SALUS SECURITY

APR 2023

CODE SECURITY ASSESSMENT

YULIVERSE

Overview

Project Summary

- Name: Yuliverse
- Version: commit 22e9fbc
- Platform: Binance Smart Chain
- Language: Solidity
- Repository: https://github.com/yuliverse-official/currency
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Yuliverse
Version	v2
Type	Solidity
Dates	Apr 11 2023
Logs	Apr 11 2023; Apr 17 2023

Vulnerability Summary

Total High-Severity issues	1
Total Medium-Severity issues	0
Total Low-Severity issues	1
Total informational issues	3
Total	5

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	Execute function lacks access control	High	Business Logic	Resolved
2	Unnecessary receive function	Low	Business Logic	Resolved
3	Floating compiler version	Informational	Configuration	Acknowledged
4	Gas optimization suggestions	Informational	Gas Optimization	Resolved
5	Missing Event in critical function	Informational	Auditing and Logging	Acknowledged

2.2 Notable Findings

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

1. Execute function lacks access control	
Severity: High	Category: Business Logic
Target: - ARG/ARG.sol	

Description

ARG/ARG.sol:L158-L162

```
function execute(address _target, bytes memory _data) external payable {
    (bool success, ) = _target.call{value: msg.value}(_data);
    require(success, "execute failed");
}
```

The Create2Proxy contract includes a deployContract function that utilizes the Create2 opcode to deploy contracts. If the AlternateRealityGemCurrency contract is deployed using this function, the Create2Proxy contract will be granted privileged roles in the AlternateRealityGemCurrency contract based on the code in its constructor function.

However, the Create2Proxy contract also has an execute function with external visibility, which means any external user can call the function by providing the _target and _data parameters to call the contract at the _target address.

In a specific situation, if an administrator uses the deployContract function to deploy AlternateRealityGemCurrency contract, it's possible for a malicious user to use the execute function with the _target set as the AlternateRealityGemCurrency contract address and the _data set as the call data for a function with access control in the AlternateRealityGemCurrency contract. Then, the malicious user can successfully call that function, because the msg.sender is the Create2Proxy contact address, which allows the user to bypass the permission check in AlternateRealityGemCurrency contract.

Proof of Concept

Using the foundry toolkit test file:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.7;

import "lib/forge-std/src/Test.sol";
import "../src/ARG/ARG.sol";

contract ARGTest is Test {
    Helper helper;
    Create2Proxy create2Proxy;
    AlternateRealityGemCurrency alternateRealityGemCurrency;
    address deployer = makeAddr("deployer");
    address exploiter = makeAddr("exploiter");
    function setUp() public {
        vm.startPrank(deployer);
        helper = new Helper();
        create2Proxy = new Create2Proxy();
        bytes memory creationCode = helper.getCreationCode();
        alternateRealityGemCurrency =
AlternateRealityGemCurrency(create2Proxy.deployContract(creationCode, 0));
        vm.stopPrank();
    }

    function testExploit() external {
        vm.startPrank(exploiter);
        bytes memory execData = abi.encodeWithSignature("mint(address,uint256)",
exploiter, 1 * 1e18);
        create2Proxy.execute(address(alternateRealityGemCurrency), execData);
        vm.stopPrank();
        emit log_named_decimal_uint("exploiter token balance",
alternateRealityGemCurrency.balanceOf(exploiter),
alternateRealityGemCurrency.decimals());
        verifyEXP();
    }

    function verifyEXP() internal {
        assertEq(alternateRealityGemCurrency.balanceOf(exploiter), 1 * 1e18);
    }
}
```

This PoC shows that when deploying AlternateRealityGemCurrency contract through the Create2Proxy contract's deployContract function, any external user can use the execute function to call functions that have access control in the AlternateRealityGemCurrency contract.

Recommendation

Consider adding access control to the execute function to ensure that it can only be called by the role with the corresponding privileges.

Status

This issue has been resolved by the team in commit 3dbd3c2.

2. Unnecessary receive function	
Severity: Low	Category: Business Logic
Target: - ARG/ARG.sol	

Description

ARG/ARG.sol:L140

```
receive() external payable {}
```

There is a payable **receive** function in the Create2Proxy contract that allows the contract to receive funds from other addresses. However, there is no withdrawal logic in the Create2Proxy contract.

As a result, if one accidentally sends BNB to the Create2Proxy contract, the funds are locked in the contract and there is no way to move the funds out.

Recommendation

Consider removing the payable receive function. Receiving BNB is an unexpected function of the Create2Proxy contract.

Status

This issue has been resolved by the team in commit 3dbd3c2.

2.3 Informational Findings

3. Floating compiler version	
Severity: Informational	Category: Configuration
Target: - all	

Description

```
pragma solidity ^0.8.7;
```

The Yuliverse contracts use a floating compiler version ^0.8.7.

Using a floating pragma statement is discouraged, as code may compile to different bytecodes with different compiler versions. Using a locked pragma statement can get a deterministic bytecode. Also using the latest Solidity version can 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. Gas optimization suggestions

Severity: Informational	Category: Gas Optimization
Target: - ARG/ARG.sol	

Description

ARG/ARG.sol:L102-L106

```
function claimTokens(
    uint256 txId,
    uint256 amount,
    bytes memory signature
) public {
```

ARG/ARG.sol:L121

```
function depositTokens(uint256 amount) public {
```

ARG/ARG.sol:L163

```
function calculateContractAddress(bytes memory bytecode, uint256 _salt, address
proxyAddress) public pure returns(address) {
```

Above functions are not called inside the contract, thus, the visibility can be changed to external to save gas.

Recommendation

Consider changing the visibility of functions that are not called inside contracts to external.

Status

This issue has been resolved by the team in commit 3dbd3c2.

5. Missing Event in critical function

Severity: Informational	Category: Auditing and Logging
Target: - ARG/ARG.sol	

Description

ARG/ARG.sol:L48

```
function snapshot() public onlyRole(DEFAULT_ADMIN_ROLE) {
```

ARG/ARG.sol:L52

```
function pause() public onlyRole(PAUSER_ROLE) {
```

ARG/ARG.sol:L56

```
function unpause() public onlyRole(PAUSER_ROLE) {
```

ARG/ARG.sol:L60-L62

```
function setSigner(address signer, bool _new)
    external
    onlyRole(DEFAULT_ADMIN_ROLE)
```

ARG/ARG.sol:L68

```
function setDepositLock(bool _new) external onlyRole(DEFAULT_ADMIN_ROLE) {
```

ARG/ARG.sol:L72

```
function mint(address to, uint256 amount) public onlyRole(MINTER_ROLE) {
```

The Governor should emit events for functions that change critical parameters. Events allow the changed parameters to be captured so that off-chain tools/interfaces can register such changes with timelocks that allow users to evaluate them and consider whether to engage/exit based on how they perceive the changes to affect the trustworthiness of the protocol. The alternative of directly querying the on-chain contract state for such changes is considered impractical for most users/usages.

Missing events do not promote transparency, and if such changes immediately affect users' perceptions of fairness or trustworthiness, they may leave the protocol, leading to a reduction in protocol users.

Throughout the Yuliverse codebase, events are lacking in the privileged setter functions (e.g. setSigner()), pause functions (e.g. pause()), mint() function and snapshot() function.

Recommendation

Consider adding events to all privileged functions that change critical parameters.

Status

This issue has been acknowledged by the team.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit 22e9fbc:

File	SHA-1 hash
ARG/ARG.sol	6abd0184fb2c485bc79c4c4c74590ec8d81c196c
ARG/LibSigVerify.sol	1baf2371b5493d3b8b554dd7e576dd19417a8783