Last Man Standing Game Security Review

Reviewer: Muhammad Faran **Date:** July 27, 2025 **Repo:** CodeHawks-Contests/2025-07-last-man-standing

Table of Contents

Disclaimer

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

Risk-Classification

The severity of each finding combines **Impact** (the damage if exploited) and **Likelihood** (the chance of exploitation). The matrix below illustrates how these two axes map to a single severity label:

Likelihood ↓ / Impact →	High	Medium	Low
High	High	High/Medium	Medium
Medium	High/Medium	Medium	Medium-Low
Low	Medium	Medium-Low	Low

Executive Summary

Severity	Number Of Issues Found
High	2
Medium	2
Low	3
Informational	2

Audit Details

Scope

```
src/
├── BeatToken.sol
├── FestivalPass.sol
├── interfaces
│   ├── IFestivalPass.sol
```

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.

Actors

This protocol includes the following 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.

[H-1] Overpayment on Throne Claim leads to Unexpected User Experience

Description

- Normally, a player should only be able to claim the throne by sending exactly the required `claimFee` amount.
- The current implementation allows users to send more than the required `claimFee`, resulting in overpayment and potential confusion.

```
function claimThrone() external payable gameNotEnded nonReentrant {
    //@audit: Takes more than claim fee, which is not expected. Better approach could be msg.value == claimFee.
@>  require(msg.value >= claimFee, "Game: Insufficient ETH sent to claim the throne.");
    //...
}
```

Risk

Likelihood:

- This will occur whenever a user mistakenly sends more ETH than the required claimFee.

Impact:

- Users may lose funds by overpaying, as the excess is not refunded.
- This can lead to a poor user experience and loss of trust in the contract.

Proof of Concept

A user sends 2x the required claimFee when claiming the throne. The contract accepts the transaction, but the extra ETH is not refunded instead it goes into the `pot`.

Proof Of Code

- Paste the following code into `Game.t.sol` and see that; overpayed `claimFee` is added into the pot and platform also benefits from it but the user is at a disadvantage as the user payed 2X the amount for no benefit.

```
function testOverPaymentPOC() public {
        console2.log("Initial claim fee:", game.claimFee());
        vm.startPrank(player1);
        game.claimThrone{value: 0.2 ether}();
        console2.log("After Player1 claims throne with 2X claim fee:");
        console2.log("  pot:", game.pot());
        console2.log("  platformFeesBalance:", game.platformFeesBalance());
        console2.log("  claimFee (next):", game.claimFee());
        vm.stopPrank();
    }
```

- The following are the logs of the above POC:

```
Logs:
  Initial claim fee: 100000000000000000
  After Player1 claims throne with 2X claim fee:
    pot: 190000000000000000
    platformFeesBalance: 10000000000000000
    claimFee (next): 110000000000000000
```

Recommended Mitigation

Require the exact amount to be sent for claiming the throne.

```
- require(msg.value >= claimFee, "Game: Insufficient ETH sent to claim the throne.");
+ require(msg.value == claimFee, "Game: Must send exact claim fee.");
```

[H-2] Forgotten Winner Declaration Allows Late Claims which Undermine's Game Finality

Description

- Normally, after the `gracePeriod` expires, the last king should be declared the winner and the game should end, preventing further claims.
- In the current implementation, if no one calls `declareWinner()` after the grace period, the game remains active, allowing a new player to claim the throne even after the original king should have won.

Risk

Likelihood:

- This will occur whenever the grace period expires and no one calls `declareWinner()`.
- Any user can claim the throne after the grace period, even though the previous king should have won.

Impact:

- The rightful winner may lose their prize if someone else claims the throne after the grace period.
- Undermines the finality and fairness of the game, leading to loss of trust.

Proof Of Concept

- After the grace period expires, no one calls declareWinner(). Another player is able to claim the throne, resetting the timer and stealing the win.

Proof Of Code

```
function testWinnerDeclarationForgotten() public {
        vm.startPrank(deployer);
        game.claimThrone{value: game.claimFee()}();
        vm.warp(block.timestamp + GRACE_PERIOD + 1);
        vm.stopPrank();

        vm.startPrank(player1);
        game.claimThrone{value: game.claimFee()}();
        vm.stopPrank();
    }
```

Recommended Mitigation

- Automatically prevent new claims after the grace period has expired, or require the contract to check for grace period expiry in `claimThrone()` and end the game if needed.

```
function claimThrone() external payable gameNotEnded nonReentrant {
+   require(block.timestamp <= lastClaimTime + gracePeriod, "Game: Grace period expired, cannot claim.");
    // ...existing code...
}
```

[M-1] Deployer of the Game Cannot Reset Without Claims

Description

- Normally, the game should be able to end and reset even if no one has claimed the throne.
- The current implementation requires at least one claim before the game can end, otherwise gameEnded remains false and the owner cannot reset the game.
- The game doesnot end with any `GRACEPERIOD` expire till someone doesnot claim the throne.
- The game cant be reset even if there is a claim throne, it resets when the winner is declared.

Risk

Likelihood:

- This will occur whenever a new game round starts and no one claims the throne.
- The contract will be stuck in a state where it cannot be reset.

Impact:

- The owner cannot reset the game, and the contract becomes unusable until a claim is made.
- Funds and game logic are locked, requiring a forced claim to proceed.

Proof of Concept

- Start a new game round and do not call claimThrone. Attempt to resetGame.

Proof Of Code

- Add the following code in `Game.t.sol` and see...

```
function testResetGameFails() public {
        vm.startPrank(deployer);

        // Simulate the passage of time
        vm.warp(block.timestamp + GRACE_PERIOD + 1);

        // Fails, game not ended
        assertTrue(game.gameEnded(), "Game should have ended after grace period");

        // Comment the above assert and see reset Game fails, deployer cant reset the game
        game.resetGame();

        vm.stopPrank();
    }
```

- The game only resets when someone claims the throne and that someone is declared a winner.

```
    function testResetGamePasses() public {
        vm.startPrank(deployer);
        // Claim the Throne
        game.claimThrone{value: game.claimFee()}();

        // Simulate the passage of time
        vm.warp(block.timestamp + GRACE_PERIOD + 1);

        // Declare winner
        game.declareWinner();

        // Passes
        assertTrue(game.gameEnded(), "Game should have ended after grace period");

        // Passes
        game.resetGame();

        vm.stopPrank();
    }
```

Recommended Mitigation

Allow the owner to reset the game even if no claims have been made, or provide a function to force-reset in such cases.

```
 - require(currentKing != address(0), "Game: No one has claimed the throne yet.");
+ if (currentKing == address(0)) {
+     gameEnded = true;
+     emit GameEnded(address(0), 0, block.timestamp, gameRound);
+     return;
+ }
```

[M-2] Backward Logic in Claim Throne; Hence No One can claim the Throne

Description

- Normally, the contract should **prevent the current king from reclaiming the throne**, ensuring that **only new players** can claim, but this current logic is bakward.
- Everytime a user tries to claim the throne, the check happens which checks that msg.sender == currentKing where `currentKing` is address(0) but user address != address(0) hence no one can claim the throne.

```
require(msg.sender == currentKing, "Game: You are already the king. No need to re-claim.");
```

- This only allows the **current king** to claim again effectively, which is an address(0) by the way; **blocking all other players** from claiming.
- This is the **opposite** of the intended behavior and would **break the game flow** if enabled.

Risk

Likelihood:

- This bug will occur **whenever the check is uncommented** or implemented as shown.
- **Any attempt by a new player** to claim the throne will **revert**, making the game **unplayable**.

Impact:

- **Only the current king** can claim again.
- **All new claims** will revert, effectively **freezing the game**.
- The **game logic is fundamentally broken**, preventing normal gameplay.

Proof of Concept

- Paste this function in the Test File and see...

```
    function testClaimThrone() public {
        vm.startPrank(player1);
        game.claimThrone{value: game.claimFee()}();
        console2.log("Player1 claimed throne successfully.");
        vm.stopPrank();
    }
```

- You will get these logs

```
    [FAIL: Game: You are already the king. No need to re-claim.]
```

Recommended Mitigation

Reverse the logic so that the **current king is prevented** from reclaiming the throne, and **new players are allowed**:

```
 - 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.");
```

This change enforces the intended access control logic, allowing only new participants to challenge the throne.

[L-1] Precision Loss in Fee Calculations; Pot and Fee Distribution

Description

- Normally, fee calculations should be as precise as possible to avoid loss of funds due to rounding errors.
- The current implementation uses integer division for percentage calculations, which can lead to precision loss, especially for small values.

```
 // Root cause in the codebase with @> marks to highlight the relevant section
function claimThrone() external payable gameNotEnded nonReentrant {
    //...
@>  currentPlatformFee = (sentAmount * platformFeePercentage) / 100;
    //...
@>  claimFee = claimFee + (claimFee * feeIncreasePercentage) / 100;
}
```

Risk

Likelihood:

- This will occur on every claim, especially when claim fees are small or percentages are not multiples of 10.
- The effect is more pronounced as the game progresses and claim fees increase.

Impact:

- Small amounts of ETH may be lost due to rounding down, reducing the pot and platform fees over time.
- Players and the platform owner may receive less than expected.

Proof of Concept

Here are some loss scenario's.

```
 // No loss scenario
sentAmount = 0.1 ETH = 100000000000000000 wei
platformFee = (100000000000000000 * 5) / 100 = 5000000000000000 wei
// Perfect division - no loss!

// Negligible Loss
sentAmount = 100000000000000001 wei (0.1 ETH + 1 wei)
platformFee = (100000000000000001 * 5) / 100 = 5000000000000000 wei
// Lost: 5 wei out of 0.1 ETH = negligible

// Loss scenario
sentAmount = 999 wei
platformFee = (999 * 5) / 100 = 49 wei   // Lost: 4995 - 4900 = 95 wei

// Small amounts
sentAmount = 99 wei
platformFeePercentage = 5%
currentPlatformFee = (99 * 5) / 100 = 495 / 100 = 4 wei
// Lost: 495 - 400 = 95 wei (truncated)

// Fee increases
claimFee = 133 wei
feeIncreasePercentage = 10%
increase = (133 * 10) / 100 = 1330 / 100 = 13 wei
// Lost: 1330 - 1300 = 30 wei (truncated)
```

Recommended Mitigation

Consider using a higher-precision base (e.g., basis points or 1e4) for percentages, or document the rounding behavior clearly.

```
 - currentPlatformFee = (sentAmount * platformFeePercentage) / 100;
+ currentPlatformFee = (sentAmount * platformFeePercentage) / 10000; // if using basis points
```

[L-2] Prize Amount in Event Is Always Zero emits Incorrect Event Data

Description

- Normally, the GameEnded event should emit the actual prize amount won by the winner.
- The current implementation sets pot = 0 before emitting the event, so the event always shows a prize of zero.

```
 // Root cause in the codebase with @> marks to highlight the relevant section
function declareWinner() external gameNotEnded {
    //...
    pendingWinnings[currentKing] = pendingWinnings[currentKing] + pot;
@>  pot = 0; // Reset pot after assigning to winner's pending winnings
    //@audit pot = 0, and then you emitted the event so it would show prize won as 0.
@>  emit GameEnded(currentKing, pot, block.timestamp, gameRound);
}
```

Risk

Likelihood:

- This will occur every time a winner is declared.
- The event log will always show a prize of zero.

Impact:

- Off-chain systems and users relying on event logs will see incorrect prize information.
- Reduces transparency and trust in the contract.

Proof of Concept

- Declare a winner and check the emitted GameEnded event; the prize amount is always zero.

Proof Of Code

```
 game.claimThrone{value: game.claimFee()}();
vm.warp(block.timestamp + game.gracePeriod() + 1);
vm.expectEmit(true, false, false, true);
emit GameEnded(game.currentKing(), game.pot(), block.timestamp, game.gameRound());
game.declareWinner();
```

Recommended Mitigation

Emit the event with the correct prize amount before resetting the pot.

```
 - pot = 0; // Reset pot after assigning to winner's pending winnings
- emit GameEnded(currentKing, pot, block.timestamp, gameRound);
+ emit GameEnded(currentKing, pot, block.timestamp, gameRound);
+ pot = 0; // Reset pot after assigning to winner's pending winnings
```

[L-3] ETH Sent Directly to Contract Is Trapped + Loss of User Funds

Description

- Normally, any ETH sent to the contract should be recoverable or used in the game logic.
- The current implementation allows ETH to be sent directly to the contract via the receive() function, but this ETH is not accessible by any withdrawal function and is not added to the pot or platform fees.

```
 // Root cause in the codebase with @> marks to highlight the relevant section
//@audit: This function is used to receive ETH directly into the contract. Which traps any ETH sent to the contract address.
@> receive() external payable {}
```

Risk

Likelihood:

- This will occur whenever a user mistakenly sends ETH directly to the contract address.
- Wallets and dApps may send ETH directly due to user error or UI bugs.

Impact:

- ETH is permanently trapped in the contract and cannot be withdrawn by anyone.
- Users lose funds, and the contract accumulates unusable ETH.

Proof of Concept

- Send ETH directly to the contract and attempt to withdraw it via any contract function; the ETH remains trapped.
- This shows unusual increase in contract balance due to trapped ETH.

Proof Of Code

```
function testEthTrappedProof() public {
        uint256 sendAmount = 1 ether;
        vm.startPrank(player1);
        (bool sent,) = address(game).call{value: sendAmount}("");
        require(sent, "Direct ETH send failed");
        vm.stopPrank();

        vm.startPrank(player2);
        game.claimThrone{value: game.claimFee()}();
        vm.warp(block.timestamp + GRACE_PERIOD + 1);
        game.declareWinner();
        game.withdrawWinnings();
        vm.stopPrank();

        vm.startPrank(deployer);
        console2.log("Contract Balance:", game.getContractBalance());
        game.withdrawPlatformFees();
        console2.log("Contract Balance:", game.getContractBalance());
        vm.stopPrank();
    }
```

Recommended Mitigation

- Either remove the receive() function or add logic to allow claimThrone on those funds.

```
 - receive() external payable {}
+ receive() external payable {
+     claimThrone();
+ }
```

[I-1] Owner Can (Even Unintentionally) Manipulate Grace Period and Claim Fee Mid-Round leading to Unfair Gameplay

Description

- Normally, game parameters such as the grace period and claim fee should remain constant during a round to ensure fairness.
- The current implementation allows the owner to update these parameters mid-round, which can be used to indirectly influence the game's outcome.
- In docs it is mentioned that Owner cannot declare a winner before the grace period expires, but in this case although not directly but indirectly it can be done by updating the grace period, eg:- Shortening the grace Period.
- Similarly updating claim fee, mid round can restrict players from claiming the throne, eg;- making it too high.

```
 // Root cause in the codebase with @> marks to highlight the relevant section
function updateGracePeriod(uint256 _newGracePeriod) external onlyOwner {
@>  gracePeriod = _newGracePeriod;
    //...
}
function updateClaimFeeParameters(uint256 _newInitialClaimFee, uint256 _newFeeIncreasePercentage) external onlyOwner isValidPercenta
@>  initialClaimFee = _newInitialClaimFee;
@>  feeIncreasePercentage = _newFeeIncreasePercentage;
    //...
}
```

Risk

Likelihood:

- This will occur whenever the owner chooses to change parameters during an active round.
- There is no restriction to prevent mid-round changes.

Impact:

- The owner can make the game unwinnable or unfair (e.g., set claim fee to an unreasonably high value).
- Indirect manipulation of grace period timing could unfairly favor the owner or selected players.

Proof of Concept

Owner calls `updateGracePeriod()` or `updateClaimFeeParameters()` during a round to make it impossible or impractical for others to play.

Proof Of Code

```
    // Example of Deployer keep on increasing Grace Period not allowing anyone to win
    function testUpdatingGracePeriodMidRound() public {
        vm.startPrank(deployer);
        game.claimThrone{value: game.claimFee()}();
        _upgradeGraceperiod(2 days);
        vm.warp(block.timestamp + GRACE_PERIOD + 1);
        game.declareWinner();
        vm.stopPrank();
    }

    // Example of Deployer reducing Grace Period allowing some one to win
    function testUpdatingGracePeriodMidRoundTwo() public {
        vm.startPrank(deployer);
        game.claimThrone{value: game.claimFee()}();
        vm.warp(block.timestamp + 2 hours);
        _upgradeGraceperiod(1 hours);
        game.declareWinner();
        vm.stopPrank();
    }
```

Recommended Mitigation

Restrict grace period and claim fee updates to only be allowed between rounds (i.e., when the game has ended).

```
 - function updateGracePeriod(uint256 _newGracePeriod) external onlyOwner {
+ function updateGracePeriod(uint256 _newGracePeriod) external onlyOwner gameEndedOnly {

- function updateClaimFeeParameters(...) external onlyOwner {
+ function updateClaimFeeParameters(...) external onlyOwner gameEndedOnly {
```

[I-2] Owner Can Set Platform Fee to 100%; Complete Fund Drain

Description

- Normally, platform fees should be capped at a reasonable percentage to ensure fair distribution of funds between players and the platform.
- The current implementation allows the owner to set the platform fee up to 100%, effectively draining all funds from the game.

```
 // Root cause in the codebase with @> marks to highlight the relevant section
//@audit: Centralization Issue:- Increasing Platform fee percentage upto 100%. These are giving serious centralization powers to the
function updatePlatformFeePercentage(uint256 _newPlatformFeePercentage) external onlyOwner isValidPercentage(_newPlatformFeePercenta
@>  platformFeePercentage = _newPlatformFeePercentage;
    //...
}
```

Risk

Likelihood:

- This will occur whenever the owner updates the platform fee during a round.
- There is no restriction or cap on the percentage that can be set.

Impact:

- The owner can set the platform fee to 100%, draining all ETH that would normally go into the pot or to players.
- This breaks game integrity and introduces a critical centralization risk.

Proof of Concept

Owner calls `updatePlatformFeePercentage(100)` during a round, claims the throne, and receives all funds as platform fees.

Proof Of Code

```
 vm.startPrank(deployer);
game.updatePlatformFeePercentage(100);
game.claimThrone{value: game.claimFee()}();
vm.stopPrank();
```

Recommended Mitigation

Restrict platform fee percentage updates to occur only between rounds and enforce a maximum cap (e.g., 20%).

```
 - function updatePlatformFeePercentage(...) external onlyOwner isValidPercentage(...) {
+ function updatePlatformFeePercentage(...) external onlyOwner gameEndedOnly isValidPercentage(...) {
```

Additionally, validate the input:

```
require(_newPlatformFeePercentage <= 20, "Fee too high");
```