SALUS SECURITY

JUNE 2023

CODE SECURITY ASSESSMENT

SIRIUS

Overview

Project Summary

- Name: Sirius
- Address: 0xf8DDA7b3748254d562f476119B0aE6044bAd10a5
- Platform: Polygon
- Language: Solidity
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Sirius
Version	v2
Type	Solidity
Date	June 27 2023
Logs	June 9 2023; June 27 2023

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	2
Total Low-Severity issues	2
Total informational issues	5
Total	9

Contact

E-mail: support@salusec.io

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.

Content

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.

Findings

2.1 Summary of Findings

ID	Title	Severity	Category	Status
1	Users may not be able to claim tokens if maxSupply is exceeded	Medium	Business Logic	Acknowledged
2	Inconsistency maxSupply between code and whitepaper	Medium	Configuration	Resolved
3	Misuse of variables in TokenAvailableToClaim function	Low	Business Logic	Acknowledged
4	Centralization risk	Low	Centralization	Mitigated
5	Use of floating pragma	Informational	Configuration	Acknowledged
6	Typo on function parameters	Informational	Code Quality	Acknowledged
7	Use of deprecated ERC777 standard	Informational	Business Logic	Acknowledged
8	Missing events for critical functions	Informational	Auditing and Logging	Acknowledged
9	Gas optimization suggestions	Informational	Gas Optimization	Acknowledged

2.2 Notable Findings

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

1. Users may not be able to claim tokens if maxSupply is exceeded	
Severity: Medium	Category: Business Logic
Target: - Token777.sol	

Description

The maximum supply of tokens is defined in the ERC777 contract. A check is performed in the _mint function to ensure that the total supply does not exceed the maxSupply. However, in the add_Users_Claiming_List and change_Users_TokensToClaim functions, which are operations performed by the owner, there is no check against the maxSupply. Therefore, if the owner mistakenly lets the sum of userTokensToClaim be a value exceeding the maxSupply, users will be unable to claim further rewards once the claimed amount reaches the supply limit.

Recommendation

Consider adding a check in the add_Users_Claiming_List and change_Users_TokensToClaim functions to ensure that the claimable tokens do not exceed maxSupply.

Status

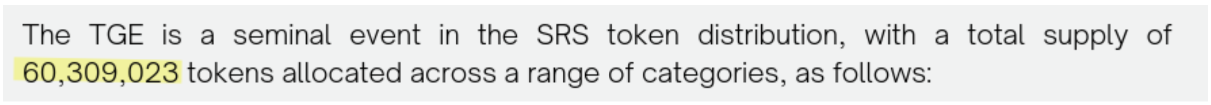
This issue has been acknowledged by the team. The team stated that they have several layers of safeguards in place to prevent it.

2. Inconsistency maxSupply between code and whitepaper

Severity: Medium	Category: Configuration
Target: - ERC777.sol	

Description

According to the whitepaper, the total supply of tokens should be 60,309,023.


The TGE is a seminal event in the SRS token distribution, with a total supply of 60,309,023 tokens allocated across a range of categories, as follows:

However, the defined value of _maxSupply in the code is 61309023 * 1e18.

ERC777.sol:L42

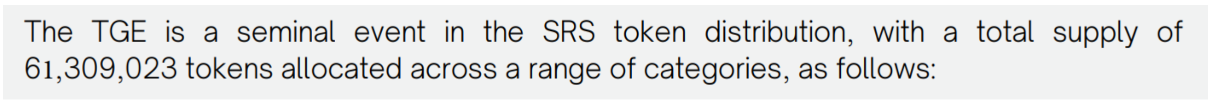
```
uint256 private _maxSupply=61309023*1e18;
```

Recommendation

Consider fixing the mismatch between code and whitepaper.

Status

This issue has been resolved by the team. According to their new whitepaper:


The TGE is a seminal event in the SRS token distribution, with a total supply of 61,309,023 tokens allocated across a range of categories, as follows:

3. Misuse of variables in TokenAvailableToClaim function	
Severity: Low	Category: Business Logic
Target: - Token777.sol	

Description

Token777.sol:L100-L125

```
function TokenAvailableToClaim(address _user) public view returns (uint256){
    if(block.timestamp>userStartClaimPeriod[msg.sender]){
        ...
    }
}
```

Here should check userStartClaimPeriod for _user instead of msg.sender.

Recommendation

Consider changing `if(block.timestamp>userStartClaimPeriod[msg.sender])` to `if(block.timestamp>userStartClaimPeriod[_user])`.

Status

This issue has been acknowledged by the team.

4. Centralization risk	
Severity: Low	Category: Centralization
Target: - Token777.sol	

Description

There is a privileged role in the Sirius_by_Humanity contract.

The owner of the Sirius_by_Humanity contract

- Can update the whitelist and the amount of tokens users can claim;
- Can withdraw all tokens inside the contract through return_To_Owner function.

If the owner's private key is compromised, an attacker could add himself to the whitelist and set a huge number of tokens he can claim. If the privileged account is a plain EOA account, this can be worrisome and pose a risk to the other users.

Recommendation

Consider transferring the privileged roles to multi-sig accounts.

Status

This issue has been mitigated by the team. The team has transferred the ownership to a Gnosis Safe multi-sig wallet.

2.3 Informational Findings

5. Use of floating pragma	
Severity: Informational	Category: Configuration
Target: - Token777.sol	

Description

```
pragma solidity ^0.8.18;
```

The Sirius_by_Humanity contract uses a floating compiler version ^0.8.18.

Using a floating pragma is discouraged, as 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.

Status

This issue has been acknowledged by the team.

6. Typo on function parameters	
Severity: Informational	Category: Code Quality
Target: - Token777.sol	

Description

Token777.sol:L40

```
function add_Users_Claiming_List(uint256[] memory listTokensToClaim, address[] memory
listAdress) onlyOwner public returns (bool)
```

Token777.sol:L60

```
function change_Users_TokensToClaim(uint256[] memory listTokensToClaim, address[] memory
listAdress) onlyOwner public returns (bool)
```

Token777.sol:L86

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

These are spelling mistakes in list**Adress**.

Recommendation

Consider changing listAdress to listAd**d**ress.

Status

This issue has been acknowledged by the team.

7. Use of deprecated ERC777 standard	
Severity: Informational	Category: Business Logic
Target: - Token777.sol	

Description

As of v4.9, OpenZeppelin’s implementation of ERC-777 is deprecated. This decision was taken mainly due to 3 factors:

1. Huge potential attack vectors arise because of callbacks;
2. The expensive gas architecture of standard;
3. Complex to implement.

Further details can be found here.

Recommendation

Consider switching to ERC20.

Status

This issue has been acknowledged by the team.

8. Missing events for critical functions	
Severity: Informational	Category: Auditing and Logging
Target: - Token777.sol	

Description

The functions that make critical changes should emit events. Events allow capturing the changed states so that off-chain tools/interfaces can register such changes with timelocks that allow users to evaluate them. The alternative of directly querying on-chain contract state for such changes is not considered practical for most users/usages.

Missing events do not promote transparency and if such changes immediately affect users' perception of fairness or trustworthiness, they could exit the protocol causing a reduction in protocol users.

Throughout the Sirius codebase, events are lacking in the whitelist modification functions (e.g. add_Users_Claiming_List(), change_Users_TokensToClaim(), remove_from_List()).

Recommendation

Consider adding events to all functions that change the whitelist.

Status

This issue has been acknowledged by the team.

9. Gas optimization suggestions	
Severity: Informational	Category: Gas Optimization
Target: - Token777.sol	

Description

Token777.sol:L43
Token777.sol:L63
Token777.sol:L87

```
for(uint i=0;i<listAdress.length;i++)
```

Since the values default to zero, the initialization can be removed to save gas.

Token777.sol:L40

```
function add_Users_Claiming_List(uint256[] memory listTokensToClaim, address[] memory
listAdress) onlyOwner public returns (bool)
```

Token777.sol:L60

```
function change_Users_TokensToClaim(uint256[] memory listTokensToClaim, address[] memory
listAdress) onlyOwner public returns (bool)
```

Token777.sol:L86

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

For read-only function parameters, a common gas-saving practice is to use calldata instead of memory.

Token777.sol:L47

```
if(userIsWhitelisted[address_user]==false)
```

Token777.sol:L66

```
if(userIsWhitelisted[address_user]==true)
```

Token777.sol:L102

```
if (userIsWhitelisted[_user]==false)
```

Token777.sol:L129

```
require(userIsWhitelisted[msg.sender]==true,"Not Whitelisted");
```

It is unnecessary to compare boolean variables to boolean literals.

Recommendation

Consider removing unnecessary variable initialization.
Consider using calldata instead of memory for read only function parameters to save gas.
Consider removing the comparison to boolean literals.

Status

This issue has been acknowledged by the team.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files from address 0xf8DDA7b3748254d562f476119B0aE6044bAd10a5:

File	SHA-1 hash
Token777.sol	4af5d6d2b0ccf0f5ea7c4e6389822b006fa7e705
IERC777.sol	e31fce05afc5e49c55620d42b7d380f46a198370
Context.sol	f2f4dfdb86e9435268219ff58ef3db28f9f98a11
IERC777Recipient.sol	0ac412b88ab50713a897b0ad4e52f0f2cf5510c5
IERC1820Registry.sol	3f38a302de4a86585ec621d99d2a7e76182116ae
Address.sol	a9ce1425371ef4514494abdbce0b1e0bd770a23e
Ownable.sol	6e1d4b1c71b11ab929022ce1194ded1b6153788e
ERC777.sol	d0ed7feb4e626578e6de81907018ba2ffd0bee57
IERC777Sender.sol	8c113b25b40235c62fc7a6020f04d84b36d8892f
ReentrancyGuard.sol	586a4b7c629326cd95ad1fa4ec56828eb653f940
IERC20.sol	f61145ff3132ad25b2906fde7381081510259789