

REVEAL³

LaunchPool

Security Audit

1. Disclaimer

Security audits cannot uncover all existing vulnerabilities; even an audit in which no vulnerabilities are found is not a guarantee of a secure system.

However, code assessments enable the discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it.

Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code audits aimed at determining all locations that need to be fixed. Within the customer-determined time frame, we performed an audit in order to discover as many vulnerabilities as possible.

The focus of our audit was limited to the code parts defined in the engagement letter. We assessed whether the project follows the provided specifications. These assessments are based on the provided threat model and trust assumptions. We draw attention to the fact that due to inherent limitations in any software development process and software product, an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third-party infrastructure which adds further inherent risks as we rely on the correct execution of the included third-party technology stack itself.

Report readers should also take into account that over the life cycle of any software, changes to the product itself or to the environment in which it is operated can have an impact leading to operational behaviors other than those initially determined in the business specification.

2. Terminology & Risk Classification

For the purpose of this audit, we adopt the following terminology: To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

Likelihood: The likelihood of a finding to be triggered or exploited in practice.

Impact: The technical and business-related consequences of a finding.

Severity: An additive consideration based on the Likelihood and the Impact (as referenced below).

We use the table below to determine the severity of any and all findings. Please note that none of these risk classifications constitute endorsement or opposition to a particular project or contract.

		<u>Impact</u>	
<u>Likelihood</u>	High	Medium	Low
High	Severity: Critical	Severity: High	Severity: Medium
Medium	Severity: High	Severity: Medium	Severity: Low
Low	Severity: Medium	Severity: Low	Severity: Low

3. Summary

Contracts In-scope:

- LaunchPoolToken.sol

Repo:

- <https://github.com/Launchpool/LPOOL-contracts-new>

Branch:

- dev

Commit hash:

- 594d6e87cf94223d7594c8e20dfe9853e4b7fafc

In the period 19 Dec. 2023 - 22 Dec. 2023 we audited LaunchPool's smart contract.

In this period of time a total of 5 of issues were found.

Severity	Issues
Critical Risk	0
High Risk	0
Medium Risk	0
Low Risk	3
Gas Optimization	-
Informational	2

4. Issues

[Informational - 001]: Missing Initialization of Inherited ERC20 Contracts

Description: The smart contract inherits ERC20BurnableUpgradeable and ERC20VotesUpgradeable from OpenZeppelin's upgradeable contracts collection. However, these inherited contracts are not properly initialized in the constructor.

Affected contracts: LaunchPoolToken.sol

Status: Solved by protocol (Commit: 97e5778fc6f338431bb9dbe8ee1a417bbe196010).

[Informational - 002]: Indirect Inheritance of ERC20 Upgradable

Description: The smart contract does not inherit ERC20 Upgradable directly. This indirect inheritance can lead to complications or misunderstandings about the contract's structure and capabilities. Direct inheritance of foundational contracts like ERC20 Upgradable is crucial for clarity, ensuring that all essential functionalities and properties are explicitly integrated and maintained.

Affected contracts: LaunchPoolToken.sol

Status: Solved by protocol (Commit: a6ab0f43279e8cd736a20025d472f6a7562a6379).

[Low - 001]: Lack of Address Validation in setTrustedForwarder Function

Description: The setTrustedForwarder function in the contract, which calls _setTrustedForwarder from ERC2771Recipient.sol, lacks validation checks for the _forwarder address. Since ERC2771Recipient.sol does not perform checks on the _forwarder address, it's crucial to implement these checks within the setTrustedForwarder function to ensure that the address is neither the zero address (0x0) nor the dead address (0xdead).



```
1  /**
2   * @notice Set new trusted forwarder from `ERC2771Recipient`
3   * @dev Sender must be the owner
4   * @param _forwarder New trusted forwarder
5   */
6  function setTrustedForwarder(address _forwarder) external onlyOwner {
7      _setTrustedForwarder(_forwarder);
8  }
```

Affected contracts: LaunchPoolToken.sol

Status: Solved by protocol (Commit: cec901bfcaf6f37684a46a7f69402ffcb3b5f47b).

[Low - 002]: Missing Validation for owner_ and treasury_ Addresses in Initialize Function

Description: In the initialize function of the smart contract, there are no checks to verify if the owner_ or treasury_ addresses are either the zero address (0x0) or the dead address (0xdead). Without these validations, there's a risk that the initial supply could inadvertently be sent to these invalid addresses if treasury_ is set as either the zero or dead address. This could lead to the permanent loss of the initial supply.

This issue has been marked as low severity because the contracts can be redeployed. Therefore, no funds are at risk even if the initial deployment assigns invalid addresses.

```
1  /**
2  * @notice The initialization function which replaces constructor (See upgradeable [docs](https://docs.openzeppelin.com/upgrades-plugins/1.x/))
3  * @param name_ ERC20 token name
4  * @param symbol_ ERC20 token symbol
5  * @param initialSupply_ Initial supply of tokens minted to treasury
6  * @param treasury_ Address which receives initial supply of tokens
7  * @param owner_ Address which can upgrade smart-contract and call pause(), unpause() functions
8  */
9  function initialize(
10     string memory name_,
11     string memory symbol_,
12     uint initialSupply_,
13     address treasury_,
14     address owner_
15 ) public initializer {
16     // Initialize inherited plugins
17     __ERC20_init(name_, symbol_);
18     __ERC20Permit_init(name_);
19     __ERC20Pausable_init();
20     __Ownable_init(owner_);
21
22     // Mint initial supply
23     _mint(treasury_, initialSupply_);
24 }
```

Affected contracts: LaunchPoolToken.sol

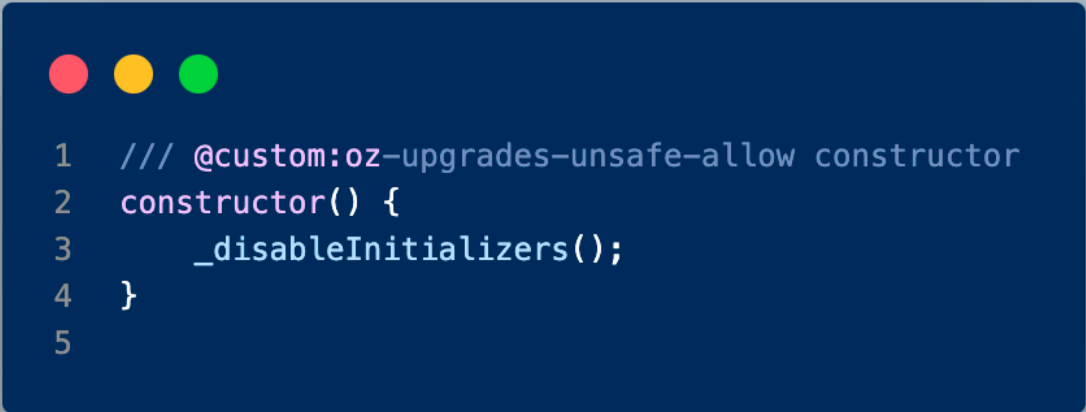
Status: Solved by protocol (Commit: e3225d9fc992048ecf4f38bba6ef8d85ae619397).

[Low - 003]: Lack of Constructor Disablement in Contract Using Initializable.sol

Description: The smart contract employs Initializable.sol from OpenZeppelin but does not explicitly disable the constructor as recommended in OpenZeppelin's documentation. Without disabling the constructor using `_disableInitializers()`, there is a risk that an uninitialized contract could be susceptible to takeover by an attacker. This vulnerability is particularly concerning because an attacker could potentially snipe the deployment transaction and front-run the initialize transaction, thereby gaining ownership or control over the contract.

Remediation: Modify the contract to include the `_disableInitializers()` function within the constructor. This ensures that the contract is automatically locked when deployed, preventing its use before initialization and safeguarding against unauthorized takeovers.

This issue has been marked as low severity because the contracts can be redeployed.



```
1  /// @custom:oz-upgrades-unsafe-allow constructor
2  constructor() {
3      _disableInitializers();
4  }
5
```

Affected contracts: LaunchPoolToken.sol

Status: Solved by protocol (Commit: [ac26e18cea97ac32c9f2e2b7d116a709349d0da9](#)).

5. Recommendations

- **Caution with block.number on Arbitrum:** In future versions, avoid using block.number on the Arbitrum network.

block.number on Arbitrum returns the most recently synced Ethereum mainnet (L1) block number, not the Arbitrum (L2) block number.

The Sequencer updates the block number to match the L1 block number approximately once per minute.

Relying on block.number as a timing mechanism can lead to inaccuracies due to this discrepancy.

Instead, consider using alternative methods for time-dependent logic.

- **Compatibility Issues with Upgradable Contracts:** If updating another smart contract that was deployed using OpenZeppelin's upgradable contracts (versions prior to 5), be aware of potential compatibility issues.

Versions 4 and 5 of OpenZeppelin's upgradable contracts use different approaches for handling storage layout:

Version 4: Storage slot collisions are managed using a 'gap' strategy. This involves leaving space between storage variables to allow for future variable insertion without disrupting existing storage layout.

Version 5: Storage slot collisions are handled using ERC7201, which offers a more structured/secure approach to storage layout management.