shieldify

Rooster

SECURITY REVIEW

Date: 8 January 2025

CONTENTS

1. About Shieldify

Positioned as the first hybrid Web3 Security company, Shieldify shakes things up with a unique subscription-based auditing model that entitles the customer to unlimited audits within its duration, as well as top-notch service quality thanks to a disruptive 6-layered security approach. The company works with very well-established researchers in the space and has secured multiple millions in TVL across protocols, also can audit codebases written in Solidity, Rust, Go, Vyper, Move and Cairo.

Learn more about us at shieldify.org.

2. Disclaimer

This security review does not guarantee bulletproof protection against a hack or exploit. Smart contracts are a novel technological feat with many known and unknown risks. The protocol, which this report is intended for, indemnifies Shieldify Security against any responsibility for any misbehavior, bugs, or exploits affecting the audited code during any part of the project's life cycle. It is also pivotal to acknowledge that modifications made to the audited code, including fixes for the issues described in this report, may introduce new problems and necessitate additional auditing.

3. About Rooster

Rooster is DEX on the Plume blockchain built on top of the Maverick AMM.

4. Risk Classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

4.1 Impact

- **High** – results in a significant risk for the protocol's overall well-being. Affects all or most users
- **Medium** – results in a non-critical risk for the protocol affects all or only a subset of users, but is still unacceptable
- **Low** – losses will be limited but bearable – and covers vectors similar to griefing attacks that can be easily repaired

4.2 Likelihood

- **High** – almost certain to happen and highly lucrative for execution by malicious actors
- **Medium** – still relatively likely, although only conditionally possible
- **Low** – requires a unique set of circumstances and poses non-lucrative cost-of-execution to rewards ratio for the actor

5. Security Review Summary

The security review was conducted over the span of 9 days by the core Shieldify team. The code is written professionally with all best practices considered. The security review identified two Medium-severity issues, concerning input that is not properly checked and several low and informational issues that impact the UX.

Shieldify extends their gratitude to Rooster for cooperating over all the questions our team had.

5.1 Protocol Summary

Project Name	**Rooster**
Repository	rooster-contracts
Type of Project	DEX, Built on top of the Maverick AMM
Audit Timeline	9 days
Review Commit Hash	44b242b46f76b8ede8ebb4244c876211d15fa48a
Fixes Review Commit Hash	77f0696d3bab638836bb3d81b8a7c80665276d82

5.2 Scope

The following smart contracts were in the scope of the security review:

File	nSLOC
ve33/contracts/VotingEscrow.sol	123
ve33/contracts/interfaces/IVotingEscrowNft.sol	13
ve33/contracts/interfaces/IVotingDistributor.sol	88
ve33/contracts/interfaces/IPoolDistributor.sol	15
ve33/contracts/interfaces/IOrphanDistributor.sol	12
ve33/contracts/interfaces/INftImage.sol	4
ve33/contracts/interfaces/IVotingEscrow.sol	23
ve33/contracts/interfaces/ILpReward.sol	45
ve33/contracts/LpReward.sol	187
ve33/contracts/VotingEscrowNft.sol	139
ve33/contracts/NftImage.sol	94
ve33/contracts/OrphanDistributor.sol	39
ve33/contracts/Rooster.sol	18

ve33/contracts/PoolDistributor.sol	83
ve33/contracts/VotingDistributor.sol	289
ve33/contracts/base/ConfigAdmin.sol	24
ve33/contracts/base/IConfigAdmin.sol	5
ve33/contracts/base/IEpoch.sol	7
ve33/contracts/base/Epoch.sol	48
Total	**1256**

6. Findings Summary

The following number of issues have been identified, sorted by their severity:

- **Medium** issues: **1**
- **Low** issues: **1**
- **Informational** issues: **4**

ID	Title	Severity	Status
[M-01]	BinIds Are Not Checked In `PoolDistributor.claimLpAmount()` Function	Medium	Fixed
[L-01]	Users Will Not Be Able To Vote On The Epoch Time Itself	Low	Acknowledged
[I-01]	BinId That Is 0 Cannot Be Claimed	Informational	Acknowledged
[I-02]	Permalock In `VotingEscrowNft.sol` Does Not Actually Permanently Lock Voting Duration	Informational	Acknowledged
[I-03]	Weights Can Be Set At Any Arbitrary Amount	Informational	Acknowledged
[I-04]	User Can Stake And Vote On The Same Epoch Itself	Informational	Acknowledged

7. Findings

[M-01] BinIds Are Not Checked In `PoolDistributor.claimLpAmount()` Function

Severity

Medium Risk

Description

In the `PoolDistributor.claimLpAmount()` function, one of the parameters is an array of `uint32 binIds`. These IDs are supposed to be sorted in order, as seen in the code:

```
function claimLpAmount(
  uint256 tokenId,
  IMaverickV2Pool pool,
  uint32[] memory binIds,
  uint256 epoch
) public view returns (uint256 amount) {
  ClaimData memory claimData;
  claimData.epochDistributionAmount = disbursement[epoch][pool];
  if (claimData.epochDistributionAmount == 0) return 0;

  claimData.totalAFeeForEpoch = lpReward.globalTracker(pool, true, epoch)
     ;
  claimData.totalBFeeForEpoch = lpReward.globalTracker(pool, false, epoch
     );

@> claimData.previousBinId = 0;
@>for (uint256 k; k < binIds.length; k++) {
@>  uint32 binId = binIds[k];
@>  if (binId <= claimData.previousBinId) revert BinIdsMustBeOrdered();
    uint256 tokenADistributionFractionD18 = lpReward.feeProrata(pool,
       true, tokenId, binId, epoch);
    uint256 tokenBDistributionFractionD18 = lpReward.feeProrata(pool,
       false, tokenId, binId, epoch);

    if (claimData.totalAFeeForEpoch == 0) {
      amount += Math.mulFloor(claimData.epochDistributionAmount,
         tokenBDistributionFractionD18);
    } else if (claimData.totalBFeeForEpoch == 0) {
      amount += Math.mulFloor(claimData.epochDistributionAmount,
         tokenADistributionFractionD18);
    } else {
      // half the distribution is distributed to each prorata
      amount += Math.mulFloor(claimData.epochDistributionAmount,
         tokenADistributionFractionD18) / 2;
      amount += Math.mulFloor(claimData.epochDistributionAmount,
         tokenBDistributionFractionD18) / 2;`
    }
  }
}
```

However, `claimData.previousBinId` is not updated after every loop. This means that every `binId` is checked to be greater than 0.

If there is an array of `binIds = [10,20,30]`, then for every loop,

- `binId` = 10, 10 <= 0, no revert
- `binId` = 20, 20 <= 0, no revert
- `binId` = 30, 30 <= 0, no revert

Also, a user can call the same binId, eg [10,10,10], since `lpClaims[epoch][tokenId][pool][binId]`

is set to true only after `updateBalances()` is looped. A user cannot call `claimLp()` twice with the same binId, but he can call `claimLp()` with an array of duplicate binIds.

```
function claimLp(
  address recipient,
  uint256 tokenId,
  IMaverickV2Pool pool,
  uint32[] memory binIds,
  uint256 epoch
) public returns (uint256 amount) {
  // if lp is claiming and there is no disbursement yet, call distribute
     on VD
  if (disbursement[epoch][pool] == 0) votingDistributor.distribute(pool,
     epoch);
  if (disbursement[epoch][pool] == 0) revert NothingToClaim();

  address owner = position.checkAuthorized(msg.sender, tokenId);

  for (uint256 k; k < binIds.length; k++) {
    uint32 binId = binIds[k];
    if (lpClaims[epoch][tokenId][pool][binId]) revert AlreadyClaimed(owner
       , tokenId, pool, epoch, binId);
    lpReward.updateBalances(pool, true, tokenId, binId);`
@>  lpReward.updateBalances(pool, false, tokenId, binId);
  }

  // checks binId sorting
  amount = claimLpAmount(tokenId, pool, binIds, epoch);

  for (uint256 k; k < binIds.length; k++) {
    uint32 binId = binIds[k];
@>  lpClaims[epoch][tokenId][pool][binId] = true;
  }
}
```

Impact

BinIds are not checked for order.

Recommendation

Ensure `claimData.previousBinId` is updated after every loop, something like this:

```
function claimLpAmount(
  uint256 tokenId,
  IMaverickV2Pool pool,
  uint32[] memory binIds,
  uint256 epoch
) public view returns (uint256 amount) {
  ClaimData memory claimData;
  claimData.epochDistributionAmount = disbursement[epoch][pool];
  if (claimData.epochDistributionAmount == 0) return 0;

  claimData.totalAFeeForEpoch = lpReward.globalTracker(pool, true, epoch)
     ;
  claimData.totalBFeeForEpoch = lpReward.globalTracker(pool, false, epoch
     );

  claimData.previousBinId = 0;
  for (uint256 k; k < binIds.length; k++) {
    uint32 binId = binIds[k];
    if (binId <= claimData.previousBinId) revert BinIdsMustBeOrdered();
@>  claimData.previousBinId = binIds[k];
    uint256 tokenADistributionFractionD18 = lpReward.feeProrata(pool,
       true, tokenId, binId, epoch);
    uint256 tokenBDistributionFractionD18 = lpReward.feeProrata(pool,
       false, tokenId, binId, epoch);

    if (claimData.totalAFeeForEpoch == 0) {
      amount += Math.mulFloor(claimData.epochDistributionAmount,
         tokenBDistributionFractionD18);
    } else if (claimData.totalBFeeForEpoch == 0) {
      amount += Math.mulFloor(claimData.epochDistributionAmount,
         tokenADistributionFractionD18);
    } else {
      // half the distribution is distributed to each prorata
      amount += Math.mulFloor(claimData.epochDistributionAmount,
         tokenADistributionFractionD18) / 2;
      amount += Math.mulFloor(claimData.epochDistributionAmount,
         tokenBDistributionFractionD18) / 2;
    }
  }
}
```

Team Response

Fixed.

[L-01] Users Will Not Be Able To Vote On The Epoch Time Itself

Severity

Low Risk

Description

When `vote()` in VotingDistributor.sol is called, the function checks the current epoch time and calls `veToken.getPastTokenSupply()`.

On the current epoch time itself, if a user calls `vote()`, the function will revert with the `ERC5805FutureLookup` error. For example, if the current time is `1735171200`, which is 26 December 0000 UTC (isEpoch returns true), and a user staked some tokens beforehand and wants to vote, if the user calls `vote()` on `1735171200`, the function will revert. If the user calls on `1735171201` or `1735171199` (1 second after or before), the `vote()` will pass.

Location of Affected Code

```
function vote(IMaverickV2Pool[] memory voteTargets, uint256[] memory
    weights) external nonReentrant {
  uint256 epoch = currentEpoch();
  _inVotePeriodCheck(epoch);

  // code

  // get voting power of sender; includes any voting power delegated to
  // this sender; voting power is ve pro rata at beginning of vote
  // period
@> uint256 totalVote = veToken.getPastTotalSupply(epoch);
  votingPower = Math.divFloor(veToken.getPastVotes(msg.sender, epoch),
     totalVote);

  // @ code
}
```

Proof Of Concept

Copy and paste the test file and run `forge test -vv --mt test_f1`:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.25;

import {IMaverickV2Pool} from "@maverick/v2-common/contracts/interfaces/
   IMaverickV2Pool.sol";
import {IMaverickV2Factory} from "@maverick/v2-common/contracts/
   interfaces/IMaverickV2Factory.sol";
import {IERC20} from "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import {Ownable} from "@openzeppelin/contracts/access/Ownable.sol";
.
.
.
```

```
contract MockPool {
  IERC20 public tokenA;
  IERC20 public tokenB;
  constructor(IERC20 _a, IERC20 _b) {
    tokenA = _a;
    tokenB = _b;
  }

  function distributeFees(bool isTokenA) public returns (uint256
     protocolFee, IERC20 token) {
    token = isTokenA ? tokenA : tokenB;
    // protocolFee = 0;
    protocolFee = 1e18;
    token.transfer(msg.sender, protocolFee);
  }
}

contract VotingDistributorTest is BaseTest, Epoch {
  IMaverickV2Pool public p1;
  IMaverickV2Pool public p2;
  IMaverickV2Pool public p3;
  Ownable public factory = Ownable(this_);
  IVotingEscrowNft public ve;
  IVotingDistributor public vd;
  IPoolDistributor public pd;
  IOrphanDistributor internal od;
  ILpReward public lpReward;
  Rooster public r;
  uint256 internal epochLength;
  // tokenIds
  uint256 public t1;
  uint256 public t2;
  uint256 public t3;
  uint128 public a1 = 1e18;
  address public u1 = address(1000);
  address public u2 = address(2000);
  uint32 public b1 = 10;
  uint32 public b2 = 20;
  uint32 public b3 = 30;
  // epochs
  uint256 public e1;
  uint256 public e2;
  uint256 public e3;
  IVotingEscrow.Lockup public l1;
  IVotingEscrow.Lockup public l2;
  IVotingEscrow.Lockup public l3;
  IVotingEscrow.Lockup public l4;
  uint256 public preBalance;
  IMaverickV2Pool[] public pools;
  uint256[] public weights;
  uint32[] public binIds;
```

```
address public Alice = makeAddr("Alice");
address public Bob = makeAddr("Bob");
address public Cat = makeAddr("Cat");

function owner() external view returns (address) {
  return this_;
}

function isFactoryPool(IMaverickV2Pool) public pure returns (bool) {
  return true;
}

function setUp() public {
  // console.log("Block", block.timestamp);
  // Warp 26 December, 12 midnight UTC
  vm.warp(1735171200);
  deployTokens();
  p1 = IMaverickV2Pool(address(new MockPool(token1, token2)));
  p2 = IMaverickV2Pool(address(new MockPool(token2, token3)));
  p3 = IMaverickV2Pool(address(new MockPool(token1, token3)));

  r = new Rooster(this_);
  ve = new VotingEscrowNft(r, factory);
  vd = new VotingDistributor(IMaverickV2Factory(address(factory)),
     IERC5805(address(ve)), r);
  lpReward = new LpReward(this_);
  pd = new PoolDistributor(INft(address(ve)), lpReward, vd);
  od = new OrphanDistributor(vd);

  pools = new IMaverickV2Pool[](2);
  pools[0] = p1 < p2 ? p1 : p2;
  pools[1] = p1 < p2 ? p2 : p1;
  weights = new uint256[](2);
  binIds = new uint32[](2);
  (binIds[0], binIds[1]) = (b1, b2);


  VotingDistributor(address(vd)).setPoolDistributor(pd);
  VotingDistributor(address(vd)).setOrphanDistributor(od);
  VotingEscrow ve_ = new VotingEscrow(address(ve));
  VotingEscrowNft(address(ve)).initialize(IVotingEscrow(address(ve_)));
  r.setMinter(this_);
```

```
    epochLength = EPOCH_PERIOD;
    r.mint(u1, 10 * a1);
    r.mint(u2, 10 * a1);
    r.mint(this_, 1e30);
    r.approve(address(ve), 1e30);
    r.approve(address(vd), 1e30);
    // @audit skip 1 second from epoch start and stake tokens
    skip(1);
    (t1, l1) = ve.stake(u1, 10 weeks, 10e18);
    // @audit skip 2 epoch lengths ahead and vote
    skip(epochLength * 2 );

    e1 = currentEpoch();
    e2 = nextEpoch();
    e3 = nextEpoch() + epochLength;
    console.log(e1,e2,e3);
    VotingDistributor(address(vd)).setPoolVoteMultiplier(p1, 1e18);
    VotingDistributor(address(vd)).setPoolVoteMultiplier(p2, 1e18);
    token1.transfer(address(p1), 1e30);
    token2.transfer(address(p1), 1e30);
    token2.transfer(address(p2), 1e30);
    token3.transfer(address(p2), 1e30);
    token3.transfer(address(p3), 1e30);
    token1.transfer(address(p3), 1e30);
}

function test_f1() public {
    console.log("Current Block.Timestamp (START TEST):", block.timestamp);
    ////////////////////////////////////////
    lpReward.notifyBinLiquidity(p1, t1, b1, a1);
    lpReward.notifyBinLiquidity(p1, t1, b2, a1);
    vm.prank(address(p1));
    lpReward.registerFee(true, b1, 2 * a1);
    vm.prank(address(p1));
    lpReward.registerFee(false, b2, 2 * a1);
    lpReward.notifyBinLiquidity(p2, t2, b1, a1);
    lpReward.notifyBinLiquidity(p2, t2, b2, a1);
    vm.prank(address(p2));
    lpReward.registerFee(false, b2, 2 * a1);
    vm.prank(address(p2));
```

```
        lpReward.registerFee(true, b1, 2 * a1);
        // skip(epochLength / 100);
        vd.addDistributionBudgetCurrentEpoch(50e18);
        vd.addDistributionBudget(50e18, e1);
        ////////////////////////////////////////////

        // Add The Vote Incentive
        vd.addVoteIncentiveToCurrentEpoch(p1, r, a1 / 2);
        vd.addVoteIncentive(p1, r, a1 / 2, e1);

        console.log("User1 Votes:", ve.getVotes(u1));
        console.log("User 1 Past Votes:", ve.getPastVotes(u1, currentEpoch())
            );
        // Voting
        (weights[0], weights[1]) = (1, 1);
        // (weights[0]) = (1);
        console.log("Total Votes Before:", vd.getCheckpoint(currentEpoch()));
        vm.prank(u1);
        vd.vote(pools, weights);
        console.log("Total Votes After:", vd.getCheckpoint(currentEpoch()));
    }
}
```

- The time is 1735171200 and user 1 stakes 1 second into the epoch, 1735171201.
- 2 epochs later, 1736380801, the user calls vote. The test should pass

```
[PASS] test_f1() (gas: 2105736)
Logs:
  1736380800 1736985600 1737590400
  Current Block.Timestamp (START TEST): 1736380801
  User1 Votes: 2095677523489596760
  User 1 Past Votes: 2095677523489596760
  Total Votes Before: 0
  Total Votes After: 1000000000000000000
```

Change the values of `//@audit`, from `skip(epochLength * 2 );` to `skip(epochLength * 2 - 1 );`

- The time is 1735171200 and user 1 stakes 1 second into the epoch, 1735171201.
- 2 epochs later minus 1 second, 1736380800, the user calls vote. The test should fail.

```
Ran 1 test for test/TestFile.t.sol:VotingDistributorTest
[FAIL: ERC5805FutureLookup(1736380800 [1.736e9], 1736380800 [1.736e9])]
    test_f1() (gas: 1389128)
Logs:
  1736380800 1736985600 1737590400
  Current Block.Timestamp (START TEST): 1736380800
  User1 Votes: 2095677523489596760
```

If the value is changed to `epochLength * 2 - 2`, the test will pass again. This time the epoch is `1735776000` and the user voted at `1736380799` (1 second before the new epoch)

```
[PASS] test_f1() (gas: 2105736)
Logs:
  1735776000 1736380800 1736985600
  Current Block.Timestamp (START TEST): 1736380799
  User1 Votes: 2095677523489596760
  User 1 Past Votes: 2095677523489596760
  Total Votes Before: 0
  Total Votes After: 1000000000000000000
```

Impact

`vote()` will revert when `block.timestamp` is the epoch time.

Recommendation

If possible, don't allow anyone to vote on the epoch time itself.

Team Response

Acknowledged.

[I-01] BinId That Is 0 Cannot Be Claimed

Severity

Informational

Description

In the code, there is no mention of what `binIds` can be. The test files set `binIds` as [10,20] etc but it can also be 0.

If the `binId` is 0, `claimLp()` will revert when calling `claimLpAmount()` since `binId <= 0`.

```
function claimLpAmount(
  uint256 tokenId,
  IMaverickV2Pool pool,
  uint32[] memory binIds,
  uint256 epoch
) public view returns (uint256 amount) {
  ClaimData memory claimData;
  claimData.epochDistributionAmount = disbursement[epoch][pool];
  if (claimData.epochDistributionAmount == 0) return 0;

  claimData.totalAFeeForEpoch = lpReward.globalTracker(pool, true, epoch)
     ;
  claimData.totalBFeeForEpoch = lpReward.globalTracker(pool, false, epoch
     );

  claimData.previousBinId = 0;
  for (uint256 k; k < binIds.length; k++) {
    uint32 binId = binIds[k];
@>  if (binId <= claimData.previousBinId) revert BinIdsMustBeOrdered();

    // code
  }
}
```

Impact

User cannot claim their Lp on binId = 0.

Recommendation

Recommend skipping the `binId <= claimData.previousBinId` to check if `binId[0]` is actually 0.

Team Response

Acknowledged.

[I-02] Permalock In `VotingEscrowNft.sol` Does Not Actually Permanently Lock Voting Duration

Severity

Informational

Description

In `VotingEscrowNft.sol`, owners of the `tokenId` can call PermaLock to extend the duration of the lock to its max (4 years). This is an extension, however, and not a permanent lock. The user gets a fixed amount of votes once they lock, instead of a decaying amount, so permanently locking votes is not needed.

Location of Affected Code

File: ve33/contracts/VotingEscrowNft.sol#L161

```
function extendPermalock(uint256 tokenId) external returns (IVotingEscrow
    .Lockup memory newLockup) {
  if (!permalockAllowed[tokenId]) revert PermalockNotenabled();
  address owner = ownerOf(tokenId);
  return ve.extend(owner, tokenId, ve.MAX_STAKE_DURATION(), 0);
}
```

Impact

Ambiguity in function name.

Recommendation

Consider calling the function `maxLock` instead.

Team Response

Acknowledged.

[I-03] Weights Can Be Set At Any Arbitrary Amount

Severity

Informational

Description

The VotingDistributor.sol code states that:

```
 The protocol can set the vote weight of a given pool.  The typical
    weight
* will be one (1e18) for pools that are part of the loop.
```

When voting, the weights can actually be set at any amount by the caller, eg (1,1e30).

```
function vote(IMaverickV2Pool[] memory voteTargets, uint256[] memory
   weights) external nonReentrant {
```

In the test functions, weights are set as (1,1) for both the 2 pools.

If the weight is heavily skewed to a pool (1,1e30), the user may face a truncation to zero error.

Impact

`vote()` may revert, which will waste gas.

Recommendation

This is the caller's input, ensure that the caller understands the risk and does not set an unrealistic weight to each pool to prevent any reverts.

Team Response

Acknowledged.

[I-04] User Can Stake And Vote On The Same Epoch Itself

Severity

Informational

Description

When a user wants to vote, he needs to stake some Rooster tokens to get veRooster tokens.

During voting, a user cannot stake and vote in the same epoch, because their `pastTotalSupply` will not be registered yet until the new epoch starts.

However, if the user stakes exactly on the epoch time and votes one second later, he will be able to stake and vote in the same epoch.

Proof of Concept

Similar PoC to L-01, at the epoch time, call stake and one second later call vote. Epoch time is at 1735171200, so user1 stakes at 1735171200 and calls vote at 1735171201 (epoch is the same).

```
    (t1, l1) = ve.stake(u1, 1 weeks, 10e18);
    skip(1);
```

This test will pass with a stake and vote on the same epoch.

```
[PASS] test_f1() (gas: 2116034)
Logs:
  1735171200 1735776000 1736380800
  Current Block.Timestamp (START TEST): 1735171201
  User1 Votes: 193489366596254910
  User 1 Past Votes: 193489366596254910
  Total Votes Before: 0
  Total Votes After: 1000000000000000000
  User 1 Votes on Pool 1 Before Unstake: 500000000000000000
  User 1 Votes on Pool 2 Before Unstake: 500000000000000000
  Past Total Supply: 193489366596254910
```

If the user calls stake on epoch+1 seconds onwards and calls vote all the way to the new epoch timing, the function will not work, eg

```
    skip(1)
    (t1, l1) = ve.stake(u1, 1 weeks, 10e18);
    skip(1);
```

This will fail with

```
[FAIL: SenderHasNoVotingPower(0x0000000000000000000000000000000000000003e8,
    1735171200 [1.735e9])] test_f1() (gas: 1486542)
Logs:
  1735171200 1735776000 1736380800
  Current Block.Timestamp (START TEST): 1735171202
  User1 Votes: 193489369433776020
  User 1 Past Votes: 0
  Total Votes Before: 0
```

Recommendation

The impact seems to be negligible. Take note that a user can stake and vote on the same epoch if they staked exactly at the epoch time.

Team Response

Acknowledged.

shieldify

Thank you!