Spirit Protocol Audit Report

Prepared by 0xSimao

Version 1.0

December 9, 2025

Contents

1 About 0xSimao **2**

2 Disclaimer **2**

3 Risk Classification **2**

4 Protocol Summary **2**
4.1 Core Contracts . . . 2
4.1.1 SpiritToken / ChildSuperToken . . . 2
4.1.2 SpiritFactory . . . 2
4.1.3 StakingPool . . . 3
4.1.4 RewardController . . . 3
4.1.5 SpiritVestingFactory / SpiritVesting . . . 3
4.2 Token Distribution . . . 3
4.2.1 CHILD Token Distribution (1B per token) . . . 3
4.2.2 Staking Multipliers . . . 3
4.3 Key Parameters . . . 4
4.4 External Dependencies . . . 4

5 Audit Scope **4**

6 Executive Summary **4**

7 Findings **6**
7.1 Critical Risk . . . 6
7.1.1 Wrong rounding direction in StakingPool::unstake() can be abused . . . 6
7.2 High Risk . . . 7
7.2.1 SuperTokens are vulnerable to attacker frontrunning and minting free tokens . . . 7
7.2.2 Uniswap v4 pool initialization can be frontrunned, setting an arbitrary price, and stealing tokens 7
7.3 Medium Risk . . . 9
7.3.1 SpiritFactory won't be able to stop the Airstream contract . . . 9
7.4 Low Risk . . . 10
7.4.1 StakingPool is missing rewards distribution end logic . . . 10
7.5 Informational . . . 11
7.5.1 SpiritVestingFactory::createSpiritVestingContract overwrites the spiritVestings mapping for the same recipient . . . 11
7.5.2 SpiritVestingFactory::setTreasury() could be 2 factor, as well as the AccessControl OZ contract used in the codebase . . . 12

1 About 0xSimao

0xSimao is an indepedent security researcher, #2 on Sherlock, top-ranked on Cantina and Code4rena, and a member of Blackthorn, a leading auditing firm. Previously, he served as Head of Security at Three Sigma.

He has placed in the Top 3 in 28 public audits and has led over 60 private engagements. For private audits or collaboration opportunities, feel free to reach out to him on X (@0xSimao), Telegram (@OxSimao), or Discord (@0xSimao).

2 Disclaimer

0xSimao 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

Spirit Protocol is a **decentralized token distribution and staking system** built on Ethereum using Superfluid streaming technology and Uniswap V4 liquidity pools. The protocol enables the creation of "child" tokens associated with artists/agents, where users can stake these child tokens to earn continuous SPIRIT token rewards via real-time streaming.

4.1 Core Contracts

4.1.1 SpiritToken / ChildSuperToken

- **Type:** Superfluid SuperToken.
- **Supply:** 1 billion tokens each.
- **Features:** Native streaming capabilities via Superfluid protocol.

4.1.2 SpiritFactory

- **Purpose:** Creates new child tokens with associated infrastructure.
- **Access:** Admin-only (`DEFAULT_ADMIN_ROLE`).
- **Creates per child token:**
  - Child SuperToken (1B supply).
  - StakingPool contract.
  - CHILD/SPIRIT Uniswap V4 liquidity pool.
  - Airstream distribution (merkle-based airdrop).

4.1.3 StakingPool

- **Purpose:** Users stake CHILD tokens to earn SPIRIT rewards.
- **Mechanism:** Superfluid GDA (General Distribution Agreement) pool.
- **Features:**
  - Time-locked staking (1 week to 3 years).
  - Multiplier bonuses based on lock duration (1x to 36x).
  - Continuous SPIRIT streaming to stakers.
  - 1-week reward smoothing period.

4.1.4 RewardController

- **Purpose:** Distributes SPIRIT rewards to staking pools.
- **Roles:**
  - `FACTORY_ROLE` - Can register new staking pools.
  - `DISTRIBUTOR_ROLE` - Can send rewards to pools.
  - `DEFAULT_ADMIN_ROLE` - Can upgrade contract.

4.1.5 SpiritVestingFactory / SpiritVesting

- **Purpose:** Team/operations token vesting.
- **Mechanism:** Superfluid VestingSchedulerV3.
- **Features:**
  - Cliff + streaming vesting schedules.
  - Treasury-controlled cancellation.
  - vestedSPIRIT representation.

4.2 Token Distribution

4.2.1 CHILD Token Distribution (1B per token)

Allocation	Amount	Percentage
Artist (locked 1 year)	250M	25%
Agent (locked 1 year)	250M	25%
Uniswap V4 Liquidity	250M	25%
Airstream (1-year airdrop)	250M	25%

4.2.2 Staking Multipliers

Lock Duration	Multiplier	Reward Share
1 week (min)	10,000	1x
52 weeks (1 year)	~125,000	~12.5x

Lock Duration	Multiplier	Reward Share
156 weeks (3 years, max)	360,000	36x

4.3 Key Parameters

Parameter	Value
Minimum Stake	1 CHILD token
Minimum Lock	1 week
Maximum Lock	156 weeks (3 years)
Reward Stream Duration	1 week
Stakeholder Lock Period	52 weeks
Airstream Duration	52 weeks

4.4 External Dependencies

- **Superfluid Protocol** - Real-time token streaming.
- **Uniswap V4** - Liquidity pools and position management.
- **OpenZeppelin** - Access control, proxies, safe math.
- **Permit2** - Token approvals for Uniswap.

5 Audit Scope

Contract	nSLOC
src/factory/SpiritFactory.sol	218
src/core/StakingPool.sol	139
src/vesting/SpiritVestingFactory.sol	58
src/vesting/SpiritVesting.sol	55
src/core/RewardController.sol	49
src/token/ChildSuperToken.sol	21
src/token/SpiritToken.sol	21
Total	**561**

6 Executive Summary

Over the course of 3 days, 0xSimao conducted an audit on the Spirit Protocol. In this period, a total of 7 issues were found.

Summary

Project Name	Spirit Protocol
Repository	spirit-contracts
Commit	f6644959e3bc...
Fix Commit	e62cd68e7b4f...
Audit Timeline	Nov 28th - Dec 2nd
Methods	Manual Review, Stateful Fuzzing

Issues Found

Critical Risk	1
High Risk	2
Medium Risk	1
Low Risk	1
Informational	2
Gas Optimizations	0
Total Issues	7

Summary of Findings

[C-1] Wrong rounding direction in StakingPool::unstake() can be abused	Resolved
[H-1] SuperTokens are vulnerable to attacker frontrunning and minting free tokens	Resolved
[H-2] Uniswap v4 pool initialization can be frontrunned, setting an arbitrary price, and stealing tokens	Resolved
[M-1] SpiritFactory won't be able to stop the Airstream contract	Resolved
[L-1] StakingPool is missing rewards distribution end logic	Resolved
[I-1] SpiritVestingFactory::createSpiritVestingContract overwrites the spiritVestings mapping for the same recipient	Resolved
[I-2] SpiritVestingFactory::setTreasury() could be 2 factor, as well as the AccessControl OZ contract used in the codebase	Acknowledged

7 Findings

7.1 Critical Risk

7.1.1 Wrong rounding direction in StakingPool::unstake() can be abused

Description: StakingPool::unstake() reduces the units of the user pro-rata to the unstaked amount:

```
function unstake(uint256 amount) external {
    ...

    // Get current units and calculate units to remove proportionally
    // This maintains the exact proportional relationship between units and staked amount
    uint128 currentUnits = distributionPool.getUnits(msg.sender);
    uint128 unitsToRemove = uint128((amount * currentUnits) / userStakingInfo.stakedAmount);

    // Update member units
    distributionPool.decreaseMemberUnits(msg.sender, unitsToRemove);

    // Update staking info
    _stakingInfo[msg.sender].stakedAmount -= amount;

    // If all tokens are unstaked, reset the staking info
    if (_stakingInfo[msg.sender].stakedAmount == 0) {
        delete _stakingInfo[msg.sender];
    }

    // Transfer staked tokens back to user
    child.transfer(msg.sender, amount);

    emit Unstaked(msg.sender, amount);
}
```

The issue is that, as it rounds the units down, a staker is able to abuse it by unstaking many times, each time rounding it down, withdrawing all the stake, but keeping many units. Since 1000e18 stake is 1000 units, if they withdraw 1.9999e18, they remove only 1 unit, since `1.9999e18*1000/1000e18 = 1`. By repeating this, they can unstake the full stake, and keep 50% staked.

Impact: Stolen rewards.

Recommended Mitigation: Round up. Also needs to make sure it doesn't underflow when reducing units.

0xSimao:

Fixed in PR #5.

7.2 High Risk

7.2.1 SuperTokens are vulnerable to attacker frontrunning and minting free tokens

Description: SpiritToken::initialize() is vulnerable to an attacker frontrunning the initialization with a malicious implementation, minting tokens for themselves, and then setting the implementation to `address(0)` again.

The inherited `UUPSProxy` allows this because it has no access control, and although it checks for `initialAddress != address(0)`, this check won't exist on the new attacker implementation, so they can just reset it to 0 after minting free tokens.

```
function initializeProxy(address initialAddress) external {
    require(initialAddress != address(0), "UUPSProxy: zero address");
    require(UUPSUtils.implementation() == address(0), "UUPSProxy: already initialized");
    UUPSUtils.setImplementation(initialAddress);
}
```

Impact: Spirit and Child tokens can be stolen.

Recommended Mitigation: Add a deployer contract or make it access controlled by a trusted address in the constructor.

0xSimao:

Fixed in PR #6.

7.2.2 Uniswap v4 pool initialization can be frontrunned, setting an arbitrary price, and stealing tokens

Description: SpiritFactory::_setupUniswapPool() mints a single sided position, from the current tick to the max tick. Thus, users can buy Child tokens for at least a price of the tick, which goes up as liquidity decreases.

However, an attacker can predict the address of the Child token (deterministic), or frontrun the transaction, and initialize the pool to the lowest price possible. This will make it add liquidity in the full range of the Uniswap pool, and an attacker can get all Child very cheaply.

```
function _setupUniswapPool(address childToken, uint256 childTokenAmount, uint160 initialSqrtPriceX96)
    internal
    returns (uint256 tokenId)
{
    // Ensure tokens are in the correct order (lower address first)
    Currency currency0 =
        childToken < address(SPIRIT) ? Currency.wrap(address(childToken)) :
        ↪  Currency.wrap(address(SPIRIT));
    Currency currency1 =
        childToken > address(SPIRIT) ? Currency.wrap(address(childToken)) :
        ↪  Currency.wrap(address(SPIRIT));

    // Create the pool key
    PoolKey memory poolKey = PoolKey({
        currency0: currency0,
        currency1: currency1,
        fee: DEFAULT_POOL_FEE,
        tickSpacing: DEFAULT_TICK_SPACING,
        hooks: IHooks(address(0))
    });

    // Initialize the pool
    POSITION_MANAGER.initializePool(poolKey, initialSqrtPriceX96); //@audit anyone can frontrun this

    tokenId = _mintSingleSidedLiquidityPosition(childToken, childTokenAmount, poolKey);
}
```

The key thing is that `POSITION_MANAGER.initializePool()` doesn't revert, it fails silently, so there is nothing the admin can do:

```
abstract contract PoolInitializer_v4 is ImmutableState, IPoolInitializer_v4 {
    /// @inheritdoc IPoolInitializer_v4
    function initializePool(PoolKey calldata key, uint160 sqrtPriceX96) external payable returns
    ↪  (int24) {
        try poolManager.initialize(key, sqrtPriceX96) returns (int24 tick) {
            return tick;
        } catch {
            return type(int24).max;
        }
    }
}
```

Impact: Stolen Child

Recommended Mitigation: Check the return value of the pool initialization, it must not be the maximum tick, revert if it is. Additionally, add a salt to the child token address generation, so any name/symbol token can be created. This may cause conflicts, so make sure to track the deployed name/symbol pair (can just add a mapping in the factory).

0xSimao:

Fixed in PR #8.

7.3 Medium Risk

7.3.1 SpiritFactory won't be able to stop the Airstream contract

Description: The Airstream controller is owned by the `SpiritFactory`, but the latter has no function to call the AirstreamController. Since the `SpiritFactory` is upgradeable, this can be mitigated.

The AirstreamController has 3 important functions that may be called, `resumeAirstream()`, `pauseAirstream()` and `withdraw()` (redirectRewards()‘ is optional).

The most important one is `pauseAirstream()`, since it stops the flow, preventing the controller from being liquidated and losing some of the funds.

Impact: Airstream partially liquidated.

Recommended Mitigation: Add the necessary functions to the `SpiritFactory`, `pauseAirstream()` being the most important.

0xSimao:

Fixed in PR #7.

7.4 Low Risk

7.4.1 StakingPool is missing rewards distribution end logic

Description: The StakingPool has 1 unit so no flow is lost, but there is no way for the owner to get this Spirit back. It will flow in the next distribution, but this issue refers specifically to when there is no next distribution. Additionally, there is no way to stop the flow, so it will be liquidated and some Spirit will be lost.

Impact: Lost Spirit.

Recommended Mitigation: Add a way to stop the flow and transfer the remaining Spirit.

0xSimao:

Fixed in PR #4.

7.5 Informational

7.5.1 SpiritVestingFactory::createSpiritVestingContract overwrites the spiritVestings mapping for the same recipient

Description: SpiritVestingFactory::createSpiritVestingContract() is as follows:

```
function createSpiritVestingContract(
    address recipient,
    uint256 amount,
    uint256 cliffAmount,
    uint32 cliffDate,
    uint32 endDate
) external onlyTreasury returns (address newSpiritVestingContract) {
    if (!(cliffAmount < amount)) revert FORBIDDEN();

    uint256 vestingDuration = endDate - cliffDate;

    uint256 vestingAmount = amount - cliffAmount;
    int96 flowRate = int256(vestingAmount / vestingDuration).toInt96();

    // Add the remainder to the cliff amount
    cliffAmount += vestingAmount - (uint96(flowRate) * vestingDuration);

    // Deploy the new SPIRIT Token Vesting contract
    newSpiritVestingContract =
        address(new SpiritVesting(VESTING_SCHEDULER, SPIRIT, recipient, cliffDate, flowRate,
        ↪  cliffAmount, endDate));

    // Maps the recipient address to the new SPIRIT Token Vesting contract
    spiritVestings[recipient] = newSpiritVestingContract; //@audit overwrites previous vesting contract
    ↪  if exists

    // Transfer the tokens from the treasury to the new vesting contract
    SPIRIT.transferFrom(msg.sender, newSpiritVestingContract, amount);

    // Emit the events
    emit Transfer(address(0), recipient, amount);
    emit SpiritVestingCreated(recipient, newSpiritVestingContract);
}
```

Note that creating more than 1 vesting contract for the same recipient overwrites the old now. The only impact is that the `SpiritVestingFactory::balanceOf()` function will point to the new vesting contract only.

Impact: Informational, inconsistent behaviour.

Recommended Mitigation: Consider replacing the mapping with an array.

0xSimao:

Fixed in PR #2.

7.5.2 SpiritVestingFactory::setTreasury() could be 2 factor, as well as the AccessControl OZ contract used in the codebase

Description: SpiritVestingFactory::setTreasury() is single factor, so a single mistake can lose treasury access forever, which is responsible for cancelling the treasury in the `SpiritVesting` contract.

```
function setTreasury(address newTreasury) external onlyTreasury {
    // Ensure the new treasury address is not the zero address
    if (newTreasury == address(0)) revert FORBIDDEN();
    treasury = newTreasury;
}
```

Additionally, throughout the codebase, `AccessControl` is used, which also doesn't have 2 factor. The `SpiritFactory` is mostly stateless, besides the fact that it is a role in the `RewardsController`, so the ability to set staking pools could be compromised. The `RewardController` relies on roles for upgrades, rewards distributions, so it is also a pain point.

Impact: Lost roles.

Recommended Mitigation: Use the AccessControlDefaultAdminRules for better role management.