SHERLOCK

Security Review For EasyA Kickstart

Collaborative Audit Prepared For: **EasyA Kickstart**
Lead Security Expert(s): **0x3b**
Date Audited: **February 20 - February 22, 2026**

Introduction

This security review focused on Kickstart, which is the onchain launchpad for idea coins.

Buy into startup ideas with instant trading. Fair launch, no presale, community-driven coin launches.

Scope

Repository: EasyA-Tech/pump-2

Audited Commit: c675d0a0663806fa67023b02b37214b593f3cba7

Final Commit: c54929f8627496e3e8dbaf3d8461e581d1bf5253

Files:

- contracts/contracts/AerodromeLib.sol
- contracts/contracts/interfaces/IAerodrome.sol
- contracts/contracts/LPFeeCollector.sol
- contracts/contracts/PumpBondingCurve.sol
- contracts/contracts/PumpFactory.sol
- contracts/scripts/upgrade-cl-migration.ts
- contracts/scripts/utils/shared.ts

Final Commit Hash

c54929f8627496e3e8dbaf3d8461e581d1bf5253

Findings

Each issue has an assigned severity:

- High issues are directly exploitable security vulnerabilities that need to be fixed.
- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- Low/Info issues are non-exploitable, informational findings that do not pose a security risk or impact the system's integrity. These issues are typically cosmetic or related to compliance requirements, and are not considered a priority for remediation.

Issues Found

High	Medium	Low/Info
2	0	1

Issues Not Fixed and Not Acknowledged

High	Medium	Low/Info
0	0	0

Issue H-1: Pools can be DOSed [RESOLVED]

Source: https://github.com/sherlock-audit/2026-02-easya-feb-19th/issues/8

Summary

All pools can be DOSed from graduation, due to anyone being able to set custom `sqrtPriceX96`, which results in more than 5% slippage.

Vulnerability Detail

When graduating we create the pool and set our desired price with `sqrtPriceX96`

```
uint160 sqrtPriceX96 = _computeSqrtPriceX96(amount0Desired, amount1Desired);

try ICLFactory(clFactory).createPool(token0, token1, CL_TICK_SPACING, sqrtPriceX96)
↪  {}
catch {
    // Pool already exists -- proceed. PumpToken transfer restriction
    // guarantees it has no token liquidity, so attacker can't cause any damage.
}
```

After that we mint a position with the tokens, having max slippage of 5%

```
INonfungiblePositionManager.MintParams memory params =
↪  INonfungiblePositionManager.MintParams({
    token0: token0,
    token1: token1,
    tickSpacing: CL_TICK_SPACING,
    tickLower: CL_MIN_TICK,
    tickUpper: CL_MAX_TICK,
    amount0Desired: amount0Desired,
    amount1Desired: amount1Desired,
    amount0Min: (amount0Desired * 95) / 100,  // 5% slippage
    amount1Min: (amount1Desired * 95) / 100,  // 5% slippage
    recipient: lpRecipient,
    deadline: deadline,
    sqrtPriceX96: 0  // Pool already exists; skip createPool
});

(uint256 tokenId, uint128 liquidity, uint256 amount0Used, uint256 amount1Used) =
    INonfungiblePositionManager(nfpManager).mint(params);
```

However the issue here is that the pool can be create and set with custom `sqrtPriceX96` and 0 liquidity. This will result in. us entering the `catch` and then trying to mint a position with 5% slippage, where if the price results in more than the allowed slippage the TX will revert.

https://github.com/aerodrome-finance/slipstream/blob/main/contracts/core/CLFactory.sol#L70-L102

Impact

All pools can be DOSed from graduation, due to anyone being able to set custom `sqrtPriceX96`, which results in more than 5% slippage.

Tool Used

Manual Review

Recommendation

Calculate the `sqrtPriceX96` in initialize.

Issue H-2: All updated pools will have the wrong `predictedPool` [RESOLVED]

Source: https://github.com/sherlock-audit/2026-02-easya-feb-19th/issues/9

Summary

All updated pools will have the wrong `predictedPool`, which will enable anyone to transfer tokens to the aerodrome pool before it's deployed and skew the price and thus DOS `_graduate`

Vulnerability Detail

The main way to stop people from skewing the price before initial liquidity is added is by predicting the pool by not allowing any transfers to it:

```
function _update(address from, address to, uint256 value) internal override {
    // Check if this transfer is going to the predicted pool address
    if (to == predictedPool) {
        if (!IPumpBondingCurve(bondingCurve).graduated()) {
            revert TransferToPoolBeforeGraduation();
        }
    }

    super._update(from, to, value);
}
```

With the new code, graduation creates a CL (Slipstream) pool instead of a V2 pool. The prediction was updated accordingly:

```
address predictedPoolAddress = AerodromeLib.predictCLPoolAddress(
    _clFactory,
    poolImplementation,
    predictedTokenAddress,
    _weth,
    CL_TICK_SPACING
);
```

These produce completely different addresses, due to different factory and implementation

Since `PumpBondingCurve` is gonna be upgraded, but `PumpToken` is not, the upgrade only affects the graduation logic. Existing ungraduated tokens still have `predictedPool` set to the old V2 pool address.

After the upgrade, `_graduate()` creates a CL pool at a completely different address, one

the token has no transfer restriction on. This means an attacker can freely transfer tokens to the actual CL pool address before graduation, change the price and brick the graduation.

Note that this only effects old pools that are not graduate, but are upgraded.

Impact

Users can freely transfer ungraduated, but upgraded bonding curve tokens to the pools and manipulate the price.

Tool Used

Manual Review

Recommendation

Consider not upgrading any pools that have not yet graduated.

Issue L-1: `claimGaugeRewards` reward claim can be simplified [RESOLVED]

Source: https://github.com/sherlock-audit/2026-02-easya-feb-19th/issues/10

Summary

`claimGaugeRewards` reward claim can be simplified.

Vulnerability Detail

When claiming gauge rewards we do before and after math to calculate how much we've claimed, however instead of needing this math we could simply call `_sweepToken` to collect all rewards and transfer them to `_getFeeCollector`

```
function claimGaugeRewards(address pool) external {
    // ...
    uint256 balanceBefore = IERC20(rewardToken).balanceOf(address(this));

    ICLGauge(gauge).getReward(tokenId);

    uint256 earned = IERC20(rewardToken).balanceOf(address(this)) - balanceBefore;
    if (earned == 0) revert NoRewardsToClaim();

    address recipient = _getFeeCollector();
    IERC20(rewardToken).safeTransfer(recipient, earned);
```

Tool Used

Manual Review

Recommendation

Consider using `_sweepToken`, you can edit it to return the amount claimed in case if you want to keep emitting the claimed rewards event.

Disclaimers

Sherlock does not provide guarantees nor warranties relating to the security of the project.

Usage of all smart contract software is at the respective users' sole risk and is the users' responsibility.