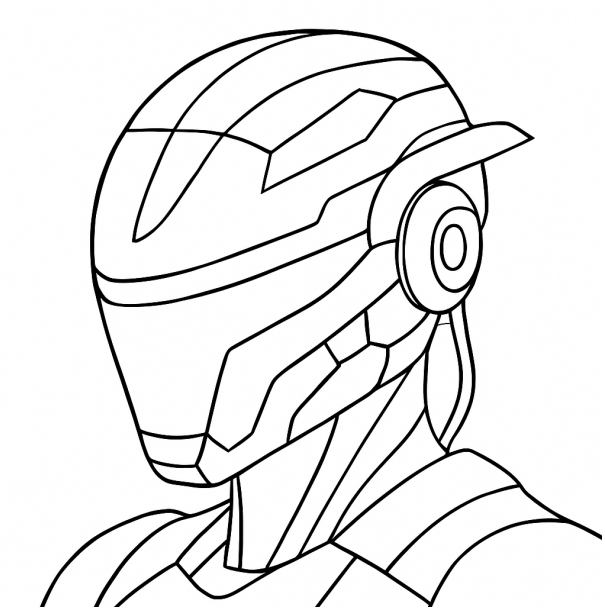

Last Man Standing Game Audit Report

Version 1.0

X: @AlexScherbatyuk

August 07, 2025

Last Man Standing Game Audit Report

Alexander Scherbatyuk

August 07, 2025

Prepared by: X: @AlexScherbatyuk Lead Auditors: - Alexander Scherbatyuk

Table of Contents

Protocol Summary

The Last Man Standing Game is a decentralized “King of the Hill” style game implemented as a Solidity smart contract on the Ethereum Virtual Machine (EVM). It creates a competitive environment where players vie for the title of “King” by paying an increasing fee. The game’s core mechanic revolves around a grace period: if no new player claims the throne before this period expires, the current King wins the entire accumulated prize pot.

Disclaimer

The @AlexScherbatyuk 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		
		High	Medium	Low
	High	H	H/M	M
Likelihood	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

```
src/
--- Game.sol
```

Roles

1. Owner (Deployer)

Powers: * Deploys the `Game` contract. * Can update game parameters: `gracePeriod`, `initialClaimFee`, `feeIncreasePercentage`, `platformFeePercentage`. * Can `resetGame()` to start a new round after a winner has been declared. * Can `withdrawPlatformFees()` accumulated from claims. **Limitations:** * Cannot claim the throne if they are already the current king. * Cannot declare a winner before the grace period expires. * Cannot reset the game if a round is still active.

2. King (Current King)

Powers: * The last player to successfully `claimThrone()`. * Receives a small payout from the next player's `claimFee` (if applicable). * Wins the entire `pot` if no one claims the throne before the `gracePeriod` expires. * Can `withdrawWinnings()` once declared a winner. **Limitations:** * Must pay the current `claimFee` to become king. * Cannot claim the throne if they are already the current king. * Their reign is temporary and can be overthrown by any other player.

3. Players (Claimants)

Powers: * Can `claimThrone()` by sending the required `claimFee`. * Can become the `currentKing`. * Can potentially win the `pot` if they are the last king when the grace period expires. **Limitations:** * Must send sufficient ETH to match or exceed the `claimFee`. * Cannot claim if the game has ended. * Cannot claim if they are already the current king.

4. Anyone (Declarer)

Powers: * Can call `declareWinner()` once the `gracePeriod` has expired. **Limitations:** * Cannot declare a winner if the grace period has not expired. * Cannot declare a winner if no one has ever claimed the throne. * Cannot declare a winner if the game has already ended.

Executive Summary

Issues found

Severity	Number of issues found
High	0
Medium	1
Low	1
Info	0
Gas Optimizations	0
Total	2

Findings

Medium

[M-1] Incorrect require condition prevents throne claim by any player, making the game unplayable.

Description: Any contender who is not a current King and pays an amount that is above or equal to the claimFee should be able to claim the Throne. The claimThrone() function contains a critical logic error where the condition is inverted. The current implementation requires the msg.sender to be the current King which prevents anyone except the current King from claiming the throne. However, since the default value of current King is address(0), this condition completely disables the claiming throne functionality, making the game unplayable.

```
1 function claimThrone() external payable gameNotEnded nonReentrant {
2         require(msg.value >= claimFee, "Game: Insufficient ETH sent to
             claim the throne.");
3 @>      require(msg.sender == currentKing, "Game: You are already the
     king. No need to re-claim.");
4         ...
5     }
```

Impact: The bug triggers with 100% likelihood on the very first throne claim attempt by any player. The throne claiming system is the game's core mechanic instantly and permanently fails, making the entire game unplayable for all players.

Proof of Concept: This test:

verifies that current King is address(0) verifies that claimThrone reverts on any user with address not equal to address(0) verifies that claimThrone allows only address(0) to claim the throne.

```
    function testOnlyAddressZeoCanClaimThrone() public {
        // check if currentKing is address(0)
        address currentKing = game.currentKing();
        assertEq(currentKing, address(0));
        // check no one can claim the throne except address(0)
        vm.prank(player1);
        vm.expectRevert("Game: You are already the king. No need to re-
            claim.");
        game.claimThrone{value: INITIAL_CLAIM_FEE}();
        // check if address(0) can claim the throne
        vm.deal(address(0), INITIAL_CLAIM_FEE);
        vm.prank(address(0));
        game.claimThrone{value: INITIAL_CLAIM_FEE}();

        address newCurrentKing = game.currentKing();
        assertEq(newCurrentKing, address(0));
```

Recommended Mitigation: This mitigation allows any user to claim the throne and prevents the current King from reclaiming the throne.

```
function claimThrone() external payable gameNotEnded nonReentrant {
  require(msg.value >= claimFee, "Game: Insufficient ETH sent to claim
     the throne.");
- require(msg.sender == currentKing, "Game: You are already the king.
    No need to re-claim.");
+ require(msg.sender != currentKing, "Game: You are already the king.
    No need to re-claim.");
  ...
```

Low

[L-1] Incorrect Prize Amount in GameEnded Event Emission

Description: The GameEnded event should emit the actual prize amount that the winner receives when the game ends. According to the event documentation, the prizeAmount parameter represents "The total prize amount won." This should be the accumulated pot value that gets transferred to the winner's pending winnings. The event incorrectly emits pot as the prize amount after it has already been reset to 0, instead of emitting the actual prize amount that was awarded to the winner

```
 function declareWinner() external gameNotEnded {
        require(currentKing != address(0), "Game: No one has claimed
            the throne yet.");
        require(block.timestamp > lastClaimTime + gracePeriod, "Game:
            Grace period has not expired yet."); // Works fine !!! //@>
            need to check sequence of events probably should be block.
            timestamp > (lastClaimTime + gracePeriod)
```

```
4           gameEnded = true;
5           pendingWinnings[currentKing] = pendingWinnings[currentKing] +
                pot;
6   @>          pot = 0; // Reset pot after assigning to winner's pending
        winnings
7   @>          emit GameEnded(currentKing, pot, block.timestamp, gameRound);
         //@> pot is 0 ????
8       }
```

Impact: This is an informational/logging issue that doesn't affect the core protocol functionality or put funds at risk However, external systems, frontends, or analytics tools relying on this event data would receive incorrect information about prize amounts.

Proof of Concept: This test shows that the GameEnded event emits 0 as the prize amount instead of the actual pot value

```
1   function testVulnerability_GameEndedEventEmitsZeroPot() public {
2           // Claim the throne to accumulate pot
3           vm.prank(player1);
4           game.claimThrone{value: INITIAL_CLAIM_FEE}();
5           // Get the pot value before declaring winner
6           uint256 potBeforeDeclare = game.pot();
7           uint256 lastClaimTime = game.lastClaimTime();
8           uint256 gracePeriod = game.gracePeriod();
9           uint256 gameRound = game.gameRound();
10          address currentKing = game.currentKing();
11          // Verify pot has accumulated value (should be 95% of claim fee
                after platform fee)
12          uint256 expectedPot = (INITIAL_CLAIM_FEE * 95) / 100; // 95%
                goes to pot (5% platform fee)
13          assertEq(potBeforeDeclare, expectedPot, "Pot should contain
                expected amount before declaring winner");
14          assertGt(potBeforeDeclare, 0, "Pot should be greater than 0
                before declaring winner");
15          // Warp time past grace period
16          vm.warp(lastClaimTime + gracePeriod + 1 hours);
17          // Expect the event to emit 0 as prize amount (demonstrating
                the vulnerability)
18          vm.expectEmit(true, false, false, false);
19          emit GameEnded(currentKing, 0, block.timestamp, gameRound);
20          // Declare winner - this will emit the event with pot = 0
21          game.declareWinner();
22          // Verify the vulnerability: pot is now 0 after declaring
                winner
23          uint256 potAfterDeclare = game.pot();
24          assertEq(potAfterDeclare, 0, "Pot should be 0 after declaring
                winner");
25          // Verify the winner received the correct amount in pending
                winnings
26          uint256 winnerPendingWinnings = game.pendingWinnings(
```

```
            currentKing);
27          assertEq(winnerPendingWinnings, expectedPot, "Winner should
            have correct amount in pending winnings");
28          // The vulnerability: Event emitted 0 as prize amount instead
            of the actual prize (expectedPot)
29          // This demonstrates that external systems listening to this
            event would receive incorrect information
30      }
```

Recommended Mitigation: The code correctly captures prizeAmount before resetting pot. The event emits the correct prize amount.

```
1       function declareWinner() external gameNotEnded {
2           require(currentKing != address(0), "Game: No one has claimed
            the throne yet.");
3           require(block.timestamp > lastClaimTime + gracePeriod, "Game:
            Grace period has not expired yet.");
4           gameEnded = true;
5   +       uint256 prizeAmount = pot;
6           pendingWinnings[currentKing] = pendingWinnings[currentKing] +
            pot;
7           pot = 0; // Reset pot after assigning to winner's pending
            winnings
8   -       emit GameEnded(currentKing, 0, block.timestamp, gameRound)
9   +       emit GameEnded(currentKing, prizeAmount, block.timestamp,
        gameRound)
10
11      }
```