SHERLOCK

Security Review For Livo Labs

Collaborative Audit Prepared For:	**Livo Labs**
Lead Security Expert(s):	**samuraii77**
	ZeroTrust
Date Audited:	**January 23 - January 25, 2026**

Introduction

LivoLaunchpad is an ERC20 token launchpad in ethereum mainnet, where users can deploy tokens at minimal gas cost and zero fees.

This audit report covers an update to allow users to deploy taxable tokens via Livo Launchpad leveraging Uniswap V4 hook to perform the tax collection on swaps.

Scope

Repository: LivoLaunchpad/livo-contracts

Audited Commit: e14de330bf8ecaad285eb3f69f7740e04a15ccf1

Final Commit: 8a105bd07f68458d5dc46f8db78b2de78aa43eb6

Files:

- src/graduators/LivoGraduatorUniswapV4.sol
- src/hooks/LivoSwapHook.sol
- src/interfaces/ILivoTaxableTokenUniV4.sol
- src/LivoLaunchpad.sol
- src/tokens/LivoTaxableTokenUniV4.sol

Final Commit Hash

8a105bd07f68458d5dc46f8db78b2de78aa43eb6

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
0	2	6

Issues Not Fixed and Not Acknowledged

High	Medium	Low/Info
0	0	0

Issue M-1: `createCustomToken` bypasses component whitelisting but still relies on `whitelistedComponents` for graduation settings, potentially deploying permanently non-functional tokens [RESOLVED]

Source: https://github.com/sherlock-audit/2026-01-livo-launchpad-jan-23rd/issues/7

Summary

`createCustomToken()` is meant to let admins deploy tokens without requiring a `(implementation, bondingCurve, graduator)` tuple to be whitelisted. However, the deployment path still pulls `GraduationSettings` from `whitelistedComponents` inside `_createToken()`. If the tuple was never configured, all graduation settings default to `0`, resulting in `maxEthReserves == 0` and effectively preventing any buys; the token can never collect ETH and therefore can never graduate.

Vulnerability Detail

`createCustomToken()` skips the whitelist check performed in `createToken()`, but both functions call the same `_createToken()` implementation.

Inside `_createToken()`, the contract reads:

- `ethGraduationThreshold`
- `maxExcessOverThreshold`
- `graduationEthFee`

from `whitelistedComponents[implementation][bondingCurve][graduator]` and stores them into `tokenConfigs[token]`.

If the tuple is not configured in `whitelistedComponents`, these values are all `0` by default. This makes:

```
maxEthReserves = ethGraduationThreshold + maxExcessOverThreshold = 0
```

Later, buys enforce:

```
tokenState.ethCollected + ethForReserves <= tokenConfig.maxEthReserves()
```

Since `ethForReserves > 0` for any `msg.value > 0`, the buy path always reverts. With no buys possible, `ethCollected` can never increase, and graduation (which only triggers from `buyTokensWithExactEth`) can never happen.

Impact

- Admin can deploy a token via `createCustomToken()` that is permanently unusable on the launchpad (cannot be bought, cannot collect ETH, cannot graduate).
- The issue is a configuration footgun: it is easy to assume `createCustomToken()` is fully independent from `whitelistedComponents`, but it is not.
- If deployed in production, it can cause failed launches, wasted deployment gas, and user confusion/reputational risk.

Tool Used

Manual Review

Recommendation

- Make `createCustomToken()` independent of `whitelistedComponents` by taking explicit `GraduationSettings`/threshold parameters and writing them into `tokenConfigs[token]` during deployment; or
- Add an explicit guard for custom deployments (e.g., revert if `ethGraduationThreshold == 0` in `_createToken()`), and provide a separate admin-only way to set per-token graduation parameters before enabling trading.

Issue M-2: Graduation DoS via forcing perfect liquidity mint. [RESOLVED]

Source: https://github.com/sherlock-audit/2026-01-livo-launchpad-jan-23rd/issues/11

Summary

Graduation DoS via forcing perfect liquidity mint.

Vulnerability Detail

The graduation works by adding token + ETH liquidity which under regular conditions would use up ~80% of the ETH liquidity and 100% of the tokens. Then, we add single-sided liquidity of ETH to use up the rest of the ETH. This is how the ETH and tokens for graduation are computed:

```
uint256 ethCollected = tokenState.ethCollected;
uint256 ethForGraduation = ethCollected - tokenConfig.graduationEthFee;

// ...

uint256 tokensForGraduation = token.balanceOf(address(this)) - tokensForOwner;

tokenConfig.graduator.graduateToken{value: ethForGraduation}(tokenAddress,
 ↪  tokensForGraduation);
```

The issue is that the tokens for graduation are computed using `balanceOf`. Thus, the attacker can:

1. Buy tokens for ~0.13 ETH.
2. Token is about to graduate, final buy happens.
3. Send the tokens bought from step 1 in the launchpad contract (as a frontrun to step 2).
4. This makes the first liquidity mint a perfect one, all ETH and tokens are used up (actual requirement is for all ETH to be used up, doesn't necessarily have to be an absolutely perfect mint as long as ETH becomes the limiting asset).
5. `liquidity2` is computed as 0 due to `remainingEth` being 0.
6. We add 0 liquidity which reverts in UniswapV4 here:

```
if (liquidityDelta == 0) {
    // disallow pokes for 0 liquidity positions
    if (liquidity == 0) CannotUpdateEmptyPosition.selector.revertWith();
}
```

Impact

Graduation DoS.

Tool Used

Manual Review

Recommendation

Either only add the liquidity if above 0 or use code similar to the one in `_availableTokensForPurchase()`.

Issue L-1: Graduation can be DoSed by griefers. [RESOLVED]

Source: https://github.com/sherlock-audit/2026-01-livo-launchpad-jan-23rd/issues/6

Summary

Graduation can be DoSed by griefers.

Vulnerability Detail

Upon graduation, ETH and tokens for graduation are sent to the graduator contract. Then, liquidity is added which uses the `tokenAmount` provided + the `ethValue`. The issue is here:

```
uint256 tokenBalanceAfterDeposit = token.balanceOf(address(this));
uint256 tokensDeposited = tokenAmount - tokenBalanceAfterDeposit;
```

We get all of the token balance and subtract it from the `tokenAmount`. Thus, if the token balance after the liquidity mint is more than what was added as liquidity, we will revert.

A griefer can:

1. Buy a bit more than 190M tokens via the bonding curve. This will cost him ~0.625 ETH if he is the first buyer.
2. Wait until a graduation purchase happens.
3. Directly send the tokens bought from step 1 to the graduator contract.
4. Upon graduation, `tokenAmount` is ~190M tokens based on configurations and since `tokenBalanceAfterDeposit` is equal to the tokens sent in step 3, then an underflow error will happen.

Impact

Graduation DoS, griefing attack.

Tool Used

Manual Review

Recommendation

Implement logic similar to how `remaningEth` is computed.

Issue L-2: `LivoTaxableTokenUniV4._update()` triggers an on-chain swap, enabling MEV tax extraction and unpredictable transfer gas [RESOLVED]

Source: https://github.com/sherlock-audit/2026-01-livo-launchpad-jan-23rd/issues/8

Summary

`LivoTaxableTokenUniV4` performs a Uniswap V4 swap from inside `_update()` (i.e., during normal ERC20 transfers) once certain conditions are met. Because any user can trigger `_update()` via a simple `transfer`, attackers can (1) force the tax-swap to execute at an attacker-chosen time with no slippage protection, making it easy to sandwich and reduce tax proceeds, and (2) cause ordinary users to unexpectedly pay very high gas for what they believe is a simple transfer.

Vulnerability Detail

In `src/tokens/LivoTaxableTokenUniV4.sol`, `_update()` calls `_swapAccumulatedTaxes()` after **any** transfer when:

- token is graduated (`graduationTimestamp != 0`)
- not already swapping (`_inSwap == false`)
- not accumulating taxes to self (`to != address(this)`)
- pool manager is locked / idle (`TransientStateLibrary.isUnlocked(UNIV4_POOL_MANAGER) == false`)
- this contract holds accumulated tax tokens (`balanceOf(address(this)) > 0`)
- and either the balance crosses the threshold or the tax period ended

Because `transfer()` is permissionless, an attacker can:

1) Monitor for the moment the tax-swap becomes triggerable, then send a tiny transfer to force `_swapAccumulatedTaxes()` to run at a precise block/ordering position.
2) Sandwich the swap. `_swapAccumulatedTaxes()` hardcodes `amountOutMinimum = 0` and uses `TAKE_ALL` with `minAmount = 0`, so the swap accepts any execution price.

Separately, cause ordinary users to unexpectedly pay very high gas for what they believe is a simple transfer.

Impact

1) **MEV sandwich / timing manipulation drains tax value**

- The contract's tax-sale is predictable, can be triggered by anyone, and has **no slippage protection**.
- MEV searchers can front-run to worsen price and back-run to restore it, extracting value that would otherwise go to the token owner as tax proceeds.

2) **Unexpected high gas for ordinary transfers**

- Any normal `transfer` (even unrelated to Uniswap) can suddenly execute a full router swap.
- Users can be surprised by large gas usage, and transfers may fail if the gas limit is not high enough.

Code Snippet

`src/tokens/LivoTaxableTokenUniV4.sol:209`

`src/tokens/LivoTaxableTokenUniV4.sol:263`

Tool Used

Manual Review

Recommendation

Remove (or significantly reduce) the automatic swap trigger from `_update()` and instead add an explicit function callable **only by the token owner** to execute the tax swap when the same conditions are met, with **slippage protection**.

Issue L-3: Missing zero-address validation for `treasury` and `tokenOwner` can burn fees and/or brick graduation (especially Uniswap V4) [RESOLVED]

Source: https://github.com/sherlock-audit/2026-01-livo-launchpad-jan-23rd/issues/9

Summary

`LivoLaunchpad` allows setting `treasury` and per-token `tokenOwner` to `address(0)` via admin/owner flows. This can (1) silently burn collected ETH fees by transferring to the zero address, and (2) cause Uniswap V4 graduation to mint/assign the second liquidity position to `address(0)` (or revert), potentially making graduation/lockup/position ownership behave in an unacceptable or irreversible way. Additionally, setting `tokenOwner` to `address(0)` breaks the system's own "token exists" predicate (`TokenDataLib.exists()`), which can cascade into denial-of-service for token operations and/or prevent graduation.

Vulnerability Detail

1) `treasury` can be set to `address(0)`

`setTreasuryAddress(address recipient)` writes `treasury = recipient` with no validation.

Consequences:

- `collectTreasuryFees()` uses `_transferEth(treasury, amount)`; if `treasury == address(0)`, ETH is transferred to the zero address (effectively burned).
- `LivoGraduatorUniswapV4.graduateToken()` reads `ILivoLaunchpad(LIVO_LAUNCHPAD).treasury()` and uses it as the recipient for the second (single-sided ETH) Uniswap V4 position. If `treasury == address(0)`, the downstream position manager may revert (common for ERC721 `mint/safeMint` to `address(0)`) or may create a position with an unusable owner/recipient, depending on Uniswap V4 implementation details.

2) `tokenOwner` can be set to `address(0)` after token creation

While `_createToken()` prevents `tokenOwner == address(0)` at creation time, ownership can later be updated via:

- `transferTokenOwnership(address token, address newTokenOwner)`
- `communityTakeOver(address token, address newTokenOwner)`
- `_transferTokenOwnership(address token, address newTokenOwner)` (internal sink)

None of these paths prevent `newTokenOwner == address(0)`.

Consequences:

- `TokenDataLib.exists()` uses `config.tokenOwner != address(0)` as the existence predicate. Setting `tokenOwner` to zero makes an already-created token appear to not exist, which can cause core launchpad functions to revert with `InvalidToken()` or behave inconsistently.
- During graduation, `LivoLaunchpad._graduateToken()` transfers `OWNER_RESERVED_SUPPLY` to `tokenConfig.tokenOwner`. With `tokenOwner == address(0)`, this will either revert (typical OZ ERC20 transfer-to-zero guard) and brick graduation, or "burn" the reserved supply if the token implementation allows transfer to the zero address.
- Any fee distribution logic in downstream components that consult `launchpad` for `tokenOwner` may send funds to `address(0)` (lost/burned) or revert, depending on implementation.

Additionally, `communityTakeOver()` currently lacks an explicit `tokenConfigs[token].exists()` check, allowing admin to write `tokenOwner` for arbitrary addresses that are not real launchpad tokens, creating "weird state" and making future reasoning/integration harder.

Impact

- ETH treasury fees can be irreversibly lost by sending them to `address(0)`.
- Uniswap V4 graduation can be bricked (revert) or result in a second position owned by `address(0)`, risking permanent loss of ownership/control over that liquidity position.
- A token can become effectively "non-existent" to the launchpad (`exists()` becomes false), potentially causing denial-of-service and/or blocking graduation.
- Reserved owner supply at graduation can be burned or cause graduation to revert.

Tool Used

Manual Review

Recommendation

- Add a strict non-zero check in `setTreasuryAddress`: `require(recipient != address(0), ...)`.
- Add a strict non-zero check for token ownership updates (best centralized in `_transferTokenOwnership`): `require(newTokenOwner != address(0), ...)`.

Issue L-4: `TokenOwnerUpdated` event omits the token address, preventing reliable off-chain indexing [RESOLVED]

Source: https://github.com/sherlock-audit/2026-01-livo-launchpad-jan-23rd/issues/10

Summary

`LivoLaunchpad` can manage many tokens, but its `TokenOwnerUpdated` event only includes `newOwner` and does not include which `token` was updated. As a result, off-chain indexers cannot reliably attribute ownership changes to a specific token, leading to ambiguous/incorrect indexing and downstream UI/accounting issues.

Vulnerability Detail

The launchpad stores token ownership per token in `tokenConfigs[token].tokenOwner`, and updates it via `_transferTokenOwnership(address token, address newTokenOwner)`.

However, the emitted event does not include the `token` parameter:

- Event definition: `event TokenOwnerUpdated(address indexed newOwner);`
- Emit site: `emit TokenOwnerUpdated(newTokenOwner);`

Because the event does not carry the token address, an indexer observing `TokenOwnerUpdated` cannot determine which token's `tokenOwner` changed without relying on additional context that is not available in logs (e.g., tracing the whole call and decoding calldata), which breaks "log-only" indexing and can be fragile for integrators.

Impact

- Indexers cannot reliably map owner updates to a specific token when multiple launchpad tokens exist.
- UIs and analytics may display incorrect token ownership, misattribute privileged actions, or fail to reflect ownership transfers without expensive/fragile transaction tracing.

Tool Used

Manual Review

Recommendation

- Change the event signature to include the token address, and emit it from `_transferTokenOwnership`:
  - `event TokenOwnerUpdated(address indexed token, address indexed newOwner);`
  - `emit TokenOwnerUpdated(token, newTokenOwner);`

Issue L-5: Unused `treasuryEthFees` state variable in `LivoGraduatorUniswapV4` can mislead integrators and wastes gas [RESOLVED]

Source: https://github.com/sherlock-audit/2026-01-livo-launchpad-jan-23rd/issues/12

Summary

`LivoGraduatorUniswapV4` defines a public state variable `treasuryEthFees` ("Treasury eth fees collected"), but it is never read from or written to anywhere in the contract. This introduces a misleading public getter (always returning `0`) and unnecessarily consumes a storage slot and deployment gas.

Vulnerability Detail

In `LivoGraduatorUniswapV4.sol`, the contract declares:

- `uint256 public treasuryEthFees;`

There are no references to this variable besides its declaration, meaning it is dead state. The actual "treasury portion" of LP ETH fees is handled implicitly via `address(this).balance` and can be forwarded via `sweep()`, but no accounting variable is updated.

Because the variable is `public`, off-chain users/indexers/integrators may incorrectly assume it represents cumulative treasury fees and use it for dashboards or accounting, while it will remain at its default value (`0`) forever.

Impact

- Off-chain accounting/UI can be incorrect if it relies on `treasuryEthFees()` (always `0`).
- Unnecessary storage slot increases deployment cost (and slightly increases state size / maintenance surface).

Tool Used

Manual Review

Recommendation

If fee accounting is not needed: remove `treasuryEthFees` to reduce state and avoid misleading integrators.

Issue L-6: Unnecessary logic for computing tax amount. [RESOLVED]

Source: https://github.com/sherlock-audit/2026-01-livo-launchpad-jan-23rd/issues/13

Summary

Unnecessary logic for computing tax amount.

Vulnerability Detail

Tax is computed as follows:

```
uint256 absTokenAmount = uint256(uint128(tokenDelta > 0 ? tokenDelta :
 ↪  -tokenDelta));
uint256 taxAmount = (absTokenAmount * taxBps) / BASIS_POINTS;
```

The `tokenDelta` is the output amount (amount of token we are receiving for the swap). Since that is always the case, then we know the `tokenDelta` will always be positive as the delta for the token being received will always be positive for a swap. Thus, the ternary logic is unnecessary.

Impact

Unnecessary logic, causing increased gas costs.

Tool Used

Manual Review

Recommendation

```
+ uint256 absTokenAmount = uint256(uint128(tokenDelta));
- uint256 absTokenAmount = uint256(uint128(tokenDelta > 0 ? tokenDelta :
 ↪  -tokenDelta));
```

Disclaimers