Lottery Audit Report

0xOwain

August 2025

Contents

Lottery Audit Report

Version 1.0

0xOwain

August 24, 2025

Protocol Summary

This is a simple Ethereum lottery smart contract, where a user can enter a lottery by sending a specified amount of Ether to the contract. Once enough participants have entered, the contract owner can choose a random winner who will receive the prize pool of Ether.

Disclaimer

0Owain makes all effort to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the findings provided in this document. A security audit by the team is not an endorsement of the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the Solidity implementation of the contracts.

Risk Classification

Impact Likelihood	High	Medium	Low
High	H	H/M	M
Medium	H/M	M	M/L
Low	M	M/L	L

We use the CodeHawks severity matrix to determine severity. See the documentation for more details.

Audit Details

Scope

Repository: Solidity-Lottery-Contract

Commit Hash 98f354b41af55444a0912c4a828ae352554c47c3

Contracts in Scope: - `Lottery.sol`

cloc Summary Language files blank comment code Solidity 1 7 6 27

Out of Scope:
- N/A (only one Solidity contract present in repo)

Severity Criteria

High - Direct loss of funds or permanent lock of assets.
- Anyone can exploit (not just privileged roles).
- Breaks core protocol functionality.

Medium
- Causes significant disruption (DoS, griefing, governance failure).
- Exploitable under some conditions or requires privileged roles.
- Financial loss is possible but limited.

Low
- Minor issues: inefficiencies, gas waste, unclear logic, small inconsistencies.
- Doesn't threaten core security or funds.

Informational / Non-Critical
- Code style, readability, missing comments.
- Best practices (naming conventions, event emissions, input validation improvements).
- No security impact.

Summary of Findings

Severity	Number of issues found
High	2
Medium	4
Low	2
Informational	2
Gas Optimisations	1
Total	**11**

Tools Used

Manual Review: Line-by-line code analysis of `Lottery.sol`

Testing: Custom JavaScript tests with Web3

Static Analysis: Slither 0.4.17

- Reported weak PRNG, CEI violation, unbounded array growth → all covered in manual findings
- Flagged outdated compiler version → added as Informational finding

High

[H-1] Funds locked if manager is inactive

Description: Only the manager can trigger `pickWinner()`. If the manager becomes inactive, funds remain in the contract indefinitely.

Impact: Players' funds can be permanently stuck.

Proof of Concept: Deploy contract, have multiple players enter, but never call `pickWinner()`. Balance remains locked.

Recommended Mitigation: Add an alternative path (e.g., timeout refunds or permissionless trigger) to ensure funds are always retrievable.

[H-2] Weak randomness

Description: Randomness relies on predictable block variables (`now`, `block.difficulty`) and user-controlled inputs (`players` array).

Impact: Outcome of lottery is not truly random. Miners or strategically entering players may bias the result.

Proof of Concept: Show that the same seed values -> same winner. Simulate late entry or miner timestamp influence in tests.

Recommended Mitigation: Use a commit-reveal scheme or external randomness oracle (e.g., Chainlink VRF).

Medium

[M-1] State update after external call

Description: In `pickWinner()`, the contract transfers funds before resetting the `players` array. This breaks the Checks-Effects-Interactions pattern.

Impact: If the winner is a contract, malicious fallback logic may execute before state is finalised.

Proof of Concept: Deploy a contract with a fallback function as a player, win the lottery, and inspect state before/aftr fallback.

Recommended Mitigation: Always update state before external calls.

[M-2] Division by zero in winner selection

Description: `random() % players.length` will revert if `pickWinner()` is called with no players.

Impact: Causes transaction failure and blocks progress.

Proof of Concept: Call `pickWinner()` before any players enter, the call reverts.

Recommended Mitigation: Add `require(players.lenmgth > 0` before random calculation.

[M-3] No limit on number of entries per address, which can cause bias in winner selection.

Description: The `enter()` function allows the same address to call multiple times.

Impact: A single player could gain disproportionate odds, undermining the fairness of the lottery.

Proof of Concept: Call `enter()` multiple times with the same address; it appears multiple times in `players[]`.

Recommended Mitigation: Enforce uniqueness with a mapping of address -> bool or limit entries per address.

[M-4] Unbounded Players Array Size

Description: There is no maximum limit of players. Each new round resets the array but during a round it may grow arbitarily.

Impact: If the array grows too large, calls to `pickWinner()` may fail due to block gas limits.

Proof of Concept: Simulate thousands of palyers; observe potential gas exhaustion when computing winner.

Recommended Mitigation: Set a max player cap per round to limit gas usage.

Low

[L-1] Unrestricted Ether Contribution Amounts

Description: There is a minimum entry value of 0.01 ether, but no maximum. A user sending excessive ether receives only one slot in the lottery.

Impact: Creates fairness issues and may lead to disproportionate risk of loss for players.

Proof of Concept: Enter with 100 ether and compare against 0.02 ether entry, both get 1 slot.

Recommended Mitigation: Enforce maximum entry amounts, or scale entries by value contributed.

[L-2] Missing events for critical actions

Description: Key function (`enter`, `pickWinner`, reset) do not emit events.

Impact: Transparency and off-chain tracking of participation and winners is reduced.

Proof of Concept: Observe no logs for player joins or winner selection.

Recommended Mitigation: Emit events (e.g., `PlayersEntered`, `WinnerSelected`, `RoundReset`).

Informational

[I-1] Information Disclosure in `getPlayers()`

Description: `getPlayers()` publicly exposes the full list of entrants. While storage is already public on-chain, this makes participant addresses easily accessible.

Impact: May raise privacy concerns for players.

Proof of Concept: Call `getPlayers()` from any account; retrieve full participation list.

Recommended Mitigation: Consider restricting to manager or removing if privacy is desired.

[I-2] Outdated Compiler Version (ˆ0.4.17)

Description: The contract specifies pragma solidity `^0.4.17`, which is an outdated compiler version with multiple known issues (see Solidity security advisories).

Impact: Projects compiled with outdated versions may be exposed to compiler-level bugs and lose compatibility with modern tooling.

Proof of Concept: Slither flagged `^0.4.17` as vulnerable to known historical compiler bugs.

Recommended Mitigation: Upgrade to at least Solidity 0.8.x and refactor for updated syntax and safety checks.

Gas

[G-1] Unnecessary dynamic array resets

Description: `players = new address` clears the array but doesn't refund storage as efficiently as possible.

Impact: Small gas inefficiency per round.

Proof of Concept: Observe gas cost difference between reallocating vs. using `delete players`.

Recommended Mitigation: Use `delete players` to clear array more efficiently.