SALUS SECURITY

NOV 2023

CODE SECURITY ASSESSMENT

MYSHELL

Overview

Project Summary

- Name: MyShell - CallerInfo
- Platform: opBNB
- Address: 0x4F9CE7a71Eb3795d7d38694Fdb0D897dD055E26d
- Language: Solidity
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	MyShell - CallerInfo
Version	v2
Type	Solidity
Dates	Nov 15 2023
Logs	Nov 14 2023; Nov 15 2023

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	0
Total Low-Severity issues	1
Total informational issues	3
Total	4

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 **4**
1.1 About SALUS 4
1.2 Audit Breakdown 4
1.3 Disclaimer 4
Findings **5**
2.1 Summary of Findings 5
2.2 Notable Findings 6
1. Empty name allowed in setCaller() 6
2.3 Informational Findings 7
2. Gas optimization suggestions 7
3. Floating pragma version 8
4. Redundant code 9
Appendix **10**
Appendix 1 - Files in Scope 10

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	Empty name allowed in setCaller()	Low	Data Validation	Acknowledged
2	Gas optimization suggestions	Informational	Gas Optimization	Acknowledged
3	Floating pragma version	Informational	Configuration	Acknowledged
4	Redundant code	Informational	Redundancy	Acknowledged

2.2 Notable Findings

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

1. Empty name allowed in setCaller()

Severity: Low	Category: Data Validation
Target: - CallerInfo.sol	

Description

CallerInfo.sol:L13-L18

```
function setCaller(string memory name) public {
    bytes memory nameBytes = bytes(name);
    require(nameBytes.length <= 10, "Name cannot be longer than 10 characters");
    results[msg.sender] = name;
    emit CallerSet(msg.sender, name);
}
```

Users can call the setCaller() function with an empty name string.

Since the results mapping is initialized to an empty string for all addresses by default, allowing an empty name might lead to confusion. It could make it challenging to distinguish between users who have intentionally set an empty name and those who haven't set a name at all.

Recommendation

Consider updating the setCaller() function to disallow empty names.

Status

This issue has been acknowledged by the team.

2.3 Informational Findings

2. Gas optimization suggestions	
Severity: Informational	Category: Gas Optimization
Target: - CallerInfo.sol	

Description

CallerInfo.sol:L13

```
function setCaller(string memory name) public {
    ...
}
```

In the CallerInfo contract, consider marking the setCaller() and getCaller() functions as external if they are not intended to be called from within the contract.

In the setCaller() function, consider using string calldata instead of string memory for the name parameter to optimize gas consumption.

CallerInfo.sol:L15

```
function setCaller(string memory name) public {
    ...
    require(nameBytes.length <= 10, "Name cannot be longer than 10 characters");
    ...
    }
```

Consider modifying the condition to ‘nameBytes.length < 11’ to optimize gas consumption.

Recommendation

Consider making changes based on the above suggestions.

Status

This issue has been acknowledged by the team.

3. Floating pragma version

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

Description

```
pragma solidity ^0.8.0;
```

Using a floating pragma statement 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.

4. Redundant code

Severity: Informational	Category: Redundancy
Target: - CallerInfo.sol	

Description

CallerInfo.sol:L21

```
function getCaller() public view returns (string memory) {
        return string(results[msg.sender]);
}
```

The content of ‘results[msg.sender]’ is of string type, so there is no need to convert the variable into string in the `getCaller()` function.

Recommendation

Consider removing the redundant code.

Status

This issue has been acknowledged by the team.

Appendix

Appendix 1 - Files in Scope

This audit covered the following file in address

0x4F9CE7a71Eb3795d7d38694Fdb0D897dD055E26d

File	SHA-1 hash
CallerInfo.sol	a26d0108aaabd442b5aca749b5ac393fc9be83ba