PRELIMINARY REPORT

SHERLOCK

Security Review For Superfluid

Public Best Efforts contest prepared for:	**Superfluid**
Lead Security Expert:	**0x73696d616f**
Date Audited:	**November 20 - November 24, 2024**

Introduction

Superfluid is the money streaming protocol, powering a real-time onchain economy. Start earning every second with streaming airdrops, rewards and yield.

Scope

Repository: superfluid-finance/fluid

Branch: dev

Audited Commit: 8aa3402429c5e994abfde319d95c8115dc00c1ed

Final Commit: aa6163a422d6e5af6e8628e73c3d1f6688508dda

Repository: superfluid-finance/protocol-monorepo

Branch: dev

Audited Commit: c8795f8db446761279fa4a8aee0a48f2eb374d52

Final Commit: c8795f8db446761279fa4a8aee0a48f2eb374d52

For the detailed scope, see the contest details.

Findings

Each issue has an assigned severity:

- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- High issues are directly exploitable security vulnerabilities that need to be fixed.

Issues found

High	Medium
3	3

Security experts who found valid issues

0x73696d616f
Drynooo
zxriptor
merlinboii
newspacexyz
redbeans
ke1caM
Z3R0
056Security
IvanFitro
zhenyazhd
Kyosi

Issue H-1: `FluidLocker::_getUnlockingPercentage()` incorrectly divides one of the components of the formula by `S`, leading to always having 80% penalty

Source: https://github.com/sherlock-audit/2024-11-superfluid-locking-contract-judging/issues/20

Found by

056Security, 0x73696d616f, Drynooo, IvanFitro, Kyosi, Z3R0, ke1caM, newspacexyz, redbeans, zhenyazhd

Summary

FluidLocker::_getUnlockingPercentage() calculates the percentage to unlock, which is the amount given to the user, while the remaining goes to other stakers. The formula incorrectly divides `Math.sqrt(unlockPeriod*_SCALER)` by `SCALER`:

```
function _getUnlockingPercentage(uint128 unlockPeriod) internal pure returns
↪   (uint256 unlockingPercentageBP) {
    unlockingPercentageBP = (
        _PERCENT_TO_BP
            * (
                ((80 * _SCALER) / Math.sqrt(540 * _SCALER)) *
↪   (Math.sqrt(unlockPeriod * _SCALER) / _SCALER //@audit this _SCALER should not
↪   be here)
                    + 20 * _SCALER
            )
    ) / _SCALER;
}
```

Root Cause

In `FluidLocker:388`, it incorrectly divides the term `(Math.sqrt(unlockPeriod*_SCALER)` by `_SCALER`.

Internal pre-conditions

None.

External pre-conditions

None.

Attack Path

1. User unlocks their Fluid from the locker with a duration bigger than 0, unvesting it through the fountain.

Impact

User suffers a big loss, even if they unlock with the maximum period, they will still get `80%` penalty.

PoC

The component `(Math.sqrt(unlockPeriod*_SCALER)/_SCALER)<=sqrt(540*24*3600*1e18)/1e18=0` is always null, so only `20*_SCALER` is left, which always yields a `20%` unlocking percentage.

Mitigation

Remove the extra `_SCALER`.

```
function _getUnlockingPercentage(uint128 unlockPeriod) internal pure returns
 ↪  (uint256 unlockingPercentageBP) {
    unlockingPercentageBP = (
        _PERCENT_TO_BP
            * (
                ((80 * _SCALER) / Math.sqrt(540 * _SCALER)) *
 ↪  (Math.sqrt(unlockPeriod * _SCALER))
                    + 20 * _SCALER
            )
    ) / _SCALER;
}
```

Discussion

0xPilou

Fix for this issue is included in the following PR : https://github.com/superfluid-finance/fluid/pull/6

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/superfluid-finance/fluid/pull/6

Issue H-2: FluidLocker::_getUnlockingPercentage() uses 540 instead of 540days leading to stuck funds as the unlocking percentage will be bigger than 100% and underflow

Source: https://github.com/sherlock-audit/2024-11-superfluid-locking-contract-judging/issues/21

Found by

0x73696d616f, Drynooo, merlinboii, zxriptor

Summary

FluidLocker::_getUnlockingPercentage() calculates the amount to unlock when unvesting via the FluidLocker. It incorrectly uses 540 instead of 540days, yielding a massive error such that the unlocking percentage will be much bigger than 10_000 and underflow.

```
function _getUnlockingPercentage(uint128 unlockPeriod) internal pure returns
 ↪  (uint256 unlockingPercentageBP) {
    unlockingPercentageBP = (
        _PERCENT_TO_BP
            * (
                ((80 * _SCALER) / Math.sqrt(540 * _SCALER)) *
 ↪  (Math.sqrt(unlockPeriod * _SCALER) / _SCALER)
                    + 20 * _SCALER
            )
    ) / _SCALER;
}
```

Note: due to other bugs in the calculation it will not revert.

Root Cause

In FluidLocker:388 it uses 540 instead of 540days.

Internal pre-conditions

None.

External pre-conditions

None.

Attack Path

1. User unlocks their Fluid from the `FluidLocker`, but it reverts because of the mentioned underflow. The funds within the locker will be stuck unless the user instantly unlocks and takes a `80%` penalty.

Impact

User is forced to take a `80%` penalty or have the funds stuck.

PoC

The calculation is presented in the summary. Essentially, as `540` is used in the denominator, much smaller than the correct `540days` (which is the maximum unlock period, when the percentage becomes `100%`), the value will be much bigger than `10_000`.

As the unlocking percentage is bigger than `10_000`, the unlock flow rate

```
unlockFlowRate = (globalFlowRate *
↪  int256(_getUnlockingPercentage(unlockPeriod))).toInt96()
                / int256(_BP_DENOMINATOR).toInt96();
```

will be bigger than the global flow rate, so it reverts when calculating the tax flow rate `taxFlowRate=globalFlowRate-unlockFlowRate;`.

Mitigation

Use `540days` instead of `540`.

Discussion

0xPilou

Fix for this issue is included in the following PR : https://github.com/superfluid-finance/fluid/pull/6

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/superfluid-finance/fluid/pull/6

Issue H-3: `Fontaine` never stops the flows to the tax and recipient, so the buffer component of the flows will be lost

Source: https://github.com/sherlock-audit/2024-11-superfluid-locking-contract-judging/issues/36

Found by

0x73696d616f

Summary

`Superfluid` flows reserve 4 hours of the stream flow rate as a buffer for liquidations, and returns this when the flow is closed.

However, Fontaine::initialize() never actually stops the flows, which means the deposit buffer will never be reclaimed, taking the recipient and the tax pool the loss.

Root Cause

In `Fontaine::initialize()`, the flows are never stopped, which means the buffer will not be returned.

Internal pre-conditions

None.

External pre-conditions

None.

Attack Path

1. User unlocks their `Fluid` from the `Locker` by calling `FluidLocker::unlock()` with a non null unlocking period.

Impact

The recipient and the tax distribution pool take the loss as they do not receive all their funds (the deposit buffer is never sent to them).

PoC

Add the following logs to `Fontaine.sol` and run `forgetest--mttestVestUnlock-vvvv`. `Alice` sends `10_000e18` to the fontaine, but only `9996.8e18` are left in the fontaine as a part of them are reserved for the buffer which will never be collected back.

```
function initialize(address unlockRecipient, int96 unlockFlowRate, int96 taxFlowRate) external initializer {
    // Ensure recipient is not a SuperApp
    if (ISuperfluid(FLUID.getHost()).isApp(ISuperApp(unlockRecipient))) revert CANNOT_UNLOCK_TO_SUPERAPP();

    console2.log("fontaine balance pre distributeFlow", FLUID.balanceOf(address(this)));

    // Distribute Tax flow to Staker GDA Pool
    FLUID.distributeFlow(address(this), TAX_DISTRIBUTION_POOL, taxFlowRate);

    console2.log("fontaine balance pre createFlow", FLUID.balanceOf(address(this)));

    // Create the unlocking flow from the Fontaine to the locker owner
    FLUID.createFlow(unlockRecipient, unlockFlowRate);

    console2.log("fontaine balance after createFlow", FLUID.balanceOf(address(this)));
}
```

Mitigation

Add a way to stop the flow and receive the deposit back.

Discussion

0xPilou

Fix for this issue is included in the following PR : https://github.com/superfluid-finance/fluid/pull/8

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits: https://github.com/superfluid-finance/fluid/pull/8

Issue M-1: An attacker may DoS user Fluid balance increases by frontrunning `FluidLocker::claim()` calls and calling `EP_PROGRAM_MANAGER::batchUpdateUserUnits()` directly

Source: https://github.com/sherlock-audit/2024-11-superfluid-locking-contract-judging/issues/19

Found by

0x73696d616f

Summary

FluidLocker::claim() connects to the fluid pool and then calls `EPProgramManager::batchUpdateUserUnits()` to verify the signature and update user points.

However, any attacker may frontrun this call and call directly `EPProgramManager::batchUpdateUserUnits()`, spending the signature (nonce) and making the claim transaction revert.

As the Fluid balances only increase whenever the user connects to the pool, the user balance will be 0.

An attacker will profit from this because a large tax amount may be coming from a user instantly unlocking or a program being stopped, which transfer immediately funds to the stakers by calling `FLUID.distributeToPool(address(this),TAX_DISTRIBUTION_POOL,amount);`.

As the user wanted to stake, they needed Fluid balance to do so, but as the attacker DoSed their claim transaction, they will not have balance to stake and will completely miss these large instant funds. Even if users had already staked before, this claim transaction could be increasing their points (by increasing their program points, they could get more Fluid and stake more Fluid to get more staker points), so an attacker profits from not allowing the user to do so just before the tax is distributed.

Note1: the tax is distributed pro-rata to the units of each user, so if the attacker denies a user increasing their funds, the attacker will receive a larger share of the funds.

Root Cause

`EP_PROGRAM_MANAGER::updateUserUnits()` may be called directly allowing attackers to DoS `FluidLocker::claim()`.

Internal pre-conditions

None.

External pre-conditions

None.

Attack Path

1. Attacker spots that a user is unlocking or a program is coming to an end and a large sum of tax is coming.
2. Attacker frontruns users that are going to increase their Fluid balance in the Locker by calling `FluidLocker::claim()` so that the Locker connects to the program pool and receives the corresponding Fluid tokens.
3. Due to the `FluidLocker::claim()` call reverting, users will have less Fluid in the Locker and will stake less funds, getting less points, which means the attacker will get a bigger share of the incoming TAX distribution.

Impact

Attacker profits from having a bigger share of the TAX distributions and users lose this share.

PoC

The only way to collect the Locker to the program pool is by calling `FluidLocker::claim()` which may be DoSed by calling EP_PROGRAM_MANAGER.updateUserUnits()‘ directly using the same signature.

```
function claim(uint256 programId, uint256 totalProgramUnits, uint256 nonce, bytes memory stackSignature)
    external
    nonReentrant
{
    // Get the corresponding program pool
    ISuperfluidPool programPool = EP_PROGRAM_MANAGER.getProgramPool(programId);
    if (!FLUID.isMemberConnected(address(programPool), address(this))) {
        // Connect this locker to the Program Pool
        FLUID.connectPool(programPool);
    }

    // Request program manager to update this locker's units
    EP_PROGRAM_MANAGER.updateUserUnits(lockerOwner, programId, totalProgramUnits,
 ↪  nonce, stackSignature);

    emit IFluidLocker.FluidStreamClaimed(programId, totalProgramUnits);
}
```

Mitigation

The locker should have a separate method to connect to the pool.

Discussion

0xPilou

Fix for this issue is included in the following PR : https://github.com/superfluid-finance/fluid/pull/9

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits: https://github.com/superfluid-finance/fluid/pull/9

Issue M-2: `FluidLocker::_getUnlockingPercentage()` divides before multiplying, suffering a significant precision error

Source: https://github.com/sherlock-audit/2024-11-superfluid-locking-contract-judging/issues/22

Found by

0x73696d616f

Summary

FluidLocker::_getUnlockingPercentage() is calculated as:

```
function _getUnlockingPercentage(uint128 unlockPeriod) internal pure returns
 ↪  (uint256 unlockingPercentageBP) {
    unlockingPercentageBP = (
        _PERCENT_TO_BP
            * (
                ((80 * _SCALER) / Math.sqrt(540 * _SCALER)) *
 ↪  (Math.sqrt(unlockPeriod * _SCALER) / _SCALER)
                    + 20 * _SCALER
            )
    ) / _SCALER;
}
```

As can be seen, it divides before multiplying, leading to precision loss.

The loss is always `((80*1e18)/Math.sqrt(540*24*3600*1e18))=1712139.4821`, so the 0.4821 component is discarded. This corresponds to `0.4821*sqrt(540*24*3600*1e18)*100/1e18=0.00032929935` BPS.

Note: this calculation assumed the other 2 issues are fixed.

As this loss is present in every calculation and it will make a 1 BPS different in many instances, it is significant. For example, if the maximum duration is picked, instead of `10000` BPS, it will actually be `9999BPS` and 1 BPS goes to the TAX pool. If the user unlocks for example 1e5 USD, this is a 10 USD loss. As it will happen frequently, the loss will acumulate.

Root Cause

In `FluidLocker::388`, it divides before multiplying.

Internal pre-conditions

None.

External pre-conditions

None.

Attack Path

1. User calls unlock with vesting period, but due to the precision loss it rounds down and 1 BPS more goes to the tax distribution pool.

Impact

User suffers a 1 BPS loss of funds. For example, 1e5 USD will yield a 10 USD loss.

PoC

Presented in the summary.

Mitigation

Multiply before dividing as it will never overflow.

Discussion

0xPilou

Fix for this issue is included in the following PR :
https://github.com/superfluid-finance/fluid/pull/6

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/superfluid-finance/fluid/pull/6

Issue M-3: A malicious user may unlock instantly all the funds from the `FluidLocker` when no one is staking in the Tax pool

Source: https://github.com/sherlock-audit/2024-11-superfluid-locking-contract-judging/issues/24

Found by

0x73696d616f

Summary

The missing check in FluidLocker::_instantUnlock() for stakers in the tax pool allows users to unlock all their funds without paying any tax instantly. This happens because whenever there are 0 stakers in the tax pool, the flow rate is set to 0 and it does not revert, so the user can loop unlocking until all funds are withdrawn.

Note: `FluidEPProgramManager::stopFunding()` also has issues related to 0 stakers in the program or tax pools.

Note2: the mentioned assumptions in the readme about assuming there are stakers before calls only refer to `FluidEPProgramManager::startFunding()` and `Fountaine::initialize()`, not these 2 flows mentioned here.

Root Cause

In `FluidLocker::_instantUnlock()` there is a missing check for 0 stakers in the tax pool.

Internal pre-conditions

1. There are 0 stakers in the tax pool.

External pre-conditions

None.

Attack Path

1. User loops `FluidLocker::unlock()` with a null unlocking period and instantly withdraws all their funds.

Impact

The user is able to unlock all their funds without paying any tax.

PoC

Add the following test to `FluidLocker.t.sol`.

```
function test_POC_InstantUnlock_WithoutFees() external {
    _helperFundLocker(address(aliceLocker), 10_000e18);

    assertEq(_fluidSuperToken.balanceOf(address(ALICE)), 0, "incorrect Alice bal before op");
    assertEq(_fluidSuperToken.balanceOf(address(aliceLocker)), 10_000e18, "incorrect Locker bal before op");

    _helperUpgradeLocker();

    vm.startPrank(ALICE);
    for (uint i = 0; i < 30; i++) {
        aliceLocker.unlock(0, ALICE);
    }

    assertGt(_fluidSuperToken.balanceOf(address(ALICE)), 9.98e21, "incorrect Alice bal after op");
}
```

Mitigation

Check if the pools have 0 units and revert if so.

Discussion

0xPilou

Fix for this issue is included in the following PR : https://github.com/superfluid-finance/fluid/pull/7

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits: https://github.com/superfluid-finance/fluid/pull/7

Disclaimers

Sherlock does not provide guarantees nor warranties relating to the security of the project.

Usage of all smart contract software is at the respective users’ sole risk and is the users’ responsibility.