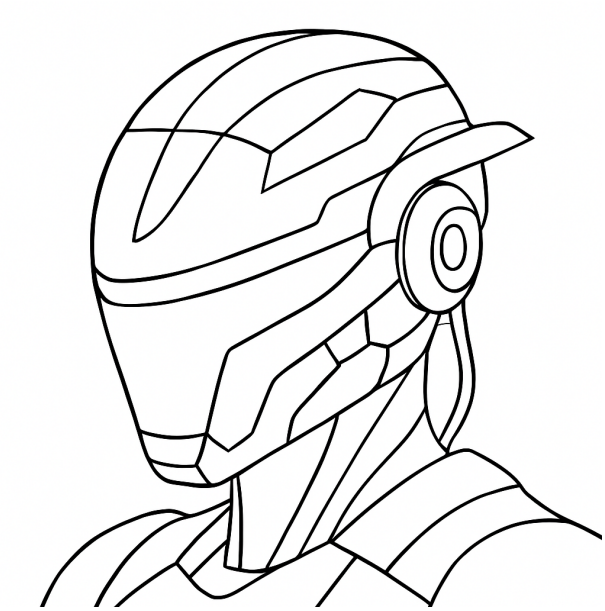

Raisebox faucet Audit Report

Version 1.0

X: @AlexScherbatyuk

October 16, 2025

Raisebox faucet Audit Report

Alexander Scherbatyuk

October 16, 2025

Prepared by: X: @AlexScherbatyuk Lead Auditors: - Alexander Scherbatyuk

Table of Contents

- High
  - [H-1] The `RaiseBoxFaucet::claimFaucetTokens` function prioritizes `sepolia ETH` drip over `faucetDrip` claim for new users, rejection of `sepolia ETH` by contract-based accounts causes denial of service (DoS) blocking users from claiming `RB` tokens
  - [H-2] The `RaiseBoxFaucet::burnFaucetTokens` transfers the entire balance of the contract to the owner instead of the `amountToBurn`, draining and potentially causing a denial of service (DoS).
- Medium
  - [M-1] Reentrancy attack in `RaiseBoxFaucet::claimFaucetTokens` allows first-time users to claim twice `faucetDrip` amount of `RB` tokens alongside `sepolia ETH` claim. Reduces initial supply, disrupts protocol's intended logic, potentially causing denial of service (DoS).
  - [M-2] The `dailySepEthCap` in the `RaiseBoxFaucet::claimFaucetTokens` function could be bypassed, causing `sepolia ETH` draining.
  - [M-3] The `RaiseBoxFaucet::mintFaucetTokens` function checks the balance of the contract against a magic number instead of `faucetDrip`, potentially causing a denial of service (DoS).
  - [M-4] The `RaiseBoxFaucet` contract lacks a native currency withdrawal function for the owner in case of an emergency, potentially leading to the loss of funds.
- Low
  - [L-1] The `claimFaucetTokens::lastDripDay` and `claimFaucetTokens::lastFaucetDripDay` storage variables are not updated correctly, which may lead to confusion and misinterpretation of the last drip day and last faucet drip day.
- Informational
  - [I-1] The `Claimed` event in the `RaiseBoxFaucet::claimFaucetTokens` function is emitted after the `faucetDrip` is transferred to the user, potentially causing confusion and misinterpretation of the event.
  - [I-2] The `MintedNewFaucetTokens` event in the `RaiseBoxFaucet::mintFaucetTokens` function is emitted after the `_mint` function call, potentially causing confusion and misinterpretation of the event.

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 @AlexScherbatyuk 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
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

Scope

```
1
2 src/
3 --- RaiseBoxFaucet.sol
4 --- DeployRaiseBoxFaucet.s.sol
```

Roles

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

Executive Summary

Issues found

Severity	Number of issues found
High	2
Medium	4

Severity	Number of issues found
Low	3
Info	0
Gas Optimizations	0
Total	9

Findings

High

[H-1] The `RaiseBoxFaucet::claimFaucetTokens` function prioritizes `sepolia ETH` drip over `faucetDrip` claim for new users, rejection of `sepolia ETH` by contract-based accounts causes denial of service (DoS) blocking users from claiming RB tokens

Description: The `RaiseBoxFaucet::claimFaucetTokens` function prioritizes `sepolia ETH` drip over `faucetDrip` claim for new users, sending eth to users before the `faucetDrip` tokens. The contract-based accounts that do not expect to receive `sepolia ETH` cause denial of service (DoS) reverting the transaction, blocking users from claiming `RB` tokens.

Root cause:

```
1      function claimFaucetTokens() public {
2
3          ...
4
5          if (dailyDrips + sepEthAmountToDrip <= dailySepEthCap &&
               address(this).balance >= sepEthAmountToDrip) {
6              hasClaimedEth[faucetClaimer] = true;
7              dailyDrips += sepEthAmountToDrip;
8
9              (bool success,) = faucetClaimer.call{value:
                   sepEthAmountToDrip}("");
10
11             if (success) {
12                 emit SepEthDripped(faucetClaimer,
                       sepEthAmountToDrip);
13             } else {
14 @>               revert RaiseBoxFaucet_EthTransferFailed();
15             }
16         } else {
17             emit SepEthDripSkipped(
```

```
                    faucetClaimer,
                    address(this).balance < sepEthAmountToDrip ? "
                        Faucet out of ETH" : "Daily ETH cap reached"
                );
            }

        ...

    }
```

Impact: Denial of service (DoS) blocking users from claiming RB tokens, severely breaking the functionality of the protocol.

Proof of Concept: Add the following code snippet to the `RaiseBoxFaucet.t.sol` test file.

ReentrancyAttacker contract:

```
    contract EthRejector {
        receive() external payable {
            revert("Rejecting ETH");
        }

        fallback() external payable {
            revert("Rejecting ETH");
        }
    }
```

This code snippet is designed to demonstrate the `RaiseBoxFaucet::claimFaucetTokens` function fails on ETH transfer causing denial of service (DoS)

```
    function testSepEthTransferFailedCausesDoS() public {
        // Create a contract that rejects ETH transfers
        EthRejector rejector = new EthRejector();

        // Try to claim with the rejector address - should fail on ETH
            transfer
        vm.prank(address(rejector));
        vm.expectRevert(RaiseBoxFaucet.RaiseBoxFaucet_EthTransferFailed
            .selector);
        raiseBoxFaucet.claimFaucetTokens();

        console.log("user balance of eth: ", address(rejector).balance)
            ;
        console.log("user balance of faucet tokens: ", raiseBoxFaucet.
            balanceOf(address(rejector)));
    }
```

Recommended Mitigation: Possible mitigation is to replace revert statement with emit of `SepEthDripSkipped` event, in order to allow users to claim RB tokens.

```
    function claimFaucetTokens() public {

    ...

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

            (bool success,) = faucetClaimer.call{value:
                sepEthAmountToDrip}("");

            if (success) {
                emit SepEthDripped(faucetClaimer, sepEthAmountToDrip);
            } else {
-               revert RaiseBoxFaucet_EthTransferFailed();
+               emit SepEthDripSkipped(faucetClaimer, "Receiver has
    rejected ETH");
            }
        } else {
            emit SepEthDripSkipped(
                faucetClaimer,
                address(this).balance < sepEthAmountToDrip ? "Faucet
                    out of ETH" : "Daily ETH cap reached"
            );
        }

        ...

    }
```

[H-2] The `RaiseBoxFaucet::burnFaucetTokens` transfers the entire balance of the contract to the owner instead of the `amountToBurn`, draining and potentially causing a denial of service (DoS).

Description: The `_transfer` function of the contract transfers the entire balance of the contract `balanceOf(address(this))` to the owner instead of the `amountToBurn`, and then burns the `amountToBurn` from the owner.

Root cause:

```
    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
```

```
6       @>  _transfer(address(this), msg.sender, balanceOf(address(this)));
7
8           _burn(msg.sender, amountToBurn);
9       }
```

Impact: Draining and potentially causing a denial of service (DoS).

Proof of Concept: Add the following code snippet to the `RaiseBoxFaucet.t.sol` test file.

```
1       function testBurnEntireSupplyFaucetTokens() public {
2           vm.prank(owner);
3           uint256 balanceOfContract = raiseBoxFaucet.balanceOf(address(
                raiseBoxFaucet));
4           uint256 amountToBurn = 1000e18;
5           raiseBoxFaucet.burnFaucetTokens(amountToBurn);
6           uint256 balanceOfContractAfterBurn = raiseBoxFaucet.balanceOf(
                address(raiseBoxFaucet));
7
8           console.log("Balance of contract after burn: ", raiseBoxFaucet.
                balanceOf(address(raiseBoxFaucet)));
9           assertNotEq(
10              balanceOfContract,
11              balanceOfContractAfterBurn + amountToBurn,
12              "Balance of contract after burn + amount to burn is not
                    equal to balance of contract before burn"
13          );
14          assertEq(raiseBoxFaucet.balanceOf(address(raiseBoxFaucet)), 0,
                "Balance of contract after burn is 0");
15      }
```

Recommended Mitigation: Use the `amountToBurn` instead of `balanceOf(address(this))` in the `_transfer` function.

```
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   +       _transfer(address(this), msg.sender, amountToBurn);
8
9           _burn(msg.sender, amountToBurn);
10      }
```

Medium

[M-1] Reentrancy attack in `RaiseBoxFaucet::claimFaucetTokens` allows first-time users to claim twice `faucetDrip` amount of RB tokens alongside `sepolia ETH` claim. Reduces initial supply, disrupts protocol's intended logic, potentially causing denial of service (DoS).

Description: The `RaiseBoxFaucet::claimFaucetTokens` function sends all first-time users `sepEthAmountToDrip` amount of `sepolia ETH` with `faucetDrip` claim. This allows a first-time user contract to claim twice `faucetDrip` amount of `RB` tokens alongside `sepolia ETH` claim.

Impact: These actions reduce initial supply, disrupt protocol's intended logic, potentially causing denial of service (DoS) speeding the run-out of `RB` tokens initial supply.

Root cause:

```
1
2       function claimFaucetTokens() public {
3       ...
4
5           if (!hasClaimedEth[faucetClaimer] && !sepEthDripsPaused) {
6               uint256 currentDay = block.timestamp / 24 hours;
7
8               if (currentDay > lastDripDay) {
9                   lastDripDay = currentDay;
10                  dailyDrips = 0;
11                  // dailyClaimCount = 0;
12              }
13
14              if (dailyDrips + sepEthAmountToDrip <= dailySepEthCap &&
                    address(this).balance >= sepEthAmountToDrip) {
15                  hasClaimedEth[faucetClaimer] = true;
16                  dailyDrips += sepEthAmountToDrip;
17
18      @>          (bool success,) = faucetClaimer.call{value:
            sepEthAmountToDrip}("");
19
20                  if (success) {
21                      emit SepEthDripped(faucetClaimer,
                            sepEthAmountToDrip);
22                  } else {
23                      revert RaiseBoxFaucet_EthTransferFailed();
24                  }
25              } else {
26                  emit SepEthDripSkipped(
27                      faucetClaimer,
28                      address(this).balance < sepEthAmountToDrip ? "
                            Faucet out of ETH" : "Daily ETH cap reached"
29                  );
```

```
30              }
31          }
32          ...
33      }
```

Proof of Concept: Add the following code snippet to the `RaiseBoxFaucet.t.sol` test file.

ReentrancyAttacker contract:

```
1       contract ReentrancyAttacker {
2           RaiseBoxFaucet victim;
3           uint256 public counter = 0;
4
5           constructor(RaiseBoxFaucet _victim) {
6               victim = _victim;
7           }
8
9           function attack() public payable {
10              victim.claimFaucetTokens();
11          }
12
13          receive() external payable {
14              console.log("balance of victim: ", address(victim).balance)
                    ;
15              if (counter < 2) {
16                  counter++;
17                  victim.claimFaucetTokens();
18              }
19          }
20      }
```

This code snippet is designed to demonstrate the `RaiseBoxFaucet::claimFaucetTokens` function being successfully called twice through the `reentrancy` attack.

```
1       function testReentrancyAttack() public {
2           ReentrancyAttacker attacker = new ReentrancyAttacker(
                raiseBoxFaucet);
3
4           uint256 lastAttackerClaimTime = raiseBoxFaucet.
                getUserLastClaimTime(address(attacker));
5           uint256 faucetDrip = raiseBoxFaucet.faucetDrip();
6           uint256 dailyClaimCount = raiseBoxFaucet.dailyClaimCount();
7           uint256 dailyClaimLimit = raiseBoxFaucet.dailyClaimLimit();
8
9           console.log("Before facuet Contract ETH balance", address(
                raiseBoxFaucet).balance);
10          console.log("Before attacker faucet token balance",
                raiseBoxFaucet.balanceOf(address(attacker)));
11
12          console.log("Last time attacker has claimed faucets: ",
                lastAttackerClaimTime);
```

```
13          console.log("faucetDrip: ", faucetDrip);
14          console.log("dailyClaimCount: ", dailyClaimCount);
15          console.log("dailyClaimLimit: ", dailyClaimLimit);
16
17          attacker.attack();
18          bool claimed = raiseBoxFaucet.getHasClaimedEth(address(attacker
                ));
19          assertEq(claimed, true, "User has not successfully claimed eth"
                );
20
21          uint256 afterLastAttackerClaimTime = raiseBoxFaucet.
                getUserLastClaimTime(address(attacker));
22          uint256 afterFucetDrip = raiseBoxFaucet.faucetDrip();
23          uint256 afterDailyClaimCount = raiseBoxFaucet.dailyClaimCount()
                ;
24          uint256 afterDailyClaimLimit = raiseBoxFaucet.dailyClaimLimit()
                ;
25
26          console.log("After facuet Contract ETH balance", address(
                raiseBoxFaucet).balance);
27          console.log("After attacker faucet token balance",
                raiseBoxFaucet.balanceOf(address(attacker)));
28
29          console.log("Last time attacker has claimed faucets: ",
                afterLastAttackerClaimTime);
30          console.log("faucetDrip: ", afterFucetDrip);
31          console.log("dailyClaimCount: ", afterDailyClaimCount);
32          console.log("dailyClaimLimit: ", afterDailyClaimLimit);
33          console.log("number of attackes", attacker.counter());
34      }
```

Recommended Mitigation: Possible mitigation is to move the `sepolia ETH` claim after `faucetDrip` claim, to ensure that the `lastClaimTime` for the user is changed before the `sepolia ETH` claim.

```
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
75  +        // drip sepolia eth to first time claimers if supply hasn't
    ran out or sepolia drip not paused**
76  +       // still checks
77  +       if (!hasClaimedEth[faucetClaimer] && !sepEthDripsPaused) {
78  +           uint256 currentDay = block.timestamp / 24 hours;
79  +
80  +           if (currentDay > lastDripDay) {
81  +               lastDripDay = currentDay;
82  +               dailyDrips = 0;
83  +               // dailyClaimCount = 0;
84  +           }
85  +
86  +           if (dailyDrips + sepEthAmountToDrip <= dailySepEthCap &&
    address(this).balance >= sepEthAmountToDrip) {
87  +               hasClaimedEth[faucetClaimer] = true;
88  +               dailyDrips += sepEthAmountToDrip;
89  +
90  +               (bool success,) = faucetClaimer.call{value:
    sepEthAmountToDrip}("");
91  +
92  +               if (success) {
93  +                   emit SepEthDripped(faucetClaimer,
    sepEthAmountToDrip);
94  +               } else {
95  +                   revert RaiseBoxFaucet_EthTransferFailed();
96  +               }
97  +           } else {
98  +               emit SepEthDripSkipped(
99  +                   faucetClaimer,
100 +                   address(this).balance < sepEthAmountToDrip ? "
    Faucet out of ETH" : "Daily ETH cap reached"
101 +               );
102 +           }
```

```
103 +         } else {
104 +             dailyDrips = 0;
105 +         }
106 
107       }
```

[M-2] The `dailySepEthCap` in the `RaiseBoxFaucet::claimFaucetTokens` function could be bypassed, causing `sepolia ETH` draining.

Description: The `dailyDrips` storage variable in the `RaiseBoxFaucet::claimFaucetTokens` function increments each time a new user claims `sepolia ETH`. However, when a previously claimed `sepolia ETH` user claims it again after the 3-day cooldown, the `dailyDrips` is set to `0`, allowing new users to claim `sepolia ETH` beyond the `dailySepEthCap` limit until either the `sepolia ETH` supply runs out or the `dailyClaimLimit` limit is reached.

Root cause:

```
1     function claimFaucetTokens() public {
2 
3         ...
4 
5         if (!hasClaimedEth[faucetClaimer] && !sepEthDripsPaused) {
6             uint256 currentDay = block.timestamp / 24 hours;
7 
8             if (currentDay > lastDripDay) {
9                 lastDripDay = currentDay;
10                dailyDrips = 0;
11                // dailyClaimCount = 0;
12            }
13 
14            if (dailyDrips + sepEthAmountToDrip <= dailySepEthCap &&
                  address(this).balance >= sepEthAmountToDrip) {
15                hasClaimedEth[faucetClaimer] = true;
16                dailyDrips += sepEthAmountToDrip;
17 
18                (bool success,) = faucetClaimer.call{value:
                      sepEthAmountToDrip}("");
19 
20                if (success) {
21                    emit SepEthDripped(faucetClaimer,
                          sepEthAmountToDrip);
22                } else {
23                    revert RaiseBoxFaucet_EthTransferFailed();
24                }
25            } else {
26                emit SepEthDripSkipped(
27                    faucetClaimer,
```

```
28                      address(this).balance < sepEthAmountToDrip ? "
                            Faucet out of ETH" : "Daily ETH cap reached"
29                  );
30              }
31          } else {
32  @>              dailyDrips = 0;
33          }
34
35          ...
36      }
```

Impact: `sepolia ETH` draining, causing disruption in the protocol functionality.

Proof of Concept: Add the following code snippet to the `RaiseBoxFaucet.t.sol` test file.

This code snippet is designed to demonstrate the `RaiseBoxFaucet::claimFaucetTokens` function bypasses the `dailySepEthCap` limit, causing `sepolia ETH` draining.

```
1       function testEthClaimSupply() public {
2           owner = address(this);
3
4           raiseBoxFaucet = new RaiseBoxFaucet("raiseboxtoken", "RB", 1000
                * 10 ** 18, 0.02 ether, 0.5 ether);
5
6           raiseBoxFaucetContractAddress = address(raiseBoxFaucet);
7
8           vm.deal(raiseBoxFaucetContractAddress, 1 ether);
9
10          uint256 initialSepoliaETHSupply = address(raiseBoxFaucet).
                balance;
11          console.log("\n");
12          console.log("Initial Contract ETH balance",
                initialSepoliaETHSupply);
13
14          uint256 totalClaims = 1;
15
16          // A user interaction before the drain
17          address previousUser = makeAddr("previousUser");
18          vm.prank(previousUser);
19          raiseBoxFaucet.claimFaucetTokens();
20
21          console.log("\n");
22          console.log("A user claims faucets");
23          console.log("User address: ", previousUser);
24          console.log("Contract ETH balance", address(raiseBoxFaucet).
                balance);
25          console.log("User faucet token balance", raiseBoxFaucet.
                balanceOf(previousUser));
26          console.log("User ETH balance", previousUser.balance);
27          console.log("dailyClaimCount: ", raiseBoxFaucet.dailyClaimCount
                ());
```

```
28          console.log("dailyClaimLimit: ", raiseBoxFaucet.dailyClaimLimit
                ());
29
30          console.log("\n");
31          console.log("Whait for 3 days cooldown");
32
33          // Cooldown
34          vm.warp(block.timestamp + 3 days + 1 minutes);
35          vm.roll(block.number + 100);
36
37          console.log("\n");
38          console.log("Drain starts here");
39
40          address user;
41          for (uint256 i = 1; i <= 50; ++i) {
42              user = payable(address(uint160(100 + i)));
43
44              vm.prank(user);
45              raiseBoxFaucet.claimFaucetTokens();
46              totalClaims += 1;
47          }
48
49          console.log("\n");
50          console.log("Middle way, a user after cooldown claims again,
                resetting dailyDrips to 0");
51
52          vm.prank(previousUser);
53          raiseBoxFaucet.claimFaucetTokens();
54
55          console.log("\n");
56          console.log("User address: ", previousUser);
57          console.log("Contract ETH balance", address(raiseBoxFaucet).
                balance);
58          console.log("User faucet token balance", raiseBoxFaucet.
                balanceOf(previousUser));
59          console.log("User ETH balance", previousUser.balance);
60          console.log("dailyClaimCount: ", raiseBoxFaucet.dailyClaimCount
                ());
61          console.log("dailyClaimLimit: ", raiseBoxFaucet.dailyClaimLimit
                ());
62          console.log("dailyDrips has been reset to: ", raiseBoxFaucet.
                dailyDrips());
63
64          console.log("\n");
65          console.log("Drain continues");
66
67          address newUser;
68          for (uint256 i = 1; i <= 49; ++i) {
69              newUser = payable(address(uint160(200 + i)));
70
71              vm.prank(newUser);
```

```
72              raiseBoxFaucet.claimFaucetTokens();
73
74              totalClaims += 1;
75          }
76
77          uint256 afterDrainSepoliaETHSupply = address(raiseBoxFaucet).
                balance;
78
79          console.log("\n");
80          console.log("dailyClaimCount: ", raiseBoxFaucet.dailyClaimCount
                ());
81          console.log("dailyClaimLimit: ", raiseBoxFaucet.dailyClaimLimit
                ());
82          console.log("Total claimes: ", totalClaims);
83          console.log("After the drain, Contract ETH balance",
                afterDrainSepoliaETHSupply);
84
85          assertGt(initialSepoliaETHSupply, 0, "Contract ETH balance has
                not been supplied");
86          assertEq(afterDrainSepoliaETHSupply, 0, "Contract ETH balance
                has not been drained");
87      }
```

Recommended Mitigation: Possible mitigation is to reset the `dailyDrips` storage variable to `0` only on the next day by checking the `lastDripDay` storage variable.

```
1       function claimFaucetTokens() public {
2
3           ...
4
5               if (!hasClaimedEth[faucetClaimer] && !sepEthDripsPaused) {
6               uint256 currentDay = block.timestamp / 24 hours;
7
8               if (currentDay > lastDripDay) {
9                   lastDripDay = currentDay;
10                  dailyDrips = 0;
11                  // dailyClaimCount = 0;
12              }
13
14              if (dailyDrips + sepEthAmountToDrip <= dailySepEthCap &&
                    address(this).balance >= sepEthAmountToDrip) {
15                  hasClaimedEth[faucetClaimer] = true;
16                  dailyDrips += sepEthAmountToDrip;
17
18                  (bool success,) = faucetClaimer.call{value:
                        sepEthAmountToDrip}("");
19
20                  if (success) {
21                      emit SepEthDripped(faucetClaimer,
                            sepEthAmountToDrip);
22                  } else {
```

```
                    revert RaiseBoxFaucet_EthTransferFailed();
                }
            } else {
                emit SepEthDripSkipped(
                    faucetClaimer,
                    address(this).balance < sepEthAmountToDrip ? "
                        Faucet out of ETH" : "Daily ETH cap reached"
                );
            }
+         }
-         } else {
-             dailyDrips = 0;
-         }

        ...

    }
```

[M-3] The `RaiseBoxFaucet::mintFaucetTokens` function checks the balance of the contract against a magic number instead of `faucetDrip`, potentially causing a denial of service (DoS).

Description: The `RaiseBoxFaucet::mintFaucetTokens` function checks the balance of the contract against a magic number instead of `faucetDrip` before minting. The check verifies that the contract has a balance greater than `1000e18` no matter the value of `faucetDrip`, potentially causing a denial of service (DoS).

Root cause:

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

@>      if (balanceOf(address(to)) > 1000 * 10 ** 18) {
            revert RaiseBoxFaucet_FaucetNotOutOfTokens();
        }

        _mint(to, amount);

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

Impact: Denial of service (DoS) blocking the minting of `RB` tokens and users' ability to claim `RB` tokens, as the static value might be less than `faucetDrip`.

Proof of Concept: Add the following code snippet to the `RaiseBoxFaucet.t.sol` test file.

This code snippet is designed to demonstrate the `RaiseBoxFaucet::mintFaucetTokens` function checks the balance of the contract against a magic number instead of `faucetDrip`, causing a denial of service (DoS) blocking the minting of `RB` tokens required for claiming `RB` tokens.

```
    function testMintedNewFaucetTokensEventsLessThanFaucetDrip() public {
        owner = address(this);
        raiseBoxFaucet = new RaiseBoxFaucet("raiseboxtoken", "RB", 49000000 * 10 ** 18, 0.02 ether, 0.5 ether);
        raiseBoxFaucetContractAddress = address(raiseBoxFaucet);

        vm.deal(raiseBoxFaucetContractAddress, 1 ether);

        console.log("Initial balance of contract: ", raiseBoxFaucet.balanceOf(address(raiseBoxFaucet)));

        // reduce balance of contract
        address newUser;
        for (uint256 i = 1; i <= 20; ++i) {
            newUser = payable(address(uint160(200 + i)));

            vm.prank(newUser);
            raiseBoxFaucet.claimFaucetTokens();
            console.log("User address: ", newUser);
            console.log("User balance of ETH: ", newUser.balance);
            console.log("User balance of faucet tokens: ", raiseBoxFaucet.balanceOf(newUser));
        }

        console.log("\n Actual balance: ", raiseBoxFaucet.balanceOf(address(raiseBoxFaucet)));

        vm.prank(user1);
        vm.expectRevert(RaiseBoxFaucet.RaiseBoxFaucet_InsufficientContractBalance.selector);
        raiseBoxFaucet.claimFaucetTokens();

        // Try to mint more;
        uint256 faucetDrip = raiseBoxFaucet.faucetDrip();
        console.log("Balance required for claim: ", faucetDrip);
        vm.prank(owner);
        vm.expectRevert(RaiseBoxFaucet.RaiseBoxFaucet_FaucetNotOutOfTokens.selector);
        raiseBoxFaucet.mintFaucetTokens(address(raiseBoxFaucet), faucetDrip);

        assertGt(
            faucetDrip,
            raiseBoxFaucet.balanceOf(address(raiseBoxFaucet)),
```

```
38              "Balance required for claim is greater than balance of
                    contract"
39          );
40      }
```

Recommended Mitigation: The magic number `1000 * 10 ** 18` should be replaced with `faucetDrip` to prevent denial of service (DoS) blocking minting `RB` tokens required for claiming `RB` tokens.

```
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      }
```

[M-4] The `RaiseBoxFaucet` contract lacks a native currency withdrawal function for the owner in case of an emergency, potentially leading to the loss of funds.

Description: The `RaiseBoxFaucet` contract, which has functionality to receive and distribute `Sepolia ETH`, lacks admin functionality to withdraw Sepolia ETH in case of an emergency that would require redeploying the contract and refunding the new contract.

Impact: Loss of funds in case of an emergency that would require redeploying the contract and refunding the new contract.

Proof of Concept:

A possible cause for an emergency is another finding where the faucetDrip value, incorrectly set during deployment, breaks the contract's functionality.

Recommended Mitigation: Consider adding a RaiseBoxFaucet::withdrawNativeFunds function to allow the owner to withdraw or transfer Sepolia ETH in case of an emergency.

```
1   +    function withdrawNativeFunds(addess receiver) external onlyOwner {
2   +        payable(recipient).call{value: address(this).balance}("");
3   +    }
```

Low

[L-1] The `claimFaucetTokens::lastDripDay` and `claimFaucetTokens::lastFaucetDripDay` storage variables are not updated correctly, which may lead to confusion and misinterpretation of the last drip day and last faucet drip day.

Description: The `claimFaucetTokens::lastDripDay` variable stores an index from the epoch, which advances based on a precise timestamp, making it more accurate. On the other hand, the `claimFaucetTokens::lastFaucetDripDay` variable is stored as a raw timestamp and is used to reset dailyClaimCount after 24 hours from the precise timestamp. This causes a difference between the last drip day and the last faucet drip day.

Root cause:

```
1          function claimFaucetTokens() public {
2
3          ...
4
5          // drip sepolia eth to first time claimers if supply hasn't ran
               out or sepolia drip not paused**
6          // still checks
7          if (!hasClaimedEth[faucetClaimer] && !sepEthDripsPaused) {
8              uint256 currentDay = block.timestamp / 24 hours;
9
10             if (currentDay > lastDripDay) {
11                 lastDripDay = currentDay;
12                 dailyDrips = 0;
13                 // dailyClaimCount = 0;
14             }
15
16         ...
17
18         /**
19          *
20          * @param lastFaucetDripDay tracks the last day a claim was
               made
21          * @notice resets the @param dailyClaimCount every 24 hours
22          */
23         if (block.timestamp > lastFaucetDripDay + 1 days) {
24 @>             lastFaucetDripDay = block.timestamp;
25             dailyClaimCount = 0;
26         }
27
28         ...
29     }
```

Impact: Confusion and misinterpretation of the last drip day and last faucet drip day creating difference of the last drip day and last faucet drip day.

Proof of Concept: Add the following code snippet to the `RaiseBoxFaucet.t.sol` test file.

This code snippet is designed to demonstrate the `claimFaucetTokens::lastDripDay` and `claimFaucetTokens::lastFaucetDripDay` storage variables show different values after 24 hours.

```
    function testCheckLastFaucetDripDayAndLastDripDayDifferenceAfter24Hours() public {
        vm.prank(user1);
        raiseBoxFaucet.claimFaucetTokens();
        console.log("Day 1 claim");

        uint256 firstLastDripDay = raiseBoxFaucet.lastDripDay() * 24 hours;
        uint256 firstLastFaucetDripDay = raiseBoxFaucet.lastFaucetDripDay();

        console.log("Last drip day: ", firstLastDripDay);
        console.log("Last faucet drip day: ", firstLastFaucetDripDay);

        vm.warp(block.timestamp + 1 days);

        console.log("\n");
        console.log("Day 2 claim");

        vm.prank(user2);
        raiseBoxFaucet.claimFaucetTokens();

        uint256 secondLastDripDay = raiseBoxFaucet.lastDripDay() * 24 hours;
        uint256 secondLastFaucetDripDay = raiseBoxFaucet.lastFaucetDripDay();

        console.log("Last drip day: ", secondLastDripDay);
        console.log("Last faucet drip day: ", secondLastFaucetDripDay);

        assertEq(
            firstLastDripDay, firstLastFaucetDripDay, "At first claim, last drip day is equal to last faucet drip day"
        );

        assertNotEq(
            secondLastDripDay,
            secondLastFaucetDripDay,
            "At second claim, last drip day is not equal to last faucet drip day"
        );
    }
```

Recommended Mitigation: Update the `claimFaucetTokens::lastFaucetDripDay` storage

variable correctly.

```
-         if (block.timestamp > lastFaucetDripDay + 1 days) {
-             lastFaucetDripDay = block.timestamp;
-             dailyClaimCount = 0;
-         }

+         uint256 currentFaucetDripDay = block.timestamp / 24 hours;
+
+         if (currentFaucetDripDay > lastFaucetDripDay) {
+             lastFaucetDripDay = currentFaucetDripDay;
+             dailyClaimCount = 0;
+         }
```

Informational

[I-1] The `Claimed` event in the `RaiseBoxFaucet::claimFaucetTokens` function is emitted after the `faucetDrip` is transferred to the user, potentially causing confusion and misinterpretation of the event.

Description: The `Claimed` event is emitted after the `faucetDrip` is transferred to the user, potentially causing confusion and misinterpretation of the event by users and dapps.

Root cause:

```
    function claimFaucetTokens() public {

    ...

        // Effects

        lastClaimTime[faucetClaimer] = block.timestamp;
        dailyClaimCount++;


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

@>      emit Claimed(msg.sender, faucetDrip);
    ...
    }
```

Impact: Confusion and misinterpretation of the event by users and dapps.

Proof of Concept: Add the following code snippet to the `RaiseBoxFaucet.t.sol` test file.

This code snippet is designed to demonstrate the `Claimed` event is emitted after the `faucetDrip` is transferred to the user, after all standard ERC20 events have already been emitted.

```
    event SepEthDripped(address indexed claimant, uint256 amount);
    event Transfer(address indexed from, address indexed to, uint256
        value);
    event Claimed(address indexed user, uint256 amount);

    function testClaimedEvents() public {
        vm.expectEmit(true, false, false, true, address(raiseBoxFaucet)
            );
        emit SepEthDripped(user1, raiseBoxFaucet.sepEthAmountToDrip());
        vm.expectEmit(true, true, false, true, address(raiseBoxFaucet))
            ;
        emit Transfer(address(raiseBoxFaucet), user1, raiseBoxFaucet.
            faucetDrip());
        vm.expectEmit(true, false, false, true, address(raiseBoxFaucet)
            );
        emit Claimed(user1, raiseBoxFaucet.faucetDrip());
        vm.prank(user1);
        raiseBoxFaucet.claimFaucetTokens();
        console.log("user balance of ETH: ", user1.balance);
        console.log("user balance of faucet tokens: ", raiseBoxFaucet.
            balanceOf(user1));
    }
```

Recommended Mitigation: To follow the Check-Effects-Interactions (CEI) pattern and emit the event before the interaction, the `Claimed` event should be emitted before the `faucetDrip` is transferred to the user.

```
    function claimFaucetTokens() public {

    ...

        // Effects

        lastClaimTime[faucetClaimer] = block.timestamp;
        dailyClaimCount++;
+        emit Claimed(msg.sender, faucetDrip);

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

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

[I-2] The `MintedNewFaucetTokens` event in the `RaiseBoxFaucet::mintFaucetTokens` function is emitted after the `_mint` function call, potentially causing confusion and misinterpretation of the event.

Description: The `MintedNewFaucetTokens` event is emitted after the `_mint` function call, potentially causing confusion and misinterpretation of the event.

Root cause:

```
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     }
```

Impact: Confusion and misinterpretation of the event by users and dapps.

Proof of Concept: Add the following code snippet to the `RaiseBoxFaucet.t.sol` test file.

This code snippet is designed to demonstrate the `MintedNewFaucetTokens` event is emitted after the `_mint` function, which also emits events.

```
1      function testMintNewFaucetTokensEvents() public {
2          // reduce balance of contract
3          console.log("Balance of contract: ", raiseBoxFaucet.balanceOf(
               address(raiseBoxFaucet)));
4          vm.prank(owner);
5          raiseBoxFaucet.burnFaucetTokens(raiseBoxFaucet.balanceOf(
               address(raiseBoxFaucet)) - 1);
6          console.log("Balance of contract after burn: ", raiseBoxFaucet.
               balanceOf(address(raiseBoxFaucet)));
7          // Try to mint more;
8          vm.prank(owner);
9          vm.expectEmit(true, true, false, true, address(raiseBoxFaucet))
               ;
10         // First event transfer
11         emit Transfer(address(0), address(raiseBoxFaucet), 2222e18);
12         vm.expectEmit(true, false, false, true, address(raiseBoxFaucet)
               );
13         // Second event minted, after transfer should be in reverse
               order
```

```
14          emit MintedNewFaucetTokens(address(raiseBoxFaucet), 2222e18);
15          raiseBoxFaucet.mintFaucetTokens(address(raiseBoxFaucet), 2222
               e18);
16          console.log("Balance of contract after mint: ", raiseBoxFaucet.
               balanceOf(address(raiseBoxFaucet)));
17      }
```

Recommended Mitigation: Move the `MintedNewFaucetTokens` event before the `_mint` function call.

```
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   +        emit MintedNewFaucetTokens(to, amount);
10          _mint(to, amount);
11
12  -        emit MintedNewFaucetTokens(to, amount);
13      }
```