Wild Protocol - Security Review

06.29.2025

Conducted by:

Kann, Lead Security Researcher

Table of Contents

1 Disclaimer

Audits are a time, resource, and expertise bound effort where trained experts evaluate smart contracts using a combination of automated and manual techniques to identify as many vulnerabilities as possible. Audits can show the presence of vulnerabilities **but not their absence**.

2 Risk classification

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

2.1 Impact

- **High** - leads to a significant loss of assets in the protocol or significantly harms a group of users.
- **Medium** - only a small amount of funds can be lost or a functionality of the protocol is affected.
- **Low** - any kind of unexpected behaviour that's not so critical.

2.2 Likelihood

- **High** - direct attack vector; the cost is relatively low to the amount of funds that can be lost.
- **Medium** - only conditionally incentivized attack vector, but still relatively likely.
- **Low** - too many or too unlikely assumptions; provides little or no incentive.

3 Executive summary

Overview

Project Name	Wild Protocol
Repository	https://github.com/wildprotocol/elpee/tree/wip
Commit hash	9e66614
Resolution	Fixed
Documentation	
Methods	Manual review

Scope

/src/Deployer.sol
/src/LpLocker.sol
/src/StateManager.sol
/src/libraries/CurrencySend.sol
/src/libraries/CurveMath.sol

Issues Found

Critical risk	0
High risk	1
Medium risk	4
Low risk	1
Informational	1

4 Findings

4.1 High Risk

4.1.1 LP Fees Unclaimable via launchV4Pool()

Severity: *High risk*

Description: The function launchV4Pool() allows custom pool creation and token graduation (with manually set ticks and spacing). However, it does not call lpLocker.setTokenParams(), which is required to configure the LP locker for fee claiming.

Without calling setTokenParams(), fees earned from the pool become unclaimable, rendering the fee distribution logic non-functional for tokens launched via launchV4Pool().

In contrast, the graduateToken() function does invoke lpLocker.setTokenParams(), meaning fee claiming only works when using that path. This makes the launchV4Pool() function incomplete and effectively useless for real-world deployments where fees matter.

Resolution: Fixed

4.2 Medium Risk

4.2.1 Bonding Curve Check stepsize * numSteps May Not Match curveSupply

Severity: *Medium risk*

Description: When launching a token with bonding curve parameters, there is no validation ensuring that stepsize * numSteps == curveSupply. If this is not aligned:

If stepsize * numSteps > curveSupply, the bonding curve will appear to offer more tokens than were actually allocated, leading to potential inconsistencies (e.g., using LP pool tokens to fulfill purchases).

If stepsize * numSteps < curveSupply, part of the curve supply will become unreachable via the bonding curve, and remain unused.

While this behavior may be intentional to allow flexible bonding curve shapes, without an explicit check or warning, it can lead to unintended launch behavior due to misconfiguration.

Resolution: Fixed

4.2.2 Incorrect ETH Transfer to State Manager — Full msg.value Sent Instead of Used Amount

Severity: *Medium risk*

Description: When a user purchases a launch token using ETH (base token), the acceptAmount() function correctly calculates the amount used (amountInUsed) and refunds the excess ETH back to the user. However, despite the refund, the entire msg.value is still forwarded to the StateManager.buyToken() call, rather than just the used portion.

This results in the StateManager contract receiving more ETH than it should, creating incorrect accounting and potential fund mismanagement.

Resolution: Fixed

4.2.3 Excess Base Tokens Not Refunded for ERC20 tokens

Severity: *Medium risk*

Description: When a user purchases tokens using an ERC20 base token, the function _getBuyQuoteAndFees() correctly calculates the actual amount used (amountInUsed) and the associated fees. However, if the user sends more than amountInUsed, the excess tokens are not refunded.

In contrast, when using ETH as the base token, any overpayment is explicitly refunded. This inconsistent behavior creates a silent loss of funds for users interacting via ERC20 tokens.

Resolution: Fixed

4.2.4 ERC20 Transfer — Not All Tokens Return Boolean

Severity: *Medium risk*

Description: Uses ERC20(token).transfer(…) and checks whether it returns true to confirm success. However, some widely-used tokens like USDT do not return any value on transfer, which causes the boolean check to fail and revert even if the transfer was successful.

Resolution: Fixed

4.3 Low Risk

4.3.1 Forced Graduation Flag Ignored

Severity: *Low risk*

Description: Although the deployer can set allowForcedGraduation to false, graduateToken() and _launchV4Pool() do not validate this setting, allowing owner of deployer and creator to forcibly graduate a token even when it was explicitly disabled.

Resolution: Fixed

4.4 Informational

4.4.1 Missing Validation — numSteps and prices.length Mismatch

Severity: *Informational*

Description: In the PriceCurve bondingCurveParams, no check ensures that numSteps equals prices.length. A mismatch could result in unexpected pricing behavior or runtime errors.

Resolution: Fixed