Stake.Link Audit Report

Prepared by Cyfrin

Version 2.0

Lead Auditors

Immeas

February 28, 2025

Contents

1 About Cyfrin

Cyfrin is a Web3 security company dedicated to bringing industry-leading protection and education to our partners and their projects. Our goal is to create a safe, reliable, and transparent environment for everyone in Web3 and DeFi. Learn more about us at cyfrin.io.

2 Disclaimer

The Cyfrin team makes every effort to find as many vulnerabilities in the code as possible in the given time but holds no responsibility for the findings in this document. A security audit by the team does not endorse the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the solidity implementation of the contracts.

3 Risk Classification

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

4 Protocol Summary

The Stake.Link pull request audit focused on two feature additions:

1. **Reward Token Withdrawal** – The update enables the retrieval of arbitrary reward tokens from the Operator and Community vaults, with the only restriction being that LINK tokens cannot be withdrawn, as they are the primary staking assets in the vaults. While anyone can initiate the withdrawal, all reward tokens are directed to a protocol-controlled wallet for future distribution.
2. **Integration with delegate.xyz** – The update introduces integration with delegate.xyz for both the Community and Operator vaults. This allows the contract owner to use the `FundFlowController` to delegate rights within the delegate.xyz ecosystem.

5 Audit Scope

PR#140 which includes changes in files:

- `contracts/linkStaking/CommunityVCS.sol`
- `contracts/linkStaking/CommunityVault.sol`
- `contracts/linkStaking/FundFlowController.sol`
- `contracts/linkStaking/OperatorVCS.sol`
- `contracts/linkStaking/OperatorVault.sol`
- `contracts/linkStaking/base/Vault.sol`
- `contracts/linkStaking/base/VaultControllerStrategy.sol`
- `contracts/linkStaking/interfaces/IDelegateRegistry.sol`
- `contracts/linkStaking/interfaces/IVault.sol`

- contracts/linkStaking/interfaces/IVaultControllerStrategy.sol

6 Executive Summary

Over the course of 4 days, the Cyfrin team conducted an audit on the Stake.Link smart contracts provided by Stake.Link. In this period, a total of 2 issues were found.

The audit identified one informational finding related to event emissions and one gas optimization finding concerning the token flow when withdrawing reward tokens.

The Stake.Link team had implemented tests for the new functionality, ensuring good coverage and verification of the added features.

Summary

Project Name	Stake.Link
Repository	contracts
Commit	046c65a9c771...
Audit Timeline	Feb 25th - Feb 28th
Methods	Manual Review

Issues Found

Critical Risk	0
High Risk	0
Medium Risk	0
Low Risk	0
Informational	1
Gas Optimizations	1
Total Issues	2

Summary of Findings

[I-1] Lack of events emitted on state changes	Acknowledged
[G-1] Unnecessary token transfer when withdrawing reward tokens	Acknowledged

7 Findings

7.1 Informational

7.1.1 Lack of events emitted on state changes

Description: The following functions should ideally emit an event to enhance transparency and traceability:

`Vault::setDelegateRegistry` and `VaultControllerStrategy::setDelegateRegistry`:

```
  function setDelegateRegistry(address _delegateRegistry) external onlyOwner {
      delegateRegistry = _delegateRegistry;
+     emit SetDelegateRegistry(_delegateRegistry);
  }
```

`FundFlowController::setNonLINKRewardReceiver`:

```
  function setNonLINKRewardReceiver(address _nonLINKRewardReceiver) external onlyOwner {
      nonLINKRewardReceiver = _nonLINKRewardReceiver;
+     emit SetNonLINKRewardReceiver(_nonLINKRewardReceiver);
  }
```

Additionally, an event could be emitted when rewards are withdrawn in `FundFlowController::withdrawTokenRewards`:

```
  function withdrawTokenRewards(address[] calldata _vaults, address[] calldata _tokens) external {
      // ...
+     emit WithdrawTokenRewards(msg.sender, _vaults, _tokens);
  }
```

Consider adding events to these functions to provide a clear on-chain record of when and by whom these actions were executed. This improves transparency and makes it easier to track changes.

Stake.Link: Acknowledged.

Cyfrin: Acknowledged.

7.2 Gas Optimization

7.2.1 Unnecessary token transfer when withdrawing reward tokens

Description: When claiming non-LINK reward tokens, the tokens are transferred `Vault -> FundFlowController -> nonLINKRewardReceiver`:

`Vault::withdrawTokenRewards` transfers to `msg.sender` (`FundFlowController`):

```
function withdrawTokenRewards(address[] calldata _tokens) external onlyFundFlowController {
    for (uint256 i = 0; i < _tokens.length; ++i) {
        IERC20Upgradeable rewardToken = IERC20Upgradeable(_tokens[i]);
        uint256 balance = rewardToken.balanceOf(address(this));
        if (balance != 0) rewardToken.safeTransfer(msg.sender, balance);
    }
}
```

and `FundFlowController::withdrawTokenRewards` transfers to the protocol wallet `nonLINKRewardReceiver`:

```
function withdrawTokenRewards(address[] calldata _vaults, address[] calldata _tokens) external {
    for (uint256 i = 0; i < _vaults.length; ++i) {
        IVault(_vaults[i]).withdrawTokenRewards(_tokens);
    }

    for (uint256 i = 0; i < _tokens.length; ++i) {
        IERC20Upgradeable rewardToken = IERC20Upgradeable(_tokens[i]);
        if (address(rewardToken) == linkToken) revert InvalidToken();
        uint256 balance = rewardToken.balanceOf(address(this));
        if (balance != 0) rewardToken.safeTransfer(nonLINKRewardReceiver, balance);
    }
}
```

This could be optimized by letting the vault transfer to `nonLINKRewardReceiver` directly, thus removing one token transfer from the flow:

```
function withdrawTokenRewards(address[] calldata _vaults, address[] calldata _tokens) external {
    // cache linkToken
    address _linkToken = linkToken;

    // check for LINK token
    for (uint256 i = 0; i < _tokens.length; ) {
        if (_tokens[i] == _linkToken) revert InvalidToken();
        unchecked { ++i; }
    }

    for (uint256 i = 0; i < _vaults.length; ++i) {
        // add `nonLINKRewardReceiver` in the call to vault.withdrawTokenRewards
        IVault(_vaults[i]).withdrawTokenRewards(_tokens, nonLINKRewardReceiver);
    }
}
```

```
- function withdrawTokenRewards(address[] calldata _tokens) external onlyFundFlowController {
+ function withdrawTokenRewards(address[] calldata _tokens, address _receiver) external
↪   onlyFundFlowController {
      for (uint256 i = 0; i < _tokens.length; ++i) {
          IERC20Upgradeable rewardToken = IERC20Upgradeable(_tokens[i]);
          uint256 balance = rewardToken.balanceOf(address(this));
-         if (balance != 0) rewardToken.safeTransfer(msg.sender, balance);
+         if (balance != 0) rewardToken.safeTransfer(_receiver, balance);
      }
  }
```

As `Vault::withdrawTokenRewards` is already protected by `onlyFundFlowController` this poses no extra risk.

Stake.Link: Acknowledged.

Cyfrin: Acknowledged.