SALUS SECURITY

OCT 2024

CODE SECURITY ASSESSMENT

INFINI

Overview

Project Summary

- Name: Infini - card
- Platform: EVM-compatible chains
- Language: Solidity
- Repository:
    - https://github.com/infini-money/infini-card-contract
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Infini - card
Version	v2
Type	Solidity
Dates	Oct 28 2024
Logs	Oct 17 2024; Oct 28 2024

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	0
Total Low-Severity issues	1
Total informational issues	2
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	Use safeTransfer() instead of transfer()	Low	Risky External Calls	Resolved
2	Redundant code	Informational	Redundancy	Acknowledged
3	Use of floating pragma	Informational	Configuration	Acknowledged

2.2 Notable Findings

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

1. Use safeTransfer() instead of transfer()

Severity: Low	Category: Risky External Calls
Target: - src/library/StrategyUtils.sol - src/strategys/ethena/InfiniEthenaStrategyManager.sol	

Description

src/library/StrategyUtils.sol:L32 - L43

```
function _transferAsset(
    address token,
    uint256 amount,
    address to
) internal {
    if (token == NATIVE_TOKEN) {
        (bool res, ) = payable(to).call{value: amount}("");
        require(res);
    } else {
        IERC20(token).transfer(to, amount);
    }
}
```

src/strategys/ethena/InfiniEthenaStrategyManager.sol:L27 - L38

```
function settle(uint256 unSettleProfit) external override onlyRole(ADMIN_ROLE) {
    uint256 profit = _getProfit();
    if (profit < unSettleProfit) revert ProfitIsNotEnough();

    uint256 protocolProfit = unSettleProfit * carryRate / 10000;
    uint256 settleProfit = unSettleProfit - protocolProfit;

    IERC20(profitToken).transfer(infiniTreasure, protocolProfit);
    IERC20(profitToken).transfer(strategyVault, settleProfit);

    emit Settlement(profitToken, protocolProfit, settleProfit);
}
```

Tokens not compliant with the ERC20 specification could return false from the transfer function call to indicate the transfer fails, while the calling contract would not notice the failure if the return value is not checked. Checking the return value is a requirement, as written in the EIP-20 specification:

```
Callers MUST handle false from returns (bool success). Callers MUST NOT assume that
false is never returned!
```

Recommendation

Consider using the SafeERC20 library implementation from OpenZeppelin and call safeTransfer or safeTransferFrom when transferring ERC20 tokens.

Status

This issue has been resolved by the team with commit 1b480ee.

2.3 Informational Findings

2. Redundant code	
Severity: Informational	Category: Redundancy
Target: - src/InfinicardController.sol - src/library/StrategyUtils.sol	

Description

Unused code should be removed before deploying the contract to mainnet. We have identified the following variables are not being utilized:
src/InfinicardController.sol:L9

```
address public constant WITHDRAWOUT_STRATEGY_ADDRESS = address(0);
```

src/library/StrategyUtils.sol:L9 - L11

```
uint256 public constant _ADDRESS_MASK =
0x000000000000000000000000ffffffffffffffffffffffffffffffffffffffff;

uint256 public constant _REQUEST_MASK = uint256(1) << 255;
```

Recommendation

Consider removing the redundant code.

Status

This issue has been acknowledged by the team.

3. Use of floating pragma

Severity: Informational	Category: Configuration
Target: - All	

Description

```
pragma solidity ^0.8.20;
```

All contracts use a floating compiler version `` `^0.8.20` ``.

Using a floating pragma `` `^0.8.20` `` 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.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit 38ea00f:

File	SHA-1 hash
src/InfiniCardController.sol	764f8708d6af5ea5850eb214d703767a55d39114
src/InfiniCardVault.sol	bfbcec80c6279b567537e5a246b1ee61da234c6e
src/library/StrategyUtils.sol	7bfb7cd2b8d57c64eeb814faf38b6a0947d73ed4
src/library/VaultUtils.sol	ef16a04c8bbb8f2c17d60b08eba2a430fcaca360
src/strategys/BaseStrategyManager.sol	a0eb1ac4056cfe642170325315ad0a830b1bb80d
src/strategys/BaseStrategyVault.sol	5f4cfd7f3cc77582ee92f329a90644d61dd1d7ea
src/strategys/ethena/InfiniEthenaStrategyManager.sol	9935ed7972a0e0d5cb4cb9a7698cb00d552adea8
src/strategys/ethena/InfiniEthenaStrategyVault.sol	4eec173251f5ec7276986b0b38684d5113993f8a
src/strategys/morpho/InfiniMorphoStrategyVault.sol	7ac2f885d48a27280c275edd7a699d533c571696