Team 1

DogCoinGame Audit

SMART CONTRACT CODE REVIEW AND ANALYSIS

Prepared By : 0xSenzel

Contact : 0xSenzel@gmail.com

Date : 3 December 2022

TABLE OF CONTENTS

SUMMARY

This audit report is an exercise given by Extropy IO instructor - Laurence Kirk during Encode Club's Solidity Bootcamp as a test of understanding for students to produce an audit on a simple smart contract.

The audit process will focus on:

1) Testing the smart contract on common smart contract attacks.
2) Ensure the codebase is following a good practice of writing smart contract.
3) Manual review on code's vulnerability.
4) Gas optimization on the smart contract.

VULNERABILITY AND RISK LEVEL CATEGORIZATION

Encountered vulnerability and potential exploitable code would be listed and be given recommended actions on addressing the issues. Severity of the vulnerabilities would be categorised according to OWASP.

LEVEL	VALUE	VULNERABILITY	RISK(REQUIRED ACTION)
LOW	0 - 2.9	A vulnerability that can disrupt the contract functioning in a number of scenarios, or creates a risk that the contract may be broken.	Immediate action to reduce risk level.
MEDIUM	3.0 - 5.9	A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.	Implementation of corrective actions in a certain period.
HIGH	6.0 - 9.0	A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.	Implementation of certain corrective actions or accepting the risk.

PROJECT OVERVIEW

DogCoinGame's Description

"DogCoinGame is a game where players are added to the contract via the addPlayer function, they need to send 1 ETH to play. Once 200 players have entered, the UI will be notified by the startPayout event, and will pick 100 winners which will be added to the winners array, the UI will then call the payout function to pay each of the winners. The remaining balance will be kept as profit for the developers."

Project Summary

PROJECT NAME	DogCoinGame
Chain	Ethereum
Language	Solidity
Codebase	https://github.com/ExtropyIO/SolidityBootcamp/blob/main/audit/DogCoinGame.sol

Static Analysis

Slither was used to run analysis over the codebase. Result produced 1 high-severity warning ; 2 medium-severity warning.

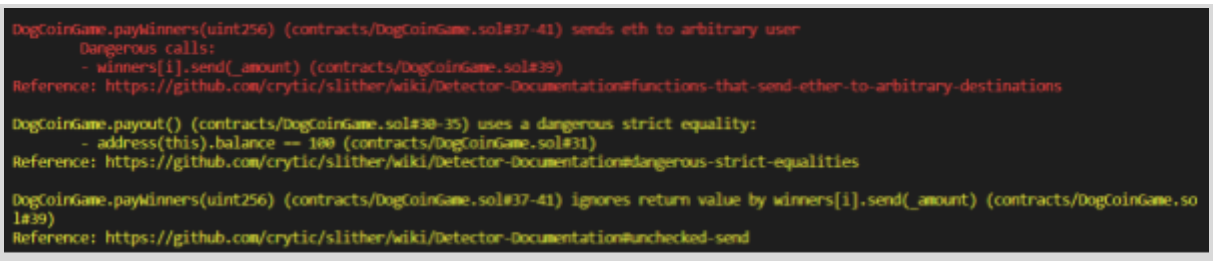

Illustration 1: Compiler warning from Slither

PROJECT METRICS

Inheritance Graph

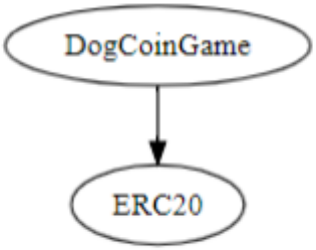

Illustration 2: Inheritance Graph

Call Graph

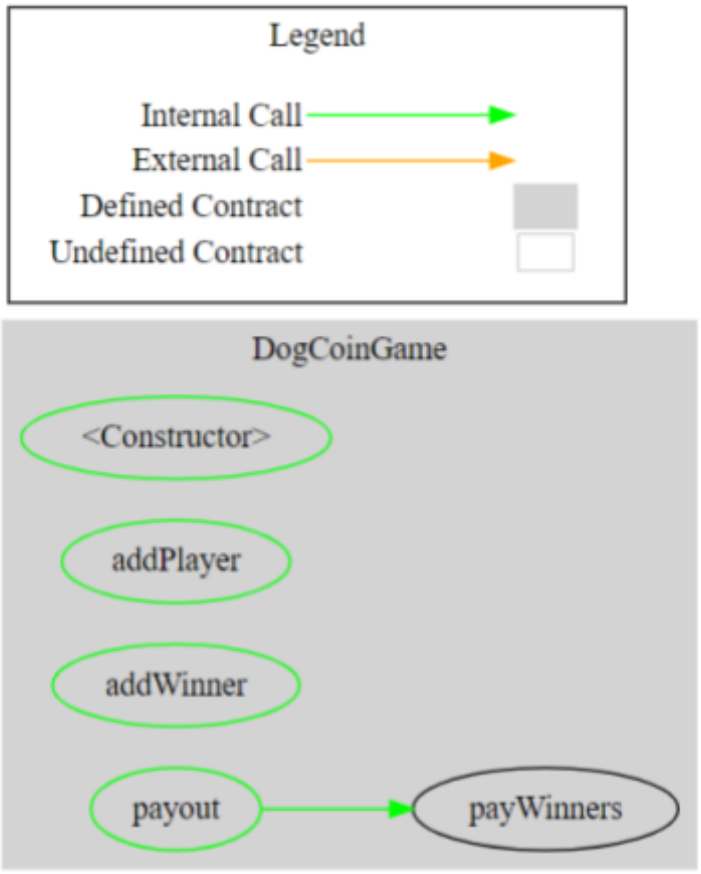

Illustration 3: Call Graph

DETAILED FINDINGS

High-Severity

1. **Unsafe fund transfer**

- payWinners() did not check whether the winners already received the funds. Winner that received the fund can get the winning fund again.

- There is no access control to critical functions such as addWinner(), payout() and payWinners(). Any user can call the function repeatedly until the contract is out of balance.

- payWinners() prone to reentrancy attack.

- payout() function will fail and be unusable as it checks the balance of contract to be an exact number before proceeding.

Mitigation:

- Apply checks-effects-interactions-pattern
- Implement access control such as onlyOwner modifier on functions such as payWinners(), addWinner(), payout().
- payout() 'if' statement should check address(this).balance >= 100 instead of a fixed equal value.
- Add a mapping to keep track of paid user
- payWinners() function visibility can be set as internal.

2. **Multiple Entry**

- addPlayers() did not check if user is registered or new entry, user can enter the contest multiple time to increase odd of winning

- Users can register his entry and call addPlayers() to increase the numberPlayers variable up to 200 to start the game and be the sole player thus winning the game.

- Game is ongoing even if number of players is not met

Mitigation:

- Add mapping to track registered 'players' and 'winners'
- Change 'if' statement to 'require' on addPlayers()
- numberPlayers++ should be executed after require statement
- Implement a checking statement / modifier on addPlayers() to stop the function after the game starts.
- Enable payout() and payWinners() function only after the game has been started.

Medium-Severity

- send() method returns a bool to determine whether a transaction failed or succeeded. If not checked with a ‘require’ statement, the failed transaction will still proceed to send to blockchain so the user will pay for the unnecessary gas.

- Missing receive payable to receive ethers sent to this contract.

Mitigation:

- Replace send() with call() method when sending ether.
- Add receive() external payable {}

Low-Severity

- Contract inherited ERC20 but did not implement any use case of the library; the library can be removed.
- Unused variable ‘currentPrize’ can be removed.