CodeHawks

First Flight
Competition

2024-08 MYCUT

X @JJS_OnChain

JORDAN J. SOLOMON

SUMMARY

Prepared By: Jordan J. Solomon

Competition Type: CodeHawks First Flight

Competition Dates: Aug 29th, 2024 → Sep 5th, 2024

Audit Summary

First Flight audits are designed to help beginner security researchers improve their skills. These audits test fundamental concepts of blockchain development and the potential attacks associated with them.
MyCut follows the same principle. It is a simple protocol that enables fund deposits and reward distribution. The codebase has a low nSLOC (non-comment source lines of code) count, making it more approachable for beginner auditors without overwhelming them.

TABLE OF CONTENTS

PROJECT OVERVIEW

Project Summary

Project Name	**MyCut First Flight Competition**
Language	Solidity
Code Base	https://github.com/Cyfrin/2024-08-MyCut
Commits	**N/A**

Audit Summary

End Date	**5th of Sep 2024**
Methodology	Manual Review,

Vulnerabilities Summary

🔴 **High**	**0**
🟠 Medium	0
🟢 Low	1
● Info/Gas	**2**

AUDIT SCOPE & METHODOLOGY

Vulnerabilities Classification

● High	Impact is extremely high on the protocol, with a combination of highly likely to happen
● Medium	Risk of exploits that may occur and impact could be important.
● Low	Minor issues that are less likely to occur and if they would the risk is not high.
● Info/Gas	Simple improvements that have no risk for the protocol. Miss use of best practices, gas efficiency etc.

Methodology

- Manual Review - Walking through the code. First getting an understanding of the code base. Then to try and find vulnerabilities.

FINDINGS

LOW

L-1	DoS attack in Constructor of Pot.sol

INFO/GAS

I-1	Unused error in Pot.sol
I-1	Miss use of the naming conventions for different storage types

[L-1] Dos Attack in Constructor of Pot.sol

Type	Severity	Location
Denial of Service	Low	Pot.sol: 32

Description

In the Pot.sol constructor there are two arrays that are passed: **players** and **rewards**. They are copied into the state variables **i_players** and **i_rewards**. Assuming that the player array and the rewards array are equal, each of the players is assigned a sum of the rewards using the **playersToRewards** mapping. To do that the protocol uses a for loop to copy the data into the mapping.

Impact

When using a list that can be potentially unlimited a DoS (Denial of Service) attack could be exploited. If an attacker decided to, they could enter the contents with a big number of different addresses and make the array enormous. This will cause the gas fees for the protocol to be unreasonably expensive, and render the protocol unusable.

Recommended Mitigation

Add a limit to the amount of players allowed to participate in each contest.

[I-1] Unused error in Pot.sol

Type	Severity	Location
Unused Code	Info/Gas	Pot.sol: 9

Description

The error `error Pot__InsufficientFunds();` is declared in Pot.sol, however never used.

Impact

More complex code and costlier gas prices.

Recommended Mitigation

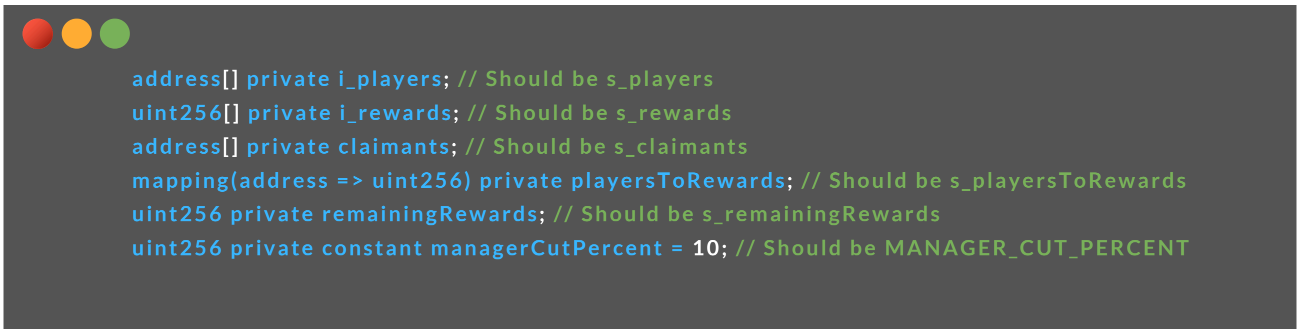
Remove unused code, or find use for it.

[I-2] Miss use of the naming conventions for different storage types

Type	Severity	Location
Best Practices	Info/Gas	Pot.sol: 12-20

Description

In Solidity there is a naming convention which decides how a variable name should start:
- For a state variable it would be `s_name`;
- For a immutable variable it would be `i_name`.
- For a constant it would be `NAME`.
However, in the Pot.sol there are variables that are miss named.



```
address[] private i_players; // Should be s_players
uint256[] private i_rewards; // Should be s_rewards
address[] private claimants; // Should be s_claimants
mapping(address => uint256) private playersToRewards; // Should be s_playersToRewards
uint256 private remainingRewards; // Should be s_remainingRewards
uint256 private constant managerCutPercent = 10; // Should be MANAGER_CUT_PERCENT
```

Impact

This can cause confusion to anyone who will read the contract.

Recommended Mitigation

Name each variable correctly, by the convention.