Risky external calls
- Integer overflow/underflow
- Transaction-ordering dependence
- Timestamp dependence
- Access control
- Call stack limits and mishandled exceptions
- Number rounding errors
- Centralization of power
- Logical oversights and denial of service
- Business logic specification
- Code clones, functionality duplication

1.3 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release and does not give any warranties on finding all possible security issues with the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues.

SALUS

Findings

2.1 Summary of Findings

ID	Title	Severity	Category	Status
1	Missing events for important parameter change	Low	Logging	Pending
2	Can not delete the IPFS with index 1	Informational	Business Logic	Pending
3	Implement code reuse	Informational	Code Quality	Pending
4	Unspecified Parent Contract in Override Function	Informational	Code Quality	Pending
5	Update the test information of the contract	Informational	Code Quality	Pending
6	Use of floating pragma	Informational	Configuration	Pending
7	Lacking of importing contract	Informational	Code Integrity	Pending
8	Missing zero address check	Informational	Data Validation	Pending
9	Gas Optimizations	Informational	Gas Optimization	Pending
10	Spelling error	Informational	Code Quality	Pending
11	Dead code	Informational	Code Quality	Pending

SALUS

2.2 Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

1. Missing events for important parameter change

Severity: Informational	Category: Logging

Target:
- Sirius.sol

Description

Important parameter or configuration changes should trigger an event to enable tracking off-chain, but functions mentioned below changing the important parameters do not emit events.

Sirius.sol:L32-L37

```
function modifyIPFS(uint256 _num, string memory _ipfs) onlyOwner public returns (bool) {
    //...
    IPFS_URI[_num]=_ipfs;
    return true;
}
```

Sirius.sol:L38-L44

```
function add_list_IPFS(string[] memory _ipfs) onlyOwner public returns (bool) {
    //...
        IPFS_URI[count_ipfs]=_ipfs[i];
    //...
}
```

Sirius.sol:L46-L50

```
function addIPFS(string memory _ipfs) onlyOwner public returns (bool) {
    //...
    IPFS_URI[count_ipfs]=_ipfs;
    return true;
}
```

Sirius.sol:L52-L59

```
function lowerCountIPFS(uint256 _count) onlyOwner public returns (bool) {
    //...
    for(uint i=_count+1;i<=count_ipfs;i++){
        IPFS_URI[i]="";
    }
    count_ipfs=_count;
    return true;
}
```

Sirius.sol:L77-L82

```
function add_whitelist(address[] memory listAdress) onlyOwner public returns (bool) {
    //...
    whitelist[listAdress[i]]=true;
    //...
}
```

Sirius.sol:L84-L89

SALUS

```
function remove_whitelist(address[] memory listAdress) onlyOwner public returns (bool) {
    //...
    whitelist[listAdress[i]]=false;
    //...
}
```

Recommendation

Design proper events and add them to the aforementioned functions.

SALUS

2.3 Informational Findings

2. Can not delete the IPFS with index 1

Severity: Informational	Category: Business Logic
Target: - Sirius.sol	

Description

The lowerCountIPFS() function can not reduce the count_ipfs to its initial value (0) because _count is not allowed to be zero. And it means the function can not delete the IPFS[1]. It can only be deleted from the IPFS[2].

Sirius.sol:L53 - L56

```
function lowerCountIPFS(uint256 _count) onlyOwner public returns (bool) {
        require(_count<count_ipfs && _count>0, 'number invalid' );
        for(uint i=_count+1;i<=count_ipfs;i++){
            IPFS_URI[i]="";
        }
        count_ipfs=_count;
        return true;
    }
```

Recommendation

Consider removing _count>0 check.

SALUS

3. Implement code reuse

Severity: Informational	Category: Code Quality
Target: - Sirius.sol	

Description

The `add_list_IPFS` function means adding multiple IPFS, while the `addIPFS` function means adding a single IPFS, there is an inherent relationship between the logic of these two functions, with “addList” calling "addOne" to accomplish the task of adding a single IPFS, and increasing code reuse and facilitating code modification.

Sirius.sol:L40-L41

```
function add_list_IPFS(string[] memory _ipfs) onlyOwner public returns (bool) {
        for(uint i=0;i<_ipfs.length;i++){
            count_ipfs++;
            IPFS_URI[count_ipfs]=_ipfs[i];
        }
        return true;
}

function addIPFS(string memory _ipfs) onlyOwner public returns (bool) {
    count_ipfs++;
    IPFS_URI[count_ipfs]=_ipfs;
    return true;
}
```

Recommendation

Consider invoking the addIPFS function within the `add_list_IPFS` function to facilitate code optimization and code reuse.

SALUS

4. Unspecified Parent Contract in Override Function

Severity: Informational	Category: Code Quality
Target: - Sirius.sol	

Description

The ERC721URIStorage contract also implements the `supportsInterface` function, if the contract inherits this contract, the function needs to specify the overridden contract "ERC721URIStorage", otherwise, it will affect the normal compilation of the contract.

Sirius.sol:L40-L41

```
function supportsInterface(bytes4 interfaceId)
        public
        view
        override(ERC721, ERC721Enumerable)
        returns (bool)
    {
        return super.supportsInterface(interfaceId);
}
```

Recommendation

Consider adding "ERC721URIStorage" in the override.

SALUS

5. Update the test information of the contract

Severity: Informational	Category: Code Quality
Target: - Sirius.sol	

Description

The contract name and the token name/symbol should be updated.

- Sirius.sol:L8

```
contract Test7 is ERC721, ERC721Enumerable, ERC721URIStorage {
```

- Sirius.sol:L20

```
constructor() ERC721("Testt7", "TEST") {
```

Recommendation

Consider updating the contract name and the token name/symbol.

SALUS

6. Use of floating pragma

Severity: Informational	Category: Configuration
Target: - Sirius.sol	

Description

```
pragma solidity ^0.8.0;
```

The contract uses a floating compiler version. So the code may compile to different bytecodes with different compiler versions. Use a locked pragma statement to get a deterministic bytecode. Also use the latest Solidity version to get all the compiler features, bug fixes and optimizations.

Recommendation

It is recommended to use a locked Solidity version throughout the project. It is also recommended to use the most stable and up-to-date version.

SALUS

7. Lacking of importing contract

Severity: Informational	Category: Code Integrity
Target: - Sirius.sol	

Description

The Test7 contract inherits from the ERC721Enumerable contract but without explicit imports for ERC721Enumerable.sol.

Sirius.sol:L5-L8

```
import "./ERC721URIStorage.sol";
import "./Counters.sol";

contract Test7 is ERC721, ERC721Enumerable, ERC721URIStorage {
    //...
}
```

Similarly, the onlyOwner modifier is used in contract, suggesting the intention to utilize the Ownable contract from OpenZeppelin. However, there is no clear indication of an import statement for the Ownable contract or any implementation of the onlyOwner modifier within the contract itself.

Sirius.sol:L32, L38, L46, L52, L77, L84

```
function modifyIPFS(uint256 _num, string memory _ipfs) onlyOwner public returns (bool){}

function add_list_IPFS(string[] memory _ipfs) onlyOwner public returns (bool){}

function addIPFS(string memory _ipfs) onlyOwner public returns (bool){}

function lowerCountIPFS(uint256 _count) onlyOwner public returns (bool){}

function add_whitelist(address[] memory listAdress) onlyOwner public returns (bool){}

function remove_whitelist(address[] memory listAdress) onlyOwner public returns (bool){}
```

It may indicate that the contracts imported by the project is not fully implemented according to the OpenZeppelin library standards, or that the relevant import is missing, which may lead to some risks.

Recommendation

It is recommended to import all the relevant contracts and verify that the imported contracts are from trusted sources like OpenZeppelin. Additionally, ensure that the imported contracts match the intended functionality.

SALUS

8. Missing zero address check

Severity: Informational	Category: Data Validation
Target: - Sirius.sol	

Description

It is considered a security best practice to verify addresses against the zero address in the constructor or setting. However, this precautionary step is absent for the variables highlighted below.
Sirius.sol:L77-L82

```
function add_whitelist(address[] memory listAdress) onlyOwner public returns (bool) {
    for(uint i=0;i<listAdress.length;i++){
        whitelist[listAdress[i]]=true;
    }
    return true;
}
```

Recommendation

Consider adding zero-address check.

SALUS

9. Gas Optimizations

Severity: Informational	Category: Gas Optimization
Target: - Sirius.sol	

Description

1.The lowerCountIPFS() function could delete storage instead of assigning an empty string.

- Sirius.sol:L54 - L56

```
for(uint i=_count+1;i<=count_ipfs;i++){
    IPFS_URI[i]="";
}
```

2.MAX_SUPPLY could be made a constant.

- Sirius.sol:L12

```
uint256 public MAX_SUPPLY=300000;
```

3.The count_ipfs variable is read multiple times in the add_list_IPFS() function and the lowerCountIPFS() function. The functions could have a local variable.

- Sirius.sol:L39 - L42

```
for(uint i=0;i<_ipfs.length;i++){
    count_ipfs++;
    IPFS_URI[count_ipfs]=_ipfs[i];
}
```

- Sirius.sol:L53 - L56

```
require(_count<count_ipfs && _count>0, 'number invalid' );
for(uint i=_count+1;i<=count_ipfs;i++){
    IPFS_URI[i]="";
}
```

Recommendation

Consider following the provided recommendations.

SALUS

10. Spelling error

Severity: Informational	Category: Code Quality
Target: - Sirius.sol	

Description

Spelling error for the name of parameters.

- Sirius.sol:L77

```
function add_whitelist(address[] memory listAdress) onlyOwner public returns (bool) {
```

- Sirius.sol:L84

```
function remove_whitelist(address[] memory listAdress) onlyOwner public returns (bool) {
```

Recommendation

Consider replacing it with listAddress.

SALUS

11. Dead code

Severity: Informational	Category: Code Quality
Target: - Sirius.sol	

Description

Unused codes could be removed.

- Sirius.sol:L21, L23 - L29

```
//start at 1
_tokenIdCounter.increment();
//add_list_IPFS(["ipfs://QmPW47twFUQ52haVb11vg5iqg9bbJS6jrG7GmZt7JQrro6","ipfs://QmWjbcm
4x4sNmNoX86oFD2Gnjoxa5XqpgwvqAL3coBGAmv"]);
//modifyIPFS(1,"ipfs://QmPW47twFUQ52haVb11vg5iqg9bbJS6jrG7GmZt7JQrro6");
//modifyIPFS(2,"ipfs://QmWjbcm4x4sNmNoX86oFD2Gnjoxa5XqpgwvqAL3coBGAmv");
/*
modifyIPFS(3,"ipfs://Qmaf9CR5CeYhqi9Vw5Ci3f1CteiHtuVQzhYbyV7HQ1x2Dz");
modifyIPFS(4,"ipfs://Qmf7i1zoRf6vHSuZLY4XjRW3PoKEcnrztJQYiJwsVcWyn8");
modifyIPFS(5,"ipfs://QmPRofyjKyBmDqe78Njn1pBxghZhu2nintKaxccvf4RAxZ");*/
```

Recommendation

Consider removing them.

SALUS

Appendix

Appendix 1 - Files in Scope

This audit covered the following file provided by the client:

File	SHA-1 hash
Sirius.sol	0a55e62b0803a89c5ea08f83b60e1a476fa7511a

SALUS