Protocol Audit Report

Version 1.0

Vesko.io

December 21, 2024

2024-112-FLEX-PERPETUALS

Vesko.io

December 16.2024

Prepared by: Vesko Lead Auditors: - Veselin Vachkov

Issues found

Severity	Number of issues found
Medium	1
Low	1
Total	2

Findings

Low

L-1 Precision Loss

The function `getCurrentEpochTimestamp` in `FlexTradeCredits` calculates the start timestamp of the current epoch based on the `epochLength` parameter.

```
function getCurrentEpochTimestamp() public view returns (uint256
    epochTimestamp) {
     return (block.timestamp / epochLength) * epochLength;
}
```

The function exhibits precision loss due to integer division before multiplication.

Example Scenario

Input Parameters:

- `block.timestamp = 1672531199` (close to the Unix timestamp for a specific date/time)
- `epochLength = 300` (epoch length of 5 minutes)

Calculation:

1. `block.timestamp / epochLength = 1672531199 / 300 = 5575103` (integer division truncates the remainder)
2. `5575103 * epochLength = 1672530900`

Precision Loss: The resulting timestamp is `1672530900`, which is 299 seconds (just under 5 minutes) earlier than the actual block timestamp. This offset represents the truncation caused by integer division.

Estimated Loss with Reasonable Inputs

Assuming reasonable values for `epochLength` such as 1 minute (60 seconds), 5 minutes (300 seconds), or 1 hour (3600 seconds):

`epochLength`	Maximum Possible Loss (Seconds)
60	59
300	299
3600	3599

The maximum possible loss is always `epochLength - 1` seconds.

Recommended Mitigation

To avoid precision loss, perform the multiplication before the division:

```
function getCurrentEpochTimestamp() public view returns (uint256
    epochTimestamp) {
     return (block.timestamp * epochLength) / epochLength;
}
```

Medium

[M-1] Lack of slippage protection leads to loss of funds

Impact

Due to the lack of a slippage parameter, an MEV attacker could execute a sandwich attack on the swap, resulting in fewer output tokens being returned to the protocol than would otherwise be expected. Additionally, this issue can lead to significant losses due to price impact or slippage during periods of high market volatility or low liquidity.

Proof of Concept

```
function swapExactTokensForTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external returns (uint[] memory amounts);

function run(
        address _tokenIn,
        address _tokenOut,
        uint256 _amountIn
    ) external override returns (uint256 _amountOut) {
        // Check
        if (routeOf[_tokenIn][_tokenOut].length == 0) {
            revert AerodromeRouterSwitchCollateralExt_BadPath();
        }

        // Approve tokenIn to Router if needed
        ERC20 _tIn = ERC20(_tokenIn);
        if (_tIn.allowance(address(this), address(router)) < _amountIn)
            _tIn.safeApprove(address(router), type(uint256).max);

        uint256 _balanceBefore = ERC20(_tokenOut).balanceOf(address(
            this));

        //@audit slippage
        router.swapExactTokensForTokens(_amountIn, 0, routeOf[_tokenIn
            ][_tokenOut], address(this), block.timestamp);
        _amountOut = ERC20(_tokenOut).balanceOf(address(this)) -
            _balanceBefore;
```

```
29
30          // Transfer tokenOut back to msg.sender
31          ERC20(_tokenOut).safeTransfer(msg.sender, _amountOut);
32      }
```

The `swapExactTokensForTokens` function in the `run` method within `AerodromeDexter` hardcodes `0` for the `amountOutMin` parameter. This parameter is designed to enforce a minimum acceptable output amount during token swaps, protecting users against slippage. By hardcoding it to `0`, slippage protection is effectively disabled, allowing swaps to proceed regardless of unfavorable exchange rates at execution time.

Issue Location

- **File:** `/src/extensions/dexters/AerodromeDexter.sol`
- **Line:** 56

Risks of Hardcoding `amountOutMin` to `0`

Hardcoding `amountOutMin` as `0` introduces the following risks:

1. **Front-Running Attacks:**
    - An attacker with higher gas prices can manipulate the transaction by executing their trade first, reducing the return for subsequent trades.
    - This leads to a direct loss of funds for the protocol or the user executing the trade.
2. **Significant Slippage:**
    - During periods of high volatility or low liquidity, the output tokens may be significantly lower than expected.
3. **Loss of User Protection:**
    - With no minimum output enforced, users are not protected against unfavorable trades.

Example Scenario

Suppose the expected output for a token swap is 1000 tokens. Due to low liquidity or market manipulation, the actual output is reduced to 10 tokens. If a proper `amountOutMin` (e.g., 950 tokens) were set, the swap would not execute. However, with `0` hardcoded, the swap proceeds, causing a significant loss of funds.

Recommended Mitigation Steps

Introduce Default Slippage Limits

1. Set a reasonable default slippage tolerance (e.g., 1%-5%) for cases where users do not specify one explicitly.
2. Dynamically calculate `amountOutMin` based on the current exchange rate and the slippage tolerance.

Code Example

```
function run(
    address _tokenIn,
    address _tokenOut,
    uint256 _amountIn
) external override returns (uint256 _amountOut) {
    // Check
    if (routeOf[_tokenIn][_tokenOut].length == 0) {
        revert AerodromeRouterSwitchCollateralExt_BadPath();
    }

    // Approve tokenIn to Router if needed
    ERC20 _tIn = ERC20(_tokenIn);
    if (_tIn.allowance(address(this), address(router)) < _amountIn)
        _tIn.safeApprove(address(router), type(uint256).max);

    uint256 _balanceBefore = ERC20(_tokenOut).balanceOf(address(this));

    // Fetch current price or expected output using a price oracle or
       router estimate
    uint256 expectedAmountOut = router.getAmountsOut(_amountIn, routeOf
       [_tokenIn][_tokenOut])[1];

    // Calculate minimum acceptable output with 95% slippage protection
    uint256 amountOutMin = (expectedAmountOut * 95) / 100;
```

```
23
24      // Perform the swap with the calculated slippage protection
25      router.swapExactTokensForTokens(
26          _amountIn,
27          amountOutMin, // Dynamic slippage protection
28          routeOf[_tokenIn][_tokenOut],
29          address(this),
30          block.timestamp
31      );
32
33      _amountOut = ERC20(_tokenOut).balanceOf(address(this)) -
            _balanceBefore;
34
35      // Transfer tokenOut back to msg.sender
36      ERC20(_tokenOut).safeTransfer(msg.sender, _amountOut);
37  }
```

Links to Affected Code

- AerodromeDexter.sol#L56