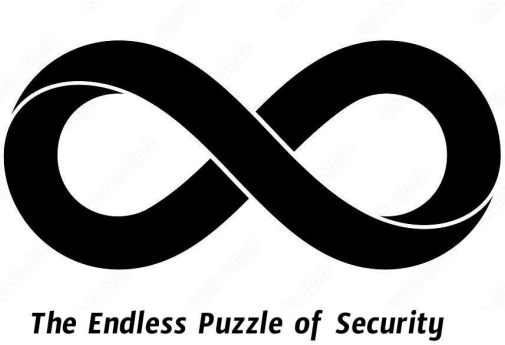

Protocol Audit Report

Version 1.0

Muhammad Kashif

October 16, 2025

Protocol Audit Report

Muhammad Kashif

October 16, 2025

Prepared by: Muhammad Kashif

Security Researcher: Muhammad Kashif

Table of Contents

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

I 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 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
Likelihood	High	H	H/M	M
	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

Github Codebase Link

```
https://github.com/CodeHawks-Contests/2025-10-raisebox-faucet.git
```

Scope

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

Roles and Actors

There are basically 3 actors in this protocol:

1. Owner:

RESPONSIBILITIES:

- deploys contract,
- mint initial supply and any new token in future,
- can burn tokens,
- can adjust daily claim limit,
- can refill sepolia eth balance

LIMITATIONS:

- cannot claimfaucet tokens

2. Claimer:

RESPONSIBILITIES:

- can claim tokens by calling the claimFaucetTokens function of this contract.

LIMITATIONS:

- Doesn't have any owner defined rights above.

3. Donators:

RESPONSIBILITIES:

- can donate sepolia eth directly to contract

Issues found

Severity	No of issues found
High	2
Medium	1
Low	3
Info	0
Total	0

Findings

High

[H-1] Daily Limit Check Before Reset Causes Denial Of Service In Protocol

Description

- The function `claimFaucetTokens()` should reset `dailyClaimCount` before checking `dailyClaimLimit`, ensuring that `dailyClaimCount` properly resets after one day.
- The `dailyClaimCount` resets happens after the `dailyClaimLimit` check, causing permanent lockout once limit is reached with in 1 day (24 hours).

```
1   if (block.timestamp > lastFaucetDripDay + 1 days) {
2               lastFaucetDripDay = block.timestamp;
3 @>            dailyClaimCount = 0;
4         }
```

Risk

Likelihood:

- Once `dailyClaimCount` reaches `dailyClaimLimit` with in 24 hours.

Impact:

- Once `dailyClaimCount` reaches `dailyClaimLimit`, the function will always revert forever for entirely new faucet claimers, even after 24 hours or 3 days.

Proof of Concept

To test this, include the following code in the `RaiseBoxFaucetTest.t.sol` file:

Proof Of Code

```
function test_DenialOfServiceAfterHundredUserClaimedFaucetWithInOneDay
    () public {
        uint256 numberOfUser = 100;

        for (uint160 i = 1; i <= numberOfUser; i++) {
            address user = address(i+100);
            vm.startPrank(user);
            raiseBoxFaucet.claimFaucetTokens();
            uint256 userBalance = raiseBoxFaucet.balanceOf(user);
            assertEq(userBalance,raiseBoxFaucet.faucetDrip());
            vm.stopPrank();
        }

        // After 100th user claimed withIn 24 Hour, The protocol
           completemely block the drips of token and sepolia Eth indeed
            the cooldown period had been passed, due to the
           dailyclaimlimit has not been reset.
        advanceBlockTime(block.timestamp + 3 days);
        vm.roll(200);
        address user101 = makeAddr("USER-101");
        vm.startPrank(user101);
        vm.expectRevert();
        raiseBoxFaucet.claimFaucetTokens();
    }
```

Recommended Mitigation

In order to mitigate the DOS, move `dailyClaimCount` reset logic before `dailyClaimLimit` logic checks.

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

        // (lastClaimTime[faucetClaimer] == 0);

        if (block.timestamp < (lastClaimTime[faucetClaimer] +
           CLAIM_COOLDOWN)) {
```

```
8                 revert RaiseBoxFaucet_ClaimCooldownOn();
9             }
10
11            if (faucetClaimer == address(0) || faucetClaimer == address(
                  this) || faucetClaimer == Ownable.owner()) {
12                revert
                      RaiseBoxFaucet_OwnerOrZeroOrContractAddressCannotCallClaim
                      ();
13            }
14
15            if (balanceOf(address(this)) <= faucetDrip) {
16                revert RaiseBoxFaucet_InsufficientContractBalance();
17            }
18
19  +         if (block.timestamp > lastFaucetDripDay + 1 days) {
20  +             lastFaucetDripDay = block.timestamp;
21  +             dailyClaimCount = 0;
22  +         }
23
24            if (dailyClaimCount >= dailyClaimLimit) {
25                revert RaiseBoxFaucet_DailyClaimLimitReached();
26            }
27
28            // drip sepolia eth to first time claimers if supply hasn't ran
                  out or sepolia drip not paused**
29            // still checks
30            if (!hasClaimedEth[faucetClaimer] && !sepEthDripsPaused) {
31                uint256 currentDay = block.timestamp / 24 hours;
32
33                if (currentDay > lastDripDay) {
34                    lastDripDay = currentDay;
35                    dailyDrips = 0;
36                    // dailyClaimCount = 0;
37                }
38
39                if (dailyDrips + sepEthAmountToDrip <= dailySepEthCap &&
                      address(this).balance >= sepEthAmountToDrip) {
40                    hasClaimedEth[faucetClaimer] = true;
41                    dailyDrips += sepEthAmountToDrip;
42
43                    (bool success,) = faucetClaimer.call{value:
                          sepEthAmountToDrip}("");
44
45                    if (success) {
46                        emit SepEthDripped(faucetClaimer,
                              sepEthAmountToDrip);
47                    } else {
48                        revert RaiseBoxFaucet_EthTransferFailed();
49                    }
50                } else {
51                    emit SepEthDripSkipped(
```

```
52                      faucetClaimer,
53                      address(this).balance < sepEthAmountToDrip ? "
                            Faucet out of ETH" : "Daily ETH cap reached"
54                  );
55              }
56          } else {
57              dailyDrips = 0;
58          }
59
60          /**
61           *
62           * @param lastFaucetDripDay tracks the last day a claim was
                 made
63           * @notice resets the @param dailyClaimCount every 24 hours
64           */
65  -       if (block.timestamp > lastFaucetDripDay + 1 days) {
66  -           lastFaucetDripDay = block.timestamp;
67  -           dailyClaimCount = 0;
68  -       }
69
70          // Effects
71
72          lastClaimTime[faucetClaimer] = block.timestamp;
73          dailyClaimCount++;
74
75          // Interactions
76          _transfer(address(this), faucetClaimer, faucetDrip);
77
78          emit Claimed(msg.sender, faucetDrip);
79      }
```

[H-2] Lack of Following CEI Pattern Leads to Reentrancy for Double Faucet Token Claims

Description

- The function `claimFaucetTokens()` should follow the CEI (Checks-Effects-Interactions) pattern to defend against reentrancy. The `lastClaimTime` variable should be marked to block.timestamp before making any external calls.
- The function `claimFaucetTokens()` makes an external call to transfer the `sepEthAmountToDrip` native ETH to the `faucetClaimer` before updating `lastClaimTime` to `block.timestamp`, allowing the caller to reenter the contract and claim `faucetDrip` twice.

```
1           // Effects
```

```
2
3 @>        lastClaimTime[faucetClaimer] = block.timestamp;
4           dailyClaimCount++;
```

Risk

Likelihood:

- For every new `faucetClaimer` can claim `faucetDrip` twice.

Impact:

- The vulnerability causes the faucet to cost nearly double for the protocol to maintain due to double unauthorized `faucetDrip` claims.

Proof of Concept

To test this, include the following code in the `RaiseBoxFaucet.t.sol` file:

Proof Of Code

```
1  function test_ReentrancyFor2xToken() public {
2          Exploit exploit = new Exploit(raiseBoxFaucetContractAddress);
3
4          exploit.claim2xTokens();
5          uint256 sepoliaEthReceivedByAttacker = address(exploit).balance
               ;
6          uint256 tokenReceivedByAttacker = raiseBoxFaucet.balanceOf(
               address(exploit));
7
8          assertEq(sepoliaEthReceivedByAttacker, raiseBoxFaucet.
               sepEthAmountToDrip(), "Sepolia Eth hasn't transferred");
9          assertEq(tokenReceivedByAttacker,(raiseBoxFaucet.faucetDrip() *
                2 ), "Token not received");
10     }
11
12 }
13
14
15 contract Exploit {
16
17     RaiseBoxFaucet raiseBoxFaucet;
18
19     constructor (address _raiseBoxFaucet) {
20         raiseBoxFaucet = RaiseBoxFaucet(payable(_raiseBoxFaucet));
21     }
22
```

```
23      function claim2xTokens() public  {
24          raiseBoxFaucet.claimFaucetTokens();
25      }
26
27      fallback() external payable {
28          raiseBoxFaucet.claimFaucetTokens();
29      }
30  }
```

Recommended Mitigation

To defend against reentrancy, strictly follow the CEI pattern in the `claimFaucetTokens()` function

```
1   function claimFaucetTokens() public {
2           // Checks
3           faucetClaimer = msg.sender;
4
5           // (lastClaimTime[faucetClaimer] == 0);
6
7           if (block.timestamp < (lastClaimTime[faucetClaimer] +
               CLAIM_COOLDOWN)) {
8               revert RaiseBoxFaucet_ClaimCooldownOn();
9           }
10
11          if (faucetClaimer == address(0) || faucetClaimer == address(
               this) || faucetClaimer == Ownable.owner()) {
12              revert
                   RaiseBoxFaucet_OwnerOrZeroOrContractAddressCannotCallClaim
                   ();
13          }
14
15          if (balanceOf(address(this)) <= faucetDrip) {
16              revert RaiseBoxFaucet_InsufficientContractBalance();
17          }
18
19          if (dailyClaimCount >= dailyClaimLimit) {
20              revert RaiseBoxFaucet_DailyClaimLimitReached();
21          }
22
23  +       // Effects
24
25  +       lastClaimTime[faucetClaimer] = block.timestamp;
26  +       dailyClaimCount++;
27
28          // drip sepolia eth to first time claimers if supply hasn't ran
                out or sepolia drip not paused**
29          // still checks
30          if (!hasClaimedEth[faucetClaimer] && !sepEthDripsPaused) {
```

```
31              uint256 currentDay = block.timestamp / 24 hours;
32
33              if (currentDay > lastDripDay) {
34                  lastDripDay = currentDay;
35                  dailyDrips = 0;
36                  // dailyClaimCount = 0;
37              }
38
39              if (dailyDrips + sepEthAmountToDrip <= dailySepEthCap &&
                    address(this).balance >= sepEthAmountToDrip) {
40                  hasClaimedEth[faucetClaimer] = true;
41                  dailyDrips += sepEthAmountToDrip;
42
43                  (bool success,) = faucetClaimer.call{value:
                        sepEthAmountToDrip}("");
44
45                  if (success) {
46                      emit SepEthDripped(faucetClaimer,
                            sepEthAmountToDrip);
47                  } else {
48                      revert RaiseBoxFaucet_EthTransferFailed();
49                  }
50              } else {
51                  emit SepEthDripSkipped(
52                      faucetClaimer,
53                      address(this).balance < sepEthAmountToDrip ? "
                            Faucet out of ETH" : "Daily ETH cap reached"
54                  );
55              }
56          } else {
57              dailyDrips = 0;
58          }
59
60          /**
61           *
62           * @param lastFaucetDripDay tracks the last day a claim was
                 made
63           * @notice resets the @param dailyClaimCount every 24 hours
64           */
65          if (block.timestamp > lastFaucetDripDay + 1 days) {
66              lastFaucetDripDay = block.timestamp;
67              dailyClaimCount = 0;
68          }
69
70  -       // Effects
71
72  -       lastClaimTime[faucetClaimer] = block.timestamp;
73  -       dailyClaimCount++;
74
75          // Interactions
76          _transfer(address(this), faucetClaimer, faucetDrip);
```

```
77
78          emit Claimed(msg.sender, faucetDrip);
79      }
```

Medium

[M-1] Full-Balance Transfer Prior to Burn Causes Faucet Drain

Description

- The `burnFaucetTokens()` function should only transfer and burn the specified amount `amountToBurn` of faucet tokens from the contract's own balance without transferring all tokens to the owner.
- The `burnFaucetTokens()` function transfer the full contract balance to owner instead of transferring `amountToBurn`.

```
1  function burnFaucetTokens(uint256 amountToBurn) public onlyOwner {
2          require(amountToBurn <= balanceOf(address(this)), "Faucet Token
               Balance: Insufficient");
3
4          // transfer faucet balance to owner first before burning
5          // ensures owner has a balance before _burn (owner only
               function) can be called successfully
6  @>      _transfer(address(this), msg.sender, balanceOf(address(this)));
7
8          _burn(msg.sender, amountToBurn);
9      }
```

Risk

Likelihood:

- The defined flawed logic **always transfers the entire balance** from the contract to the owner before burning, regardless of `amountToBurn`.

Impact:

- This drains all tokens from the contract, which can **disrupt core functionalities** that depend on the contract's token balance (e.g., faucet claims). Users can't claim faucet tokens until the `mintFaucetTokens()` called again.

Proof of Concept

To test this, include the following code in the `RaiseBoxFaucetTest.t.sol` file:

Proof Of Code

```
function test_BurnFunctionCanBurnTheSpecifiedAmount() public {
        uint256 burnAmount = 2e18;

        assertTrue(
            raiseBoxFaucet.getFaucetTotalSupply() ==
               INITIAL_SUPPLY_MINTED,
            "Total supply should be equal to Intial supply minted"
        );

        vm.prank(owner);
        raiseBoxFaucet.burnFaucetTokens(burnAmount);
        assertEq(raiseBoxFaucet.getFaucetTotalSupply(),
           INITIAL_SUPPLY_MINTED - burnAmount );

    }
```

Recommended Mitigation

To mitigate the flaw, transfer and burn only the specified `amountToBurn`, not the full contract balance.

```
function burnFaucetTokens(uint256 amountToBurn) public onlyOwner {
        require(amountToBurn <= balanceOf(address(this)), "Faucet Token
            Balance: Insufficient");

        // transfer faucet balance to owner first before burning
        // ensures owner has a balance before _burn (owner only
           function) can be called successfully
-       _transfer(address(this), msg.sender, balanceOf(address(this)));
+       _transfer(address(this), msg.sender, amountToBurn);

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

Low

[L-1] Mint Function Restricts Minting To The Contract Address, Limiting Flexibility and Scalability.

Description

- The `mintFaucetTokens()` function should mint faucet tokens based on more modular approach, allowing the owner to mint the faucet token as per declared `maxMinCap`.
- The `mintFaucetTokens()` function is restricted to magic number i.e. 1000 tokens, forcing the owner to call it frequently to refill the faucet.

```
1  function mintFaucetTokens(address to, uint256 amount) public onlyOwner
       {
2          if (to != address(this)) {
3              revert RaiseBoxFaucet_MiningToNonContractAddressFailed();
4          }
5
6  @>      if (balanceOf(address(to)) > 1000 * 10 ** 18) {
7              revert RaiseBoxFaucet_FaucetNotOutOfTokens();
8          }
9
10         _mint(to, amount);
11
12         emit MintedNewFaucetTokens(to, amount);
13     }
```

Risk

Likelihood:

- Scalability issues arises once `dailyClaimLimit` is greater than magic number 1000 tokens.

Impact:

- The faucet can run out of tokens too frequently and get stuck once balance logic misaligns with `dailyClaimLimit` or its is exceeds by the magic number i.e. 1000 tokens by calling `adjustDailyClaimLimit`.

Recommended Mitigation

In order to mitigate the issue, Use modular approach for conditioning the minting logic.

```
function mintFaucetTokens(address to, uint256 amount) public onlyOwner
    {
         if (to != address(this)) {
             revert RaiseBoxFaucet_MiningToNonContractAddressFailed();
         }

-        if (balanceOf(address(to)) > 1000 * 10 ** 18) {
+        if (balanceOf(address(to)) > (dailyClaimCount * faucetDrip) {
             // auditi - Low - Magic Numbers
             revert RaiseBoxFaucet_FaucetNotOutOfTokens();
         }

         _mint(to, amount);

         emit MintedNewFaucetTokens(to, amount);
     }
```

[L-2] Unused State Variable - blockTime

Description

- The variable `blockTime` is set once at deployment and never used.

```
@>     uint256 public blockTime = block.timestamp;
```

Risk

Likelihood:

- None

Impact:

- No functional bug, but misleading - suggests it tracks current time when it's actually static.

Proof of Concept

None

Recommended Mitigation

Remove this variable as it serves no purpose

```
-      uint256 public blockTime = block.timestamp;
+     uint256 public blockTime = block.timestamp;
```

[L-3] Inconsistent Daily Reset Logic

Description

- The `claimFaucetTokens()` function should contains same daily reset logics for both sepolia eth drip and faucet token drips.
- Two different daily reset mechanisms using different tracking variables.

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

        // (lastClaimTime[faucetClaimer] == 0);

        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();
        }

        // drip sepolia eth to first time claimers if supply hasn't ran
            out or sepolia drip not paused**
        // still checks
        if (!hasClaimedEth[faucetClaimer] && !sepEthDripsPaused) {
            uint256 currentDay = block.timestamp / 24 hours;
```

```
27
28 @>           if (currentDay > lastDripDay) {
29 @>               lastDripDay = currentDay;
30 @>               dailyDrips = 0;
31                  // dailyClaimCount = 0;
32              }
33
34              if (dailyDrips + sepEthAmountToDrip <= dailySepEthCap &&
                    address(this).balance >= sepEthAmountToDrip) {
35                  hasClaimedEth[faucetClaimer] = true;
36                  dailyDrips += sepEthAmountToDrip;
37
38                  (bool success,) = faucetClaimer.call{value:
                        sepEthAmountToDrip}("");
39
40                  if (success) {
41                      emit SepEthDripped(faucetClaimer,
                            sepEthAmountToDrip);
42                  } else {
43                      revert RaiseBoxFaucet_EthTransferFailed();
44                  }
45              } else {
46                  emit SepEthDripSkipped(
47                      faucetClaimer,
48                      address(this).balance < sepEthAmountToDrip ? "
                            Faucet out of ETH" : "Daily ETH cap reached"
49                  );
50              }
51          } else {
52              dailyDrips = 0;
53          }
54
55          /**
56           *
57           * @param lastFaucetDripDay tracks the last day a claim was
                 made
58           * @notice resets the @param dailyClaimCount every 24 hours
59           */
60 @>       if (block.timestamp > lastFaucetDripDay + 1 days) {
61 @>           lastFaucetDripDay = block.timestamp;
62 @>           dailyClaimCount = 0;
63 @>       }
64
65          // Effects
66
67          lastClaimTime[faucetClaimer] = block.timestamp;
68          dailyClaimCount++;
69
70          // Interactions
71          _transfer(address(this), faucetClaimer, faucetDrip);
72
```

```
73          emit Claimed(msg.sender, faucetDrip);
74      }
```

Risk

Likelihood:

- After every 24 hours it resets the block.timestamp that is misaligned with other days resets.

Impact:

- `dailyDrips` and `dailyClaimCount` can reset at different times, causing inconsistent daily limits

Proof of Concept

None

Recommended Mitigation

Use consistent time tracking for both resets.

```
1       function claimFaucetTokens() public {
2           // Checks
3           faucetClaimer = msg.sender;
4
5           // (lastClaimTime[faucetClaimer] == 0);
6
7           if (block.timestamp < (lastClaimTime[faucetClaimer] +
               CLAIM_COOLDOWN)) {
8               revert RaiseBoxFaucet_ClaimCooldownOn();
9           }
10
11          if (faucetClaimer == address(0) || faucetClaimer == address(
               this) || faucetClaimer == Ownable.owner()) {
12              revert
                   RaiseBoxFaucet_OwnerOrZeroOrContractAddressCannotCallClaim
                   ();
13          }
14
15          if (balanceOf(address(this)) <= faucetDrip) {
16              revert RaiseBoxFaucet_InsufficientContractBalance();
17          }
18
19          if (dailyClaimCount >= dailyClaimLimit) {
```

```
20              revert RaiseBoxFaucet_DailyClaimLimitReached();
21          }
22
23          // drip sepolia eth to first time claimers if supply hasn't ran
                out or sepolia drip not paused**
24          // still checks
25          if (!hasClaimedEth[faucetClaimer] && !sepEthDripsPaused) {
26
27  +           if (block.timestamp > lastFaucetDripDay + 1 days) {
28  +           lastFaucetDripDay = block.timestamp;
29  +           dailyDrips = 0;
30  +           }
31
32  -           uint256 currentDay = block.timestamp / 24 hours;
33  -
34  -           if (currentDay > lastDripDay) {
35  -               lastDripDay = currentDay;
36  -               dailyDrips = 0;
37  -               // dailyClaimCount = 0;
38  -           }
39
40              if (dailyDrips + sepEthAmountToDrip <= dailySepEthCap &&
                    address(this).balance >= sepEthAmountToDrip) {
41                  hasClaimedEth[faucetClaimer] = true;
42                  dailyDrips += sepEthAmountToDrip;
43
44                  (bool success,) = faucetClaimer.call{value:
                        sepEthAmountToDrip}("");
45
46                  if (success) {
47                      emit SepEthDripped(faucetClaimer,
                            sepEthAmountToDrip);
48                  } else {
49                      revert RaiseBoxFaucet_EthTransferFailed();
50                  }
51              } else {
52                  emit SepEthDripSkipped(
53                      faucetClaimer,
54                      address(this).balance < sepEthAmountToDrip ? "
                            Faucet out of ETH" : "Daily ETH cap reached"
55                  );
56              }
57          } else {
58              dailyDrips = 0;
59          }
60
61          /**
62           *
63           * @param lastFaucetDripDay tracks the last day a claim was
                 made
64           * @notice resets the @param dailyClaimCount every 24 hours
```

```
65          */
66         if (block.timestamp > lastFaucetDripDay + 1 days) {
67             lastFaucetDripDay = block.timestamp;
68             dailyClaimCount = 0;
69         }
70
71         // Effects
72
73         lastClaimTime[faucetClaimer] = block.timestamp;
74         dailyClaimCount++;
75
76         // Interactions
77         _transfer(address(this), faucetClaimer, faucetDrip);
78
79         emit Claimed(msg.sender, faucetDrip);
80     }
```