[Audited]

<https://etherscan.io/address/0x0000000000095413afC295d19EDeb1Ad7B71c952> (Contract)

[Summary]

We found 1 medium and 6 low-level issues:

[Issues & POCs]

**Issue 1**: Medium (Unchecked transfer in ‘emergencyWithdraw’ function) <https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer>

Issue Description: The ‘emergencyWithdraw’ function in Lon.sol uses the transfer method to send tokens to the ‘emergencyRecipient’ without checking the return value. This can lead to potential loss of funds if the transfer fails. Attackers could exploit this vulnerability to drain funds by preventing the transfer from succeeding.

Impact: The impact of this issue is considered medium because it allows attackers to potentially drain funds from the contract.

Proof of Concept:

function emergencyWithdraw(IERC20 token) external {

// Retrieve token balance of the contract

uint256 amount = token.balanceOf(address(this));

// Transfer tokens to emergency recipient without checking return value

emergencyRecipient.transfer(amount);

}

Recommended Mitigation Steps: To address this vulnerability, ensure that all token transfers are checked for success before proceeding. Use transfer in conjunction with a require statement to revert the transaction if the transfer fails.

**Issue 2**: Low (Variable shadowing in ‘permit’ function) <https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing>

Issue Description: The ‘permit’ function in Lon.sol shadows the owner state variable inherited from the Ownable contract. This can lead to confusion and unintended behavior as the function's owner parameter overrides the owner state variable.

Recommendation: Rename the owner parameter in the permit function to avoid shadowing and potential confusion.

**Issue 3**: Low (Lack of zero-address validation in constructor and ‘setMinter’ function) <https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation>

Issue Description: The constructor and ‘setMinter’ function in Lon.sol lack zero-address validation when assigning values to the minter and ‘emergencyRecipient’ variables. Failure to perform this validation can result in unexpected behavior if zero addresses are supplied as arguments.

Recommendation: Add require statements to validate that non-zero addresses are provided as arguments to the constructor and ‘setMinter’ function.

**Issue 4**: Low (Use of ‘block.timestamp’ for comparisons) <https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp>

Issue Description: The permit function in Lon.sol uses ‘block.timestamp’ for comparisons, which can be manipulated by miners to execute transactions at specific times. This can lead to front-running attacks or inaccurate expiration checks.

Recommendation: Avoid using ‘block.timestamp’ for sensitive operations such as time-based validations. Consider using ‘block.number’ or an external time oracle for more reliable timestamp comparisons.

**Issue 5**: Low (Different versions of Solidity used) <https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used>

Issue Description: Lon.sol and its imported interfaces use different versions of Solidity, which can introduce compatibility issues or unexpected behavior due to differences in language features and optimizations.

Recommendation: Standardize the Solidity version across all contracts and interfaces to ensure consistency and reduce the risk of compatibility issues. Update the pragma directives to use a single version of Solidity that is compatible with all contracts and interfaces.

**Issue 6**: Low (Unused functions and variables) Issue Description: Several functions and variables in the Lon.sol contract are unused, leading to redundant code and potential confusion for developers reviewing the contract.

Recommendation: Remove the unused functions and variables to improve code clarity and reduce unnecessary gas consumption during deployment and execution.

**Issue 7**: Low (Gas optimization opportunities) Issue Description: The code contains opportunities for gas optimization, such as using more efficient data types and inline assembly, to reduce gas costs and improve contract efficiency.

Recommendation: Review the code for areas where gas optimization can be applied, such as replacing bool variables with uint256 or other full-word types, and consider using inline assembly for performance-critical operations.