SALUS SECURITY

APR 2024

CODE SECURITY ASSESSMENT

ULTIVERSE

Overview

Project Summary

- Name: Ultiverse - chip
- Platform: Ethereum
- Language: Solidity
- Repository:
    - https://github.com/ultiverse-io/ChipsContract
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Ultiverse - chip
Version	v4
Type	Solidity
Dates	Apr 29 2024
Logs	Apr 22 2024; Apr 23 2024; Apr 26 2024; Apr 29 2024

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	0
Total Low-Severity issues	2
Total informational issues	1
Total	3

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	User's ETH may be locked in the contract	Low	Business Logic	Acknowledged
2	Potential cross-chain replay	Low	Business Logic	Acknowledged
3	Missing two-step transfer ownership pattern	Informational	Business Logic	Resolved

2.2 Notable Findings

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

1. User's ETH may be locked in the contract

Severity: Low	Category: Business Logic
Target: - src/Pool/ChipStakingPool.sol	

Description

In the ChipStakingPool, after users deposit funds, they can use the claimRefund() function to withdraw ETH from the pool.
src/Pool/ChipStakingPool.sol:L133-L167

```
function claimRefund(uint256 quantity, bytes32[] calldata proof) external whenNotPaused
{
    ...
    if (userRefundCount[_msgSender()] > 0) {
        revert UserAlreadyRefunded();
    }

    userRefundCount[_msgSender()] = quantity;
    ...
}
```

However, the highlighted portion of the code above restricts users to performing the refund operation only once.

Attach Scenario
This could result in two scenarios where funds might be permanently locked in the contract:
a) If a user performs a refund and then obtains a new proof and makes another deposit, the new deposit will be permanently locked in the contract.
b) If the amount corresponding to the user's refund proof does not match the ETH quantity stored in the contract, the remaining ETH will be permanently locked in the contract.

Recommendation

If users are restricted to only one claim operation, the claimRefund() function should verify if userEntries equals the quantity specified in the proof. Additionally, the deposit() function needs to prohibit users who have already claimed from depositing funds again.

Status

This issue has been acknowledged by the team. And the team states that the refunds feature will only be turned on when the deposit feature is turned off, and that by design of the protocol, users will not always be able to refund an amount equal to the ETH of the deposit.

2. Potential cross-chain replay

Severity: Low	Category: Business Logic

Target:
- src/Chip/ChipLaunch.sol
- src/Pool/ChipStakingPool.sol

Description

There are multiple operations in the codebase that require validation of proof before they can be executed.
src/Chip/ChipLaunch.sol:L39-L58

```
function mint(
    ...
    bytes32[] calldata proof
) external whenNotPaused {
    bytes32 leaf = keccak256(bytes.concat(keccak256(abi.encode(_msgSender(),
maxQuantity))));
    if (!MerkleProof.verify(proof, merkleTreeRoot, leaf)) {
        revert InvalidProof();
    }
}
```

src/Pool/ChipStakingPool.sol:L51-L131

```
function deposit(
    ...
    bytes32[] calldata proof
) external payable whenNotPaused {
    bytes32 leaf = keccak256(
        bytes.concat(keccak256(abi.encode(_msgSender(), goldLots, silverLots)))
    );
    if (!MerkleProof.verify(proof, poolConfig.merkleTreeRoot, leaf)) {
        revert InvalidProof();
    }
    ...
}
```

However, the proof does not contain the chainId field, which means that the proof that the user gets in chain1 is still valid in chain2.
If the team intends to deploy the project in multiple chains, the replay of proof across chains may lead to unexpected authorizations and loss of funds.

Recommendation

Consider including a check for chainId in the proof. For example a chainId field could be added to the leaf:

```
bytes32 leaf = keccak256(
    bytes.concat(keccak256(abi.encode(_msgSender(), chainId, goldLots, silverLots)))
);
```

Status

This issue has been acknowledged by the team. And the team states that this protocol will not be deployed on multiple chains

2.3 Informational Findings

3. Missing two-step transfer ownership pattern

Severity: Informational	Category: Business logic
Target: - src/Chip/ChipVesting.sol - src/Chip/ChipLaunch.sol - src/Pool/ChipStakingPool.sol	

Description

The ChipVesting, ChipLaunch and ChipStakingPool contract inherits from the Ownable contract. This contract does not implement a two-step process for transferring ownership. Thus, ownership of the contract can easily be lost when making a mistake in transferring ownership.

Recommendation

Consider using the Ownable2Step contract from OpenZeppelin instead.

Status

The team has resolved this issue in commit 658dfce5.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit 271a373:

File	SHA-1 hash
ChipLaunch.sol	1263eb3d0d22881be5f3ee0a67833f2e5f461e4c
ChipVesting.sol	def5478616d214222fbc4120c46e18e4b4e87ee4
GoldChip.sol	84373812f917cd59f9deb0de5861fc9187d2ad75
ChipStakingPool.sol	57201aab889ff05bcae1eb5701852c2527173549

And audited the team's new feature in commit 0381064:

File	SHA-1 hash
ChipStakingPool.sol	d86336d3d2fef9f26750bae4ae49c6a43fff2d96

new feature in commit 5259de7:

File	SHA-1 hash
ChipStakingPool.sol	9a7d86dd9be2134e350b99f6d78543b0a4a36dec