RENASCENCE

G8Keep Audit Report

Version 2.0

Audited by:

MiloTruck

alexxander

bytes032

August 6, 2024

Contents

1 Introduction

1.1 About Renascence

Renascence Labs was established by a team of experts including HollaDieWaldfee, MiloTruck, alexxander and bytes032.

Our founders have a distinguished history of achieving top honors in competitive audit contests, enhancing the security of leading protocols such as Reserve Protocol, Arbitrum, MaiaDAO, Chainlink, Dodo, Lens Protocol, Wenwin, PartyDAO, Lukso, Perennial Finance, Mute and Taurus.

We strive to deliver tailored solutions by thoroughly understanding each client's unique challenges and requirements. Our approach goes beyond addressing immediate security concerns; we are dedicated to fostering the enduring success and growth of our partners.

More of our work can be found here.

1.2 Disclaimer

This report reflects an analysis conducted within a defined scope and time frame, based on provided materials and documentation. It does not encompass all possible vulnerabilities and should not be considered exhaustive.

The review and accompanying report are presented on an 'as-is' and 'as-available' basis, without any express or implied warranties.

Furthermore, this report neither endorses any specific project or team nor assures the complete security of the project.

1.3 Risk Classification

	Impact: High	**Impact: Medium**	**Impact: Low**
Likelihood: High	High	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

1.3.1 Impact

- High - Funds are **directly** at risk, or a **severe** disruption of the protocol's core functionality
- Medium - Funds are **indirectly** at risk, or **some** disruption of the protocol's functionality
- Low - Funds are **not** at risk

1.3.2 Likelihood

- High - almost certain to happen, easy to perform, or not easy but highly incentivized
- Medium - only conditionally possible or incentivized, but still relatively likely
- Low - requires stars to align, or little-to-no incentive

2 Executive Summary

2.1 About G8Keep

G8Keep is an ERC20 launchpad that facilitates the creation and sale of tokens through Uniswap V2.

Token creators fund the Uniswap V2 pair created for their tokens with initial liquidity, and in return, receive a pre-determined percentage of the token's total supply. Token creators also have the option of distributing their balance over a period of time through vesting.

The protocol also features a unique snipe protection mechanism, which penalizes users who attempt to purchase large amounts of the token immediately after it has been deployed.

2.2 Overview

Project	G8Keep
Repository	audit
Commit Hash	332379bc85ae...
Mitigation Hash	fef8ed5da227...
Date	29 July 2024 - 06 August 2024

2.3 Issues Found

Severity	**Count**
High Risk	1
Medium Risk	0
Low Risk	3
Informational	6
Total Issues	**10**

3 Findings Summary

ID	Description	Status
H-1	Snipe protection bypass by directly calling `UniswapV2Pair.swap()` with an inflated `_amount1Out`	Resolved
L-1	Token deployments with `_deployReserve = 0` where `g8keepFactory.minimumDeployVestTime > 0` will revert	Resolved
L-2	Skimmed balance from Uniswap pair is excluded from `pairedTokenMinimumLiquidity` check	Resolved
L-3	Zero approval in `setPairedTokenSettings()` and `setApprovalToUniswapRouter()` reverts for certain tokens	Resolved
I-1	Function `g8keepVester.claim()` should check if `vestedAmount > 0`	Resolved
I-2	Unused variables in functions `g8keepToken.maxSnipeProtectionBuyWithoutPenalty()` and `g8keepToken._checkLPForIncrease()`	Resolved
I-3	The transfer of `g8keepFees` in `g8keepToken._applyFees()` can be made conditional on `G8KEEP_FEE > 0`	Resolved
I-4	Missing visibility modifiers	Resolved
I-5	Difficulty of finding a valid `tokenSalt` when calling `deployToken()` depends on pair token address	Acknowledged
I-6	`thirdPartyLPAmount` does not adjust to changes in `reserve1` from swaps	Acknowledged

4 Findings

High Risk

[H-1] Snipe protection bypass by directly calling `UniswapV2Pair.swap()` with an inflated `_amount1Out`

Context: g8keepToken.sol#L236-L240

Description: When `transfer()` is called for the token, snipe protection reduces the actual amount of tokens sent from `amount`.

However, this implementation only works when swaps are performed through `UniswapV2Router02`, where the amount of `token1` received from the swap is calculated based on the amount of `token0` sent. `UniswapV2Pair` only checks if the remaining balance of `token0` and `token1` after the swap upholds the K invariant:

UniswapV2Pair.sol#L179-L183

```
{ // scope for reserve{0,1}Adjusted, avoids stack too deep errors
uint balance0Adjusted = balance0.mul(1000).sub(amount0In.mul(3));
uint balance1Adjusted = balance1.mul(1000).sub(amount1In.mul(3));
require(balance0Adjusted.mul(balance1Adjusted) >=
uint(_reserve0).mul(_reserve1).mul(1000**2), 'UniswapV2: K');
}
```

Therefore, an attacker can bypass snipe protection by directly calling `UniswapV2Pair.swap()` with `amount1Out` as an inflated value. `_applySnipeProtection()` would then reduce the amount of `token1` sent out such that the K invariant is not violated.

With reference to this graph, where:

- x is the amount of `token0` sent to swap.
- y is the amount of `token1` received from the swap.
- The blue line is the amount of `token1` swapped from `token0` by the router.
- The orange line is the maximum amount of `token1` that can be swapped for `token0` while satisfying K.

When the input token amount is less than the point at which users start overpaying, there is a region where swapping directly through the pair will give more `token1` than swapping through the router (ie. orange line > blue line), completing ignoring snipe protection.

The following POC demonstrates how swapping directly through the pair gives more `token1` with `x = 15`, which is the red line in the graph:

```
pragma solidity ^0.8.20;

import ”./g8keepBase.t.sol”;

contract SnipeProtectionBypass is g8keepBaseTest {
    g8keepToken public token;
    function setUp() public override {
        super.setUp();

        // Give this address some WETH
        address(WETH).call{ value: 1000 ether}("");
        ERC20(WETH).approve(address(_uniswapV2Router), type(uint256).max);

        // Deploy token with 98e18 reserve0 and 1000e18 reserve1
        token = g8keepToken(
            _g8keepFactory.deployToken{value: 100 ether}(
                100 ether,
                "TEST",
                "T",
                1000 ether,
                address(this),
                0,
                0,
                WETH,
                0,
                0 days,
                30 minutes,
                bytes32(uint256(1))
            )
        );
        _uniswapV2Pair = IUniswapV2Pair(token.UNISWAP_V2_PAIR());
    }

    function testSnipeProtectionBypass() external {
        // Take a snapshot of the current state
        uint256 snapshot = vm.snapshot();

        // User performs a regular swap with 15 ETH
        address[] memory path = new address[](2);
        path[0] = address(WETH);
        path[1] = address(token);

        _uniswapV2Router.swapExactTokensForTokensSupportingFeeOnTransferTokens(
            15 ether,
            0,
            path,
            address(this),
            block.timestamp
        );

        // He receives ~99.16 tokens
        uint256 received = token.balanceOf(address(this));
        console.log("Regular swap: %e", received);

        // Revert the state
        vm.revertTo(snapshot);
        assertEq(token.balanceOf(address(this)), 0);

        // User swaps directly through the pair, inflating amount1Out
        ERC20(WETH).transfer(address(_uniswapV2Pair), 15 ether);
        _uniswapV2Pair.swap(
            0,
            191e18,
            address(this),
            ""
        );

        // He receives ~25.21 more tokens
        console.log("Direct swap: %e", token.balanceOf(address(this)));
        console.log("Profit: %e", token.balanceOf(address(this)) - received);
    }

    receive() external payable { }
}
```

Similarly, 99.16 tokens can be bought for ~12 ETH by swapping directly through the pair.

Essentially, snipe protection is only effective when swapping through the router, but not when directly swapping through `UniswapV2Pair.swap()`. As such, for smaller input amounts, it's possible for users to perform swaps as if the pair was any other regular V2 pair and snipe protection didn't exist.

Recommendation: The team has fixed this issue by ensuring `amount1Out` in `_adjustAmountOut()` is not greater than the maximum amount that would have been calculated by the router during a swap.

G8Keep: Fixed in commit f1015b3 and fef8ed5.

Renascence: Verified, `amount1Out` cannot be specified to be more than what the router would have calculated based on `amount0In`, so inflating `amount1Out` while swapping directly through the pair is no longer possible.

Note that with this fix, `skim()` cannot be called when a pair has excess `token1` while snipe protection is active. In `skim()`, `token1` is transferred after `token0`, so `balance0 - reserve0` will always be `0` in `_adjustAmountOut()`. Therefore, trying to transfer any `token1` out in `skim()` will revert.

G8Keep: We're okay with people not being able to skim during snipe protection.

Low Risk

[L-1] Token deployments with `_deployReserve = 0` where `g8keepFactory.minimumDeployVestTime > 0` will revert

Context:

- g8keepFactory.sol#L122
- g8keepToken.sol#L123-L132
- g8keepVester.sol#L49

Description: When `g8keepFactory.minimumDeployVestTime` is set to a value greater than 0, it is required in `g8keepFactory.deployToken()` that the parameter `_deployVestTime` is greater than or equal to `minimumDeployVestTime`.

An issue arises if `g8keepFactory.deployToken()` is called with `_deployReserve = 0`, i.e., the deployer won't reserve and vest tokens. In the constructor of the `g8keepToken`, `tokensToDeployer` will be equal to zero and `_deployerVestTime` will be greater than 0.

This configuration will attempt to vest 0 tokens for the supplied `_deployerVestTime`, which will always revert since `g8keepVester.deploymentVest()` reverts with `InvalidAmount` when `_tokensToDeployer = 0`.

```
# g8keepToken.sol

if (_deployerVestTime == 0) {
    // Transfer deployer tokens to deployer
    _balances[_deployer] = tokensToDeployer;
    emit Transfer(address(0), _deployer, tokensToDeployer);
} else {
    address g8keepVester = IG8keepFactory(msg.sender).g8keepTokenVesting();
    _balances[g8keepVester] = tokensToDeployer;
    emit Transfer(address(0), g8keepVester, tokensToDeployer);

    IG8keepDeployerVesting(g8keepVester).deploymentVest(_deployer, tokensToDeployer,
    _deployerVestTime);
```

Recommendation:

```
@@ -120,16 +120,18 @@ contract g8keepToken is Ownable {
         _allowances[msg.sender][_uniswapV2Router] = type(uint256).max;
         emit Approval(msg.sender, _uniswapV2Router, type(uint256).max);

-        if (_deployerVestTime == 0) {
-            // Transfer deployer tokens to deployer
-            _balances[_deployer] = tokensToDeployer;
-            emit Transfer(address(0), _deployer, tokensToDeployer);
-        } else {
-            address g8keepVester = IG8keepFactory(msg.sender).g8keepTokenVesting();
-            _balances[g8keepVester] = tokensToDeployer;
-            emit Transfer(address(0), g8keepVester, tokensToDeployer);
-
-            IG8keepDeployerVesting(g8keepVester).deploymentVest(_deployer,
tokensToDeployer, _deployerVestTime);
+        if (tokensToDeployer > 0) {
+            if (_deployerVestTime == 0) {
+                // Transfer deployer tokens to deployer
+                _balances[_deployer] = tokensToDeployer;
+                emit Transfer(address(0), _deployer, tokensToDeployer);
+            } else {
+                address g8keepVester =
IG8keepFactory(msg.sender).g8keepTokenVesting();
+                _balances[g8keepVester] = tokensToDeployer;
+                emit Transfer(address(0), g8keepVester, tokensToDeployer);
+
+                IG8keepDeployerVesting(g8keepVester).deploymentVest(_deployer,
tokensToDeployer, _deployerVestTime);
+            }
```

G8Keep: Fixed in commit 75e704a.

Renascence: The recommendation has been implemented.

[L-2] Skimmed balance from Uniswap pair is excluded from `pairedTokenMinimumLiquidity` check

Context: g8keepFactory.sol#L166-L195

Description: In `g8keepFactory.deployToken()`, the minimum `_initialLiquidity` check is performed before skimming excess `_pairedToken` liquidity from the Uniswap V2 pair:

```
if (_initialLiquidity < pairedTokenMinimumLiquidity[_pairedToken]) revert
NotEnoughInitialLiquidity();

// ...

if (IERC20(_pairedToken).balanceOf(pairAddress) > 0) {
    uint256 balanceBefore = IERC20(_pairedToken).balanceOf(address(this));
    IUniswapV2Pair(pairAddress).skim(address(this));
    uint256 balanceAfter = IERC20(_pairedToken).balanceOf(address(this));
    if (balanceAfter > balanceBefore) {
        unchecked {
            _initialLiquidity = _initialLiquidity + balanceAfter - balanceBefore;
        }
    }
}
```

Therefore, the amount of `_pairedToken` sent by the deployer MUST be more than `pairedTokenMinimumLiquidity[_pairedToken]`. If initial liquidity sent by the deployer is less than `pairedTokenMinimumLiquidity`, but becomes more when the skimmed balance is added, the function will still revert (ie. skimmed balance from the pair is excluded from this check).

Recommendation: Perform the check after excess liquidity has been skimmed from the pair:

```diff
- if (_initialLiquidity < pairedTokenMinimumLiquidity[_pairedToken]) revert
NotEnoughInitialLiquidity();

  // ...

  if (IERC20(_pairedToken).balanceOf(pairAddress) > 0) {
      uint256 balanceBefore = IERC20(_pairedToken).balanceOf(address(this));
      IUniswapV2Pair(pairAddress).skim(address(this));
      uint256 balanceAfter = IERC20(_pairedToken).balanceOf(address(this));
      if (balanceAfter > balanceBefore) {
          unchecked {
              _initialLiquidity = _initialLiquidity + balanceAfter - balanceBefore;
          }
      }
  }
+ if (_initialLiquidity < pairedTokenMinimumLiquidity[_pairedToken]) revert
NotEnoughInitialLiquidity();
```

G8Keep: Fixed in commit 26b5630.

Renascence: The recommendation has been implemented.

[L-3] Zero approval in `setPairedTokenSettings()` and `setApprovalToUniswapRouter()` reverts for certain tokens

Context:

- g8keepFactory.sol#L232
- g8keepFactory.sol#L320

Description: When the protocol owner calls `g8keepFactory.setPairedTokenSettings()` with `allowed = false` to remove a `pairedToken` from the whitelist, the function performs a zero value approval:

```
SafeTransferLib.safeApprove(pairedToken, UNISWAP_V2_ROUTER, 0);
```

Similarly, `setApprovalToUniswapRouter()` also calls `approve()` with zero value when `approved[i]` is `false`:

```
SafeTransferLib.safeApprove(tokenAddress, UNISWAP_V2_ROUTER, 0);
```

However, calling `approve()` with zero value will revert for certain tokens, such as BNB:

```
function approve(address _spender, uint256 _value)
    returns (bool success) {
    if (_value <= 0) throw;
    allowance[msg.sender][_spender] = _value;
    return true;
}
```

Therefore, if BNB is ever whitelisted as an accepted pair token, `setPairedTokenSettings()` can never be called with `allowed = false` to remove it from the whitelist.

Recommendation: A hacky fix would be to use `SafeTransferLib.safeApproveWithRetry()` and set the allowance to `1` instead of `0`:

```
- SafeTransferLib.safeApprove(pairedToken, UNISWAP_V2_ROUTER, 0);
+ SafeTransferLib.safeApproveWithRetry(pairedToken, UNISWAP_V2_ROUTER, 1);
```

Otherwise, document that BNB and tokens that revert on zero approval are not supported pair tokens.

G8Keep: Fixed in commit 5a98222.

Renascence: Verified, the function was modified to use `safeApproveWithRetry()` and the owner specifies the allowance amount.

Informational

[I-1] Function `g8keepVester.claim()` should check if `vestedAmount > 0`

Context:

- g8keepVester.sol#L80-L105

Description: In cases where `g8keepVester._vested()` returns 0 tokens due to rounding down, `g8keepVester.claim()` will update `vesting.lastClaim()` even if no tokens were claimed. Consider the following example:

- `vestingPeriod` is 1 year, which is 31536000 seconds, and `vestingAmount` is 2e6.
- The user calls `claim()` every 12 seconds, therefore, `timeSinceLastClaim` is 12 seconds. Then `vestedAmount = 2e6 * 12 / 31536000 = 0.76`, which rounds down to 0.
- If the user keeps calling `claim()` every block, they won't receive vested funds until `block.timestamp >= vestingEnd`.

Recommendation:

```diff
@@ -39,6 +39,7 @@ contract g8keepVester is IG8keepDeployerVesting {
     error InvalidCaller();
     error InvalidDuration();
     error InvalidVestingId();
+    error NoTokensVested();

     function deploymentVest(address _deployer, uint256 _tokensToDeployer, uint256
     _vestTime)
         external
@@ -88,6 +89,10 @@ contract g8keepVester is IG8keepDeployerVesting {
             revert InvalidCaller();
         }

+        if (vestedAmount == 0) {
+            revert NoTokensVested();
+        }
+
```

G8Keep: Fixed in commit 6756c91.

Renascence: The recommendation has been implemented.

[I-2] Unused variables in functions `g8keepToken.maxSnipeProtectionBuyWithoutPenalty()` and `g8keepToken._checkLPForIncrease()`

Context:

- g8keepToken.sol#L148
- g8keepToken.sol#L313

Description: The variable `uint256 reserve1In` is unused in functions `g8keepToken.maxSnipeProtectionBuyWithoutPenalty()` and `g8keepToken._checkLPForIncrease()`.

Recommendation:

```
    uint256 _cachedLPReserve1 = cachedLPReserve1;
-   uint256 reserve1In;
```

```
    if (_lastLPTotalSupply != 0) {
-       uint256 reserve1In;
```

G8Keep: Fixed in commit 1f4a3cb.

Renascence: The recommendation has been implemented.

[I-3] The transfer of `g8keepFees` in `g8keepToken._applyFees()` can be made conditional on `G8KEEP_FEE > 0`

Context:

- g8keepToken.sol#L274-L276

Description: The calculation and transfer of `g8keepFees` in `g8keepToken._applyFees()` will be redundant in the case where the `g8keepFee` is set to 0 through `g8keepFactory.setG8keepFee()`.

Consider the computation and transfer of the `g8keepFee` to occur only if `G8KEEP_FEE > 0`, similarly to how the transfer of `deployerFees` is conditional on `if (deployerFee > 0)`.

```
# g8keepToken.sol

uint256 deployerFees = 0;
if (deployerFee > 0) {
    deployerFees = (amount * deployerFee) / BPS;

    address _treasuryWallet = treasuryWallet;
    _balances[_treasuryWallet] += deployerFees;
    emit Transfer(from, _treasuryWallet, deployerFees);
}

uint256 g8keepFees = (amount * G8KEEP_FEE) / BPS;
_balances[G8KEEP] += g8keepFees;
emit Transfer(from, G8KEEP, g8keepFees);
toAmount = amount - deployerFees - g8keepFees;
```

Recommendation:

```
-   uint256 g8keepFees = (amount * G8KEEP_FEE) / BPS;
-   _balances[G8KEEP] += g8keepFees;
-   emit Transfer(from, G8KEEP, g8keepFees);
+   uint256 g8keepFees = 0;
+   if (G8KEEP_FEE > 0) {
+       g8keepFees = (amount * G8KEEP_FEE) / BPS;
+       _balances[G8KEEP] += g8keepFees;
+       emit Transfer(from, G8KEEP, g8keepFees);
+   }
```

G8Keep: Fixed in commit bfd1398.

Renascence: The recommendation has been implemented.

[I-4] Missing visibility modifiers

Context: g8keepToken.sol#L36-L38

Description: The following state variables in `g8keepToken` have their visibility missing. Consider specifying them as `private`:

```
- uint256 lastLPTotalSupply;
- uint112 thirdPartyLPAmount;
- uint112 cachedLPReserve1;
+ uint256 private lastLPTotalSupply;
+ uint112 private thirdPartyLPAmount;
+ uint112 private cachedLPReserve1;
```

G8Keep: Fixed in commit 8a44b38.

Renascence: The recommendation has been implemented.

[I-5] Difficulty of finding a valid `tokenSalt` when calling `deployToken()` depends on pair token address

Context: g8keepToken.sol#L85-L91

Description: When deploying g8keep tokens through `g8keepFactory.deployToken()`, the token contract's address must be greater than the pair token's address:

```
//Revert if deployment address is less than the paired token address
//Token must be "token1" in the v2 pool so that snipe protection can
//evaluate the paired token balance of the pool while transferring
//this token to the buyer.
if (uint160(address(this)) < uint160(_pairedToken)) {
    revert InvalidTokenAddress();
}
```

However, if the `_pairedToken` address happens to be quite large, the chances of `address(this)` being larger than `_pairedToken` becomes quite small.

For example, the USDT address is `0xdac17f958d2ee523a2206206994597c13d831ec7`. Assuming address derivation is purely random, there's an ~85.5% chance that the `g8keepToken` token address is smaller than USDT - there's only a 15% chance that `deployToken()` doesn't revert when called with USDT as the pair token.

As such, if the `_pairedToken` address is extremely large, Dapps integrating directly with the g8keep contracts will have to spend more compute to look for a valid `tokenSalt`.

Recommendation: Consider documenting that external integrations have to look for a valid `tokenSalt` for token deployments, and its difficulty is based on the `_pairedToken` address.

G8Keep: Acknowledged. The g8keep dapp has implemented address mining to produce valid salts based on the paired token.

Renascence: Acknowledged.

[I-6] `thirdPartyLPAmount` does not adjust to changes in `reserve1` from swaps

Context: g8keepToken.sol#L301-L326

Description: In `_checkLPForIncrease()`, `thirdPartyLPAmount` is only updated when `totalSupply()` of the Uniswap V2 pair changes. As such, the amount of liquidity belonging to external parties will not change when `reserve1` increases/decreases from a swap.

For example:

- A third-party calls `UniswapV2Pair.mint()` to add liquidity.
- A user calls `swap()` to buy `token1` from the pair. This causes `reserve1` to decrease, so the amount of `token1` liquidity belonging to the third-party is reduced.
- However, `_checkLPForIncrease()` doesn't update `thirdPartyLPAmount` as liquidity `totalSupply()` didn't change.

`thirdPartyLPAmount` will no longer accurately reflect the amount of `reserve1` that belongs to external LPs. As such, in `_adjustAmountOut()`, `balance1 - thirdPartyLPAmount` does not match the remaining amount of `reserve1` that the g8keep protocol owns.

The following POC demonstrates how `balance1 - thirdPartyLPAmount` doesn't match the amount of `reserve1` leftover, calculated based on the proportion of LP tokens owned:

```
pragma solidity ^0.8.20;

import ”./g8keepBase.t.sol”;

contract ThirdPartyLpAmountTest is g8keepBaseTest {
    function setUp() public override {
        super.setUp();

        // Give this address some WETH
        address(WETH).call{ value: 1000 ether}("");
        ERC20(WETH).approve(address(_uniswapV2Router), type(uint256).max);
        _g8keepToken.approve(address(_uniswapV2Router), type(uint256).max);
    }

    function testThirdPartyLPAmountInaccurate() external {
        // Buy some g8keep tokens
        address[] memory path = new address[](2);
        path[0] = address(WETH);
        path[1] = address(_g8keepToken);

        _uniswapV2Router.swapExactTokensForTokensSupportingFeeOnTransferTokens(
            0.05 ether,
            0,
            path,
            address(this),
            block.timestamp
        );
        uint256 received = _g8keepToken.balanceOf(address(this));

        // Deposit g8keep tokens as liquidity
        (uint256 amountA, uint256 amountB, uint256 liquidity) =
        _uniswapV2Router.addLiquidity(
            path[0],
            path[1],
            1000 ether,
            received,
            0,
            0,
            address(this),
            block.timestamp
        );

        // A swap occurs to buy more g8keep tokens
        _uniswapV2Router.swapExactTokensForTokensSupportingFeeOnTransferTokens(
            0.05 ether,
            0,
            path,
            address(this),
            block.timestamp
        );

        // Calculate amount of reserve1 remaining in the protocol
        (, uint256 _reserve1, ) = _uniswapV2Pair.getReserves();
        ERC20 pair = ERC20(address(_uniswapV2Pair));
        uint256 factoryReserve1 = pair.balanceOf(address(_g8keepFactory)) * _reserve1
        / pair.totalSupply();
        console.log(”reserve1 owned by protocol: %e”, factoryReserve1);

        // Amount of reserve1 remaining calculated in _adjustAmountOut()
        uint256 calculatedReserve1 = _reserve1 - _g8keepToken.thirdPartyLPAmount();
        console.log(”Calculated reserve1: %e”, calculatedReserve1);
        console.log(”Difference: %e”, factoryReserve1 - calculatedReserve1);
    }

    receive() external payable { }
}
```

Output:

```
reserve1 owned by protocol: 9.2987734067888731166996293e25
Calculated reserve1: 9.1465083356692324675183717e25
Difference: 1.522650711196406491812576e24
```

Since `thirdPartyLPAmount` is outdated (ie. greater than it should be), `adjustedBalance1` in snipe protection is under-estimated, causing snipe protection penalization to be more than it should be.

Recommendation: Calculate `thirdPartyLPAmount` directly in `_adjustAmountOut()` as such:

```
thirdPartyLPAmount = _reserve1 * (pair.totalSupply() - pair.balanceOf(G8KEEP)) /
pair.totalSupply();
```

G8Keep: Our inclination is to stick with the current methodology, while understanding that a user may be buying a portion of the tokens that others added to the LP. We feel that limiting the ability for someone to buy and re-LP to grow their LP position would be less ideal and given that snipe protection is intended to only last for a short period of time, the limitations are acceptable.

Renascence: Acknowledged.