SHERLOCK

SHERLOCK SECURITY REVIEW FOR

Prepared for:	Tokensoft
Prepared by:	Sherlock
Lead Security Expert:	stopthecap
Dates Audited:	July 17 - July 21, 2023
Prepared on:	September 4, 2023

Introduction

Enterprise services and tools for leading blockchain foundations. The only chain-agnostic compliance platform. We help Projects and Communities bootstrap to build the future of Web3.

Scope

Repository: SoftDAO/contracts

Branch: crosschain

Commit: 291df55ddb0dbf53c6ed4d5b7432db0c357ca4d3

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

Medium	High
6	1

Issues not fixed or acknowledged

Medium	High
0	0

Security experts who found valid issues

jkoppel
Yuki
0xMAKEOUTHILL
auditsea
BenRai
AkshaySrivastav
magellanXtrachev
stopthecap
kutugu

smbv-1919
Avci
pks_
Musaka
jah
ni8mare
r0bert
Czar102
pengun
circlelooper
Juntao
blackhole
caventa
0xbranded
p12473
0xDjango
0xhacksmithh
n33k
qbs
pep7siup
tsvetanovv
Vagner
GREY-HAWK-REACH
mau
dany.armstrong90
twicek
VAD37
0xDanielH
y1cunhui
p-tsanev
0xlx

Issue H-1: "Votes" balance can be increased indefinitely in multiple contracts

Source: https://github.com/sherlock-audit/2023-06-tokensoft-judging/issues/41

Found by

0xDanielH, 0xDjango, 0xbranded, 0xlx, AkshaySrivastav, BenRai, Czar102, Musaka, VAD37, Yuki, caventa, dany.armstrong90, jah, jkoppel, kutugu, magellanXtrachev, ni8mare, p-tsanev, p12473, pengun, r0bert, stopthecap, twicek, y1cunhui

Summary

The "voting power" can be easily manipulated in the following contracts:

- `ContinuousVestingMerkle`
- `PriceTierVestingMerkle`
- `PriceTierVestingSale_2_0`
- `TrancheVestingMerkle`
- `CrosschainMerkleDistributor`
- `CrosschainContinuousVestingMerkle`
- `CrosschainTrancheVestingMerkle`
- All the contracts inheriting from the contracts listed above

This is caused by the public `initializeDistributionRecord()` function that can be recalled multiple times without any kind of access control:

```
function initializeDistributionRecord(
  uint32 _domain, // the domain of the beneficiary
  address _beneficiary, // the address that will receive tokens
  uint256 _amount, // the total claimable by this beneficiary
  bytes32[] calldata merkleProof
) external validMerkleProof(_getLeaf(_beneficiary, _amount, _domain),
↪  merkleProof) {
  _initializeDistributionRecord(_beneficiary, _amount);
}
```

Vulnerability Detail

The `AdvancedDistributor` abstract contract which inherits from the `ERC20Votes`, `ERC20Permit` and `ERC20` contracts, distributes tokens to beneficiaries with voting-while-vesting and administrative controls. Basically, before the tokens are

vested/claimed by a certain group of users, these users can use these ERC20 tokens to vote. These tokens are minted through the `_initializeDistributionRecord()` function:

```
function _initializeDistributionRecord(
  address beneficiary,
  uint256 totalAmount
) internal virtual override {
  super._initializeDistributionRecord(beneficiary, totalAmount);

  // add voting power through ERC20Votes extension
  _mint(beneficiary, tokensToVotes(totalAmount));
}
```

As mentioned in the Tokensoft Discord channel these ERC20 tokens minted are used to track an address's unvested token balance, so that other projects can utilize 'voting while vesting'.

A user can simply call as many times as he wishes the `initializeDistributionRecord()` function with a valid merkle proof. With each call, the `totalAmount` of tokens will be minted. Then, the user simply can call `delegate()` and delegate those votes to himself, "recording" the inflated voting power.

Impact

The issue totally breaks the 'voting while vesting' design. Any DAO/project using these contracts to determine their voting power could be easily manipulated/exploited.

Code Snippet

- https://github.com/sherlock-audit/2023-06-tokensoft/blob/main/contracts/contracts/claim/ContinuousVestingMerkle.sol#L43-L53
- https://github.com/sherlock-audit/2023-06-tokensoft/blob/main/contracts/contracts/claim/PriceTierVestingMerkle.sol#L49-L59
- https://github.com/sherlock-audit/2023-06-tokensoft/blob/main/contracts/contracts/claim/PriceTierVestingSale_2_0.sol#L91-L95
- https://github.com/sherlock-audit/2023-06-tokensoft/blob/main/contracts/contracts/claim/TrancheVestingMerkle.sol#L39-L49
- https://github.com/sherlock-audit/2023-06-tokensoft/blob/main/contracts/contracts/claim/abstract/CrosschainMerkleDistributor.sol#L46-L53

Tool used

Manual Review

Recommendation

Only allow users to call once the `initializeDistributionRecord()` function. Consider using a mapping to store if the function was called previously or not. Keep also in mind that fully vested and claimed users should not be able to call this function and if they do, the total amount of tokens that should be minted should be 0 or proportional/related to the amount of tokens that they have already claimed.

Discussion

cr-walker

Great find! We need to preserve the ability to re-initialize distribution records (e.g. if a merkle root changes), so I believe something like this is the best fix:

```
function _initializeDistributionRecord(
  address beneficiary,
  uint256 totalAmount
) internal virtual override {
  super._initializeDistributionRecord(beneficiary, totalAmount);

  uint256 currentVotes = balanceOf(beneficiary);
  uint256 newVotes = tokensToVotes(totalAmount);

  if (currentVotes > newVotes) {
    // reduce voting power through ERC20Votes extension
    _burn(beneficiary, currentVotes - newVotes);
  } else if (currentVotes < newVotes) {
    // increase voting power through ERC20Votes extension
    _mint(beneficiary, newVotes - currentVotes);
  }
}
```

cr-walker

Fixed: https://github.com/SoftDAO/contracts/pull/9

maarcweiss

Fixed by keeping the ability to re-initialize the distribution records, but not increasing the voting power of the user

Issue M-1: setVoteFactor() does not change existing supply of votes. As a result, some may be unable to withdraw.

Source: https://github.com/sherlock-audit/2023-06-tokensoft-judging/issues/55

Found by

0xDjango, 0xbranded, AkshaySrivastav, BenRai, Czar102, Yuki, auditsea, caventa, jkoppel, mau, p12473, pengun, r0bert

Summary

`AdvancedDistributor.setVoteFactor()` does not change existing supply of vote tokens. If it is called before all distribution records have been initialized, there will be a skew between those who initialized before and those who initialized after. Further, if it is increased, those who initialized before will not have enough vote tokens to withdraw.

Vulnerability Detail

Increase scenario (very bad):

1. Owner makes airdrop and sets vote factor to 1. Many people get a claim to 1000 airdrop tokens.
2. People initialize their distribution records. They are minted 1000 vote tokens.
3. Owner sets vote factor to 2
4. The airdrop tokens vest
5. No-one who already initialized their distribution record can withdraw anything because they don't have enough vote tokens. (Vote tokens are burned when executing a claim.)

Decrease scenario (less bad):

1. Owner makes an airdrop for 1000 people managed by a CrosschainMerkleDistribution and sets vote factor to 1000
2. User Speedy Gonzalez calls initializeDistributionRecord() for himself
3. Owner decides to change the vote factor to 1 instead
4. All other users only get 1 voting token, but Speedy still has 1000

Impact

Increase scenario

If the vote factor is increased after deploying the contract, some people will not be able to withdraw, period.

It is still possible, however, for the owner of the contract to sweep the contract and manually give people their airdrop.

Decrease scenario

Cannot change vote factor after deploying contract without skewing existing votes.

Note there is no other mechanism to mint or burn vote tokens to correct this.

There is no code that currently uses voting, so this is potentially of no consequence.

However, presumably the voting functionality exists for a reason, and will be used by other code. In particular, the implementation of adjust() takes care to preserve people's number of voting tokens. As the distributor contracts are not upgradeable, this means no fair elections can be run atop airdrops deployed with the current code after setVoteFactor is called.

Code Snippet

https://github.com/sherlock-audit/2023-06-tokensoft/blob/main/contracts/contracts/claim/abstract/AdvancedDistributor.sol#L181

```
function setVoteFactor(uint256 _voteFactor) external onlyOwner {
    voteFactor = _voteFactor;
    emit SetVoteFactor(voteFactor);
  }
```

setVoteFactor does not change supply

https://github.com/sherlock-audit/2023-06-tokensoft/blob/main/contracts/contracts/claim/abstract/AdvancedDistributor.sol#L77

```
function _initializeDistributionRecord(
   address beneficiary,
   uint256 totalAmount
 ) internal virtual override {
   super._initializeDistributionRecord(beneficiary, totalAmount);

   // add voting power through ERC20Votes extension
   _mint(beneficiary, tokensToVotes(totalAmount));
 }
```

Voting tokens are minted at distribution record initialization time.

https://github.com/sherlock-audit/2023-06-tokensoft/blob/main/contracts/contracts/claim/abstract/AdvancedDistributor.sol#L87

```
  function _executeClaim(
  address beneficiary,
  uint256 totalAmount
) internal virtual override returns (uint256 _claimed) {
  _claimed = super._executeClaim(beneficiary, totalAmount);

  // reduce voting power through ERC20Votes extension
  _burn(beneficiary, tokensToVotes(_claimed));
}
```

`tokensToVotes` uses the current voteFactor. If it has increased since someone's vote tokens were minted, they will not have enough tokens to burn, and so `executeClaim` will revert.

Tool used

Manual Review

Recommendation

Do not use separate voting tokens for votes; just use the amount of unclaimed token

Discussion

cr-walker

This is a valid issue.

The admin functions like `setVoteFactor()` should only be called by admins who know what they are doing, but I agree that this would result in unexpected behavior like locking funds due to burning too many vote tokens.

Proposed solution:

- The reason we are using an internal votes token is to allow delegation, but I agree that using the unclaimed token would be simpler and generally better
- we will either remove the `setVoteFactor` method or allow users to reset their voting power, e.g. using something like the solution to: https://github.com/sherlock-audit/2023-06-tokensoft-judging/issues/188

cr-walker

Fixed by https://github.com/SoftDAO/contracts/pull/10

maarcweiss

Fixed by adding a function to update the voting power when claiming and initializing the distribution.

Issue M-2: Because of rounding issues, users may not be able to withdraw airdrop tokens if their claim has been adjust()'ed upwards

Source: https://github.com/sherlock-audit/2023-06-tokensoft-judging/issues/56

Found by

jkoppel

Summary

In order for a user to withdraw their claim, they must have enough voting tokens. However, because of rounding issues, if their voting shares are granted in multiple stages, namely by the owner adjust()-ing their share upwards, they will not have enough.

Vulnerability Detail

1. Owner creates airdrop and grants a user a claim of 1000 tokens. The voting factor is 5, and the fractionDenominator is set to 10000.
2. User initializes their distribution record. They are minted 1000*5/10000 = 0 voting tokens.
3. Owner adjusts everyone's claim up to 1000. Each user is minted another 1000*5/10000=0 voting tokens.
4. User fully vests
5. User cannot withdraw anything because, in order to withdraw, they must burn 2000*5/10000= 1 voting token.

Impact

Unless all grants and positive adjust()'s are for exact multiples of fractionDenominator, users will be prevented from withdrawing after an upwards adjustment.

Note that many comments give example values of 10000 for fraction denominator and 15000 for voteFactor. Since the intention is to use `voteFactor`'s which are not multiples of fractionDenonimator, rounding issues will occur.

Code Snippet

Rounding in tokensToVotes

https://github.com/sherlock-audit/2023-06-tokensoft/blob/main/contracts/contracts/claim/abstract/AdvancedDistributor.sol#L73

```
function tokensToVotes(uint256 tokenAmount) private view returns (uint256) {
  return (tokenAmount * voteFactor) / fractionDenominator;
}
```

_inititializeDistributionRecord and adjust() both use `tokensToVotes` to mint

https://github.com/sherlock-audit/2023-06-tokensoft/blob/main/contracts/contracts/claim/abstract/AdvancedDistributor.sol#L84C1-L85C1

https://github.com/sherlock-audit/2023-06-tokensoft/blob/main/contracts/contracts/claim/abstract/AdvancedDistributor.sol#L126

tokensToVotes is again used to burn when executing a claim

https://github.com/sherlock-audit/2023-06-tokensoft/blob/main/contracts/contracts/claim/abstract/AdvancedDistributor.sol#L87

```
function _executeClaim(
  address beneficiary,
  uint256 totalAmount
) internal virtual override returns (uint256 _claimed) {
  _claimed = super._executeClaim(beneficiary, totalAmount);

  // reduce voting power through ERC20Votes extension
  _burn(beneficiary, tokensToVotes(_claimed));
}
```

Tool used

Manual Review

Recommendation

Base votes on share of unclaimed tokens and not on a separate token.

Discussion

cr-walker

Good catch.

Solution: we'll burn the minimum of the expected quantity and current balance to get around these rounding issues.

cr-walker

Fixed by https://github.com/sherlock-audit/2023-06-tokensoft-judging/issues/56

dot-pengun

Escalate

This issue occurs when the admin sets the `voteFactor` to very low or which are not multiples. In several test files that the `voteFactor` is set to `2n * 10n ** 18n`, `2n * 10n ** 4n`, etc. (this is also mentioned in the issue above), so I believe the issue is essentially an admin error.

Also resolution in #41 resolves this issue.(https://github.com/SoftDAO/contracts/pull/9/commits/0cd8ff408632eabfc363da43255ffd4d2a8bd73e)

sherlock-admin2

> Escalate
>
> This issue occurs when the admin sets the `voteFactor` to very low or which are not multiples. In several test files that the `voteFactor` is set to `2n * 10n ** 18n`, `2n * 10n ** 4n`, etc. (this is also mentioned in the issue above), so I believe the issue is essentially an admin error.
>
> Also resolution in #41 resolves this issue.(https://github.com/SoftDAO/contracts/pull/9/commits/0cd8ff408632eabfc363da43255ffd4d2a8bd73e)

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

jkoppel

dot-penguin points out some interesting information about the test values. But, as I'll explain below, I think relying on such information is at odds with the purpose of this entire contest.

While the example scenario used a low number, the issue appears with any value of voteFactor which is not an exact multiple of fractionDenominator. Comments suggest in multiple places (e.g.: https://github.com/sherlock-audit/2023-06-tokensoft/blob/main/contracts/contracts/claim/TrancheVestingSale_2_0.sol#L31) the values of 15000 for voteFactor and 10000 for fractionDenominator, which would exhibit this issue.

If you call this invalid admin error, then any bug that occurs in some settings but not all is also admin error. This bug is present for the overwhelming majority of possible settings of voteFactor, and there is nothing to indicate that only multiples should be used.

The purpose of audits is to catch bugs not already found. Saying "the tests didn't use values that found the bug" is not a reason for non-validity.

It is irrelevant that there is a larger design change that fixes both this and other issues.

dot-pengun

I agree that this is a vulnerability, I just think the definition of admin input validation in sherlock's docs is vague. I'm not sure if input validation is just for rug pulls or if it also includes things that can harm the protocol like this.

> Admin Input/call validation: Protocol admin is considered to be trusted in most cases, hence issues where Admin incorrectly enters an input parameter. Example: Make sure interestPerMin > 1 ether as it is an important parameter. This is not a valid issue.

jkoppel

As I mentioned in the Discord, this is a contest where, for every single issue reported, I can see an argument by which that issue should be invalid. (For example, because of the Sweepable functionality, admin can manually fix any issue of funds getting stuck.) I agree that Sherlock's docs should be more clear about this --- and they have become better in my moderate amount of time on the platform. But in the past, errors like this have been deemed valid.

I have a software verification background, and I think there's a pretty simple definition that can apply here:

1. Each function makes a promise: if you pass in inputs that satisfy its precondition, then some desired outcome will occur
2. If a user calls a function in a way that doesn't satisfy its preconditions, nothing bad should happen
3. If an admin calls a function in a way that doesn't satisfy its preconditions, all bets are off
4. If an admin calls a function in a way that **does** satisfy its preconditions, no unintended bad effects should occur.

So the relevant question is: is "voteFactor must be a perfect multiple of fractionDenominator" in the communicated precondition for the various constructors? I think it's hard to argue yes, given that the example value violates that property.

Czar102

Please note that it's `tokenAmount * voteFactor` that needs to be a multiply of `fractionDenominator`. Hence, the values provided in the comment, i.e. `voteFactor` is 15000, it does not cause the bug right away. The admin just needs to set all claim amounts to an even number, then `voteFactor * tokenAmount` will always be a

multiple of 30000 at the end of the vesting, making it divisible by the `fractionDenominator`. Auditors were free to assume that nothing is broken, as long as the admin inputs all pairs `(tokenAmount, voteFactor)` (`voteFactor` is constant), which is a question of admin input validation and comments do not give any example suggesting the admin could want to input invalid values here.

jkoppel

Consider a TranchVestingSale or PriceTierVestingSale airdrop where users control the initial amount granted.

User buys 1337 tokens.

Admin calls `adjust(user, 5000)`.

But this causes the bug in question.

Actually, the admin should have called `adjust(user, 5000 - (getDistributionRecord(user).total % 2))`.

If calling `adjust(user, 5000)` is indeed considered an invalid input despite there being nothing in the docs or comments suggesting that calling `adjust(user, 5000)` may cause trapped funds, then this would be indeed an invalid issue.

But I suspect that most would not be willing to bite that bullet.

dot-pengun

I think this issue is not fundamentally caused by `voteFactor` not being a multiple of `fractionDenominator`, but rather by poor handling on `adjust`. In other words, it happens because `initializeDistributionRecord` is not performed correctly after the adjustment. Therefore, the protocol team fixed #41, and the revised commit shows that the issue is gone.

Czar102

If the user buys 1337 tokens, they get 2005 votes. Adjusting by 5000 should give the user 7500 more votes. Hence, the user has 7505, which is exactly what they need to withdraw all the amount. If the adjusted amount was positive and odd, the argument would make sense. Nice find.

The admin can still call `adjust(user, -1)`, `adjust(user, -1)`, `adjust(user, 2)` to resolve this. Hence, it is only gas that the admin loses, there is no fund loss or irreversible state change.

jkoppel

Thanks for working out my example more @Czar102. Yes, I did screw up. The argument to adjust() needs to be odd. So I can amend my statement: one would have to bite the bullet that `adjust(user, amountToGiveToManyUsers / n)` is invalid input, and so is `adjust(user, someBaseAmount * 2 / 3)`, with the correct version being `adjust(user, amountToGiveToManyUsers / n -`

`(getDistributionRecord(user).total % 2))`. This is not quite as large as a bullet to bite, but still a huge toothache if one does.

Re: @dot-pengun Yes, the logic in the fix to #41 is a good approach. adjust() is completely independent of initializeDistributionRecord(), so one solution is to apply the same fix here.

It is true that the admin can work around the issue if they understand it. But that argument also applies to literally every other issue in this contest.

KuTuGu

Escalate I want to emphasize that the precision of ERC20 tokens and votes is 18 decimals. The warden is wrong in the example mentioned above, the true value should be `1000e18 * 5 / 10000`, not `1000 * 5 / 10000`. For the dust value of 1000, tokensToVotes are 0, but this is meaningless because the tokenAmount itself is dust. In addition, if tokenAmount = 1.11...1e18, although there is a precision error in division, the error is only 1wei, which can be ignored. So I think the issue is low.

sherlock-admin2

> Escalate I want to emphasize that the precision of ERC20 tokens and votes is 18 decimals. The warden is wrong in the example mentioned above, the true value should be `1000e18 * 5 / 10000`, not `1000 * 5 / 10000`. For the dust value of 1000, tokensToVotes are 0, but this is meaningless because the tokenAmount itself is dust. In addition, if tokenAmount = 1.11...1e18, although there is a precision error in division, the error is only 1wei, which can be ignored. So I think the issue is low.

The escalation could not be created because you are not exceeding the escalation threshold.

You can view the required number of additional valid issues/judging contest payouts in your Profile page, in the Sherlock webapp.

jkoppel

@KuTuGu What is the true value you are referring to?

I explained why calling `adjust(user, someBaseAmount * 2 / 3)` on a fully-vested user would cause a user to become unable to withdraw 100% of their funds. Can you find some flaw in this reasoning? Are you saying that 100% of a user's claimable token is "dust?"

KuTuGu

I'm saying that tokens have 1e18 decimals, 1000 tokens are 1000e18, not 1000, it doesn't have a result of 0 when calculating tokensToVotes, and since fractionDenominator is a multiple of 10, there is no precision error here

jkoppel

Sorry, I'm confused. I don't understand how that is relevant to the issue.

Let's be super-specific. Here's a Chisel session mimicking the tracking of vote tokens:

1. User gets x=1000e18 / 3 tokens

```
uint256 x = 1000e18/3;
```

2. User gets z=x*voteFactor/fractionDenominator voting tokens

```
uint256 z = x * 15000 / 10000;
```

3. Admin calls adjust `adjust(user, 1000e18/3)`

```
x += 1000e18 / 3; z += (1000e18 / 3) * 15000 / 10000;
```

4. User fully vests
5. User tries to claim, burning all vote tokens

```
z -= x*votingFactor / fractionDenominator;

Traces:
  [3204] 0xBd770416a3345F91E4B34576cb804a576fa48EB1::run()
    ← "Arithmetic over/underflow"
```

Are you saying this is not the math used to mint and burn voting shares, or that an arithmetic underflow when withdrawing is not a problem?

KuTuGu

As I said before, there is an error in division, but it is only 1wei and there is no significant loss

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import "forge-std/Test.sol";
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "../contracts/claim/BasicDistributor.sol";

contract MToken is ERC20 {
    constructor() ERC20("", "") {}
}

contract CounterTest is Test {
    MToken public token;
    BasicDistributor public distributor;
    uint256 constant amount = 1000 ether;
    address constant user = address(0xdead);
```

```
    function setUp() public {
        address[] memory recipients = new address[](1);
        uint256[] memory amounts = new uint256[](1);
        recipients[0] = user;
        amounts[0] = amount / 3;
        token = new MToken();
        distributor = new BasicDistributor(token, amount / 3, "", 1.5e18,
↪   recipients, amounts);
        deal(address(token), address(distributor), amount);
    }

    function testRoundIssue() public {
        assertEq(distributor.getClaimableAmount(user), amount / 3);
        assertEq(token.balanceOf(user), 0);
        distributor.adjust(user, int256(amount / 3 * 2));
        assertEq(distributor.getClaimableAmount(user), amount / 3 * 3);
        assertEq(token.balanceOf(user), 0);
        distributor.claim(user);
        assertEq(distributor.getClaimableAmount(user), 0);
        assertEq(token.balanceOf(user), amount / 3 * 3);
        assertEq(amount * 3 / 3 - token.balanceOf(user), 1);
    }
}
```

jkoppel

@KuTuGu Cool! That test behaves exactly as I expect! Now let's try something actually related to this issue.

For a little bit of context, this issue:

1. Causes claim attempts to revert
2. Requires that tokenAmount*votingFactor is not perfectly divisible by fractionDenominator.
3. Involves a rounding error in voting tokens, not the claimed token

The scenario I wrote above:

1. Uses a votingFactor of 15000, which is the one suggested by documentation
2. Requires distributing an odd number of tokens to the user, and then adjusting by an odd number of tokens.

And this test:

1. Does not look at voting tokens
2. Does not check anything related to reversion

3. Uses a votingFactor 10^14 times higher than the one proposed, one which is a perfect multiple of votingFactor
4. Does not adjust by an odd number of tokens

It turns out that, if you run a test completely unrelated to the proposed issue, it will fail to exhibit the issue!

But it did successfully show that (amount/3)+(amount/3 * 2) == amount-1. There are also simpler tests of this.

Now let's modify it to actually relate to the issue:

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import "forge-std/Test.sol";
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "../contracts/claim/BasicDistributor.sol";

contract MToken is ERC20 {
    constructor() ERC20("", "") {}
}

contract CounterTest is Test {
    MToken public token;
    BasicDistributor public distributor;
    uint256 constant amount = 1000 ether;
    address constant user = address(0xdead);

    function setUp() public {
        address[] memory recipients = new address[](1);
        uint256[] memory amounts = new uint256[](1);
        recipients[0] = user;
        amounts[0] = amount / 3;
        token = new MToken();
        distributor = new BasicDistributor(token, amount / 3, "", 15000,
↪   recipients, amounts);
        deal(address(token), address(distributor), amount);
    }

    function testRoundIssue() public {
        distributor.adjust(user, int256(amount / 3));
        distributor.claim(user);
    }
}
```

Result:

```
Running 1 test for contracts/contracts/ContractTest.sol:CounterTest
[FAIL. Reason: ERC20: burn amount exceeds balance] testRoundIssue() (gas: 77926)
Test result: FAILED. 0 passed; 1 failed; finished in 2.63ms

Failing tests:
Encountered 1 failing test in contracts/contracts/ContractTest.sol:CounterTest
[FAIL. Reason: ERC20: burn amount exceeds balance] testRoundIssue() (gas: 77926)
```

In contrast, if you adjust by `1+amount/3` instead, then it does not revert.

So, as you can see, a very innocuous call to `adjust` renders the user unable to claim anything. This issue is valid.

KuTuGu

Sorry I confused the hard-coded parameter fractionDenominator for ContinuousVesting and BasicDistributor. You are correct, each adjust may produce 1wei error, resulting in failure to withdraw money.

hrishibhat

@jkoppel @dot-pengun Firstly thank you for the points around admin input error. I can see how the rule can be ambiguous at times, and an improvement on the same is pending.

About this issue, to summarize this seems to be creating temporary dos for user claims which can correct by the owner using adjust?

jkoppel

No, that's not an accurate summary.

The summary is that a very innocuous call to `adjust` creates a DOS for user claims, period.

The admin can fix this problem by sweeping the tokens back and distributing them manually, or by a very unintuitive sequence of admin calls which amounts to using another exploit to undo this bug.

Note that all other DOS vulnerabilities can be worked around in a similar manner: #55, #141, and #14. #141 in particular has a very easy workaround: instead of calling `setTotal(newTotal)`, call `setTotal(claimed); setTotal(newTotal)`

Admins are also able to work around the issue for #188, #143, #130, and #41.

SilentYuki

> In contrast, if you adjust by `1+amount/3` instead, then it does not revert.

As mentioned the rounding error can be prevented and depends mainly on the inputted value by the admin.

> So, as you can see, a very innocuous call to adjust renders the user unable to claim anything. This issue is valid.

Even if admin makes false call to adjust and rounding error occurs, the DoS is only temporarily as admin can re-adjust to fix the issue.

jkoppel

> Even if admin makes false call to adjust and rounding error occurs, the DoS is only temporarily as admin can re-adjust to fix the issue.

Yes, and similar is true of #55, #141, #14, #188, #143, #130, and #41. I don't see a way to mark this invalid without also marking all 7 of those other issues as invalid. Further, in this case, the proposed workaround using `adjust()` only works by exploiting buggy logic in the code.

hrishibhat

@jkoppel thanks for the context. So the way I see it, there is an error this would be an issue if `voteFactor` was set too low or not a multiple of fractionDenominator, and this value is not something that can be changed to fix this issue. correct? also there is no way to know what the best `voteFactor` would be as it could vary from case to case.

jkoppel

Correct, this is an issue if `voteFactor` is not an exact multiple of `fractionDenominator`. The protocol clearly intends to allow many different values for `voteFactor`. The docs suggest a value of 15000 for voteFactor and 10000 for fractionDenominator, which does enable the vulnerability.

hrishibhat

Result: Medium Unique Considering this issue a valid medium based on the comments above: https://github.com/sherlock-audit/2023-06-tokensoft-judging/issues/56#issuecomment-1684238186

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- dot-pengun: rejected

cr-walker

@sherlock-admin - here's the fix: https://github.com/SoftDAO/contracts/pull/11

maarcweiss

Fixed by wrapping the logic of the fix in #41 in the `_reconcileVotingPower` function and preventing rounding issues to fix that the user's were not be able to withdraw

Issue M-3: In `PriceTierVesting` there is no check if the Sequenzer for L2s is up when calling the oralce

Source: https://github.com/sherlock-audit/2023-06-tokensoft-judging/issues/64

Found by

Avci, BenRai, auditsea, magellanXtrachev, pks_, smbv-1919, stopthecap

Summary

Price from oracle on L2s can be invalid/stale if the sequencer is down. This could lead to users being able to claim tokens that they should not be able to claim.

Vulnerability Detail

If the sequencer of the L2s were to go offline the Chainlink oracle may return an invalid/stale price. This could lead to users being able to claim tokens that they should not be able to claim.

For example, if the last reference price recorded by the oracle was above the final tier price, this would have unlocked all tokens to be claimable. If the sequencer goes and in the meantime down the reference price falls below the final tier price , the oracle would still return the high price even though the price is lower now and not all tokens should be claimable.

It should always be checked if the sequencer is up before consuming any data from Chainlink. For more details on L2 Sequencer Uptime Feeds check the Chainlink docs(https://docs.chain.link/data-feeds/l2-sequencer-feeds) specify more details.

Impact

Receivers can claim tokens they should not be able to claim

Code Snippet

https://github.com/sherlock-audit/2023-06-tokensoft/blob/1f58ddb066ab383c416cfcbf95c9902683506e96/contracts/contracts/claim/abstract/PriceTierVesting.sol#L30-L45

Tool used

Manual Review

Recommendation

Include a check if the sequencer is up. If it is down, revert when calling `getVestedFraction` in `PriceTierVesting`

Discussion

BenRai1

Escalate

According to the judge, the issue was excluded because `PriceTierVesting.sol` is out of scope for the contest. Even though `PriceTierVesting.sol` was not explicitly mentioned as in scope, deriving from the following issue in the Teller contest, (https://github.com/sherlock-audit/2023-03-teller-judging/issues/328), `PriceTierVesting.sol` is implicitly in scope because `PriceTierVestingSale_2_0.sol` is in scope and it is inheriting from `PriceTierVesting`. This means all behaviour of `PriceTierVesting` that affects `PriceTierVestingSale_2_0` should also be in scope.

Therefore in my opinion, this issue should be considered as valid.

sherlock-admin2

> Escalate
>
> According to the judge, the issue was excluded because `PriceTierVesting.sol` is out of scope for the contest. Even though `PriceTierVesting.sol` was not explicitly mentioned as in scope, deriving from the following issue in the Teller contest, (https://github.com/sherlock-audit/2023-03-teller-judging/issues/328), `PriceTierVesting.sol` is implicitly in scope because `PriceTierVestingSale_2_0.sol` is in scope and it is inheriting from `PriceTierVesting`. This means all behaviour of `PriceTierVesting` that affects `PriceTierVestingSale_2_0` should also be in scope.
>
> Therefore in my opinion, this issue should be considered as valid.

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

jkoppel

Duplicate: #22, #38, #48, #218, #229, #38, #163

Shogoki

Regarding scoping:

In my opinion every contract in scope should be listed explicitly as in scope. I do get that it might makes sense to have some rules to have this contract implicitly in scope, but it creates uncertainty as it is nowhere documented. I raised this concern in discord here I will leave the final decision on the scope for this on sherlock, as there are good points for both sides, i guess. But the following discussions in discord should also be taken into account:

https://discord.com/channels/812037309376495636/1130514276570906685/1138286182254522531 https://discord.com/channels/812037309376495636/1130514276570906685/1131243195817283604

However this is decided at the end, we should take this as a chance to improve the guidelines for scope in future contests.

Regarding the Issue:

This can be a valid Medium

hrishibhat

Result: Medium Has duplicates Since `PriceTierVesting` is in scope because `PriceTierVestingSale_2_0` is in scope. This is a valid medium Agree with the @Shogoki comments that the scoping rules should be improved to avoid such confusion

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- BenRai1: accepted

cr-walker

Fixed by https://github.com/SoftDAO/contracts/pull/17

maarcweiss

Fixed by creating the `L2OracleWithSequencerCheck.sol` contract, as the contract that checks whether the sequencer is up.

Issue M-4: `SafeERC20.safeApprove` reverts for changing existing approvals

Source: https://github.com/sherlock-audit/2023-06-tokensoft-judging/issues/141

Found by

AkshaySrivastav, Juntao, Musaka, auditsea, blackhole, circlelooper, jah, kutugu, ni8mare

Summary

`SafeERC20.safeApprove` reverts when a non-zero approval is changed to a non-zero approval. The `CrosschainDistributor._setTotal` function tries to change an existing approval to a non-zero value which will revert.

Vulnerability Detail

The safeApprove function has explicit warning:

```
// safeApprove should only be called when setting an initial allowance,
// or when resetting it to zero. To increase and decrease it, use
// 'safeIncreaseAllowance' and 'safeDecreaseAllowance'
```

But still the `_setTotal` use it to change approval amount:

```
function _allowConnext(uint256 amount) internal {
  token.safeApprove(address(connext), amount);
}

/** Reset Connext allowance when total is updated */
function _setTotal(uint256 _total) internal virtual override onlyOwner {
  super._setTotal(_total);
  _allowConnext(total - claimed);
}
```

Impact

Due to this bug all calls to `setTotal` function of `CrosschainContinuousVestingMerkle` and `CrosschainTrancheVestingMerkle` will get reverted.

Tokensoft airdrop protocol is meant to be used by other protocols and the ability to change `total` parameter is an intended offering. This feature will be important for those external protocols due to the different nature & requirement of every airdrop.

But this feature will not be usable by airdrop owners due to the incorrect code implementation.

Code Snippet

https://github.com/sherlock-audit/2023-06-tokensoft/blob/main/contracts/contracts/claim/abstract/CrosschainDistributor.sol#L35-L45

Tool used

Manual Review

Recommendation

Consider using 'safeIncreaseAllowance' and 'safeDecreaseAllowance' instead of `safeApprove` in `_setTotal`.

Discussion

cr-walker

Good catch, updating the code to simply set approval to zero first and then reset approval. I don't think any of the reentrancy attacks that `safeApprove()` is worried about are relevant here (neither the owner nor the connect protocol are going to use this to rug the contract, they both have much more direct ways to take tokens if malicious!)

cr-walker

Fixed by https://github.com/SoftDAO/contracts/pull/12

maarcweiss

Fixed by approving to 0 before approving again by implementing the following line: `token.safeApprove(address(connext), 0);`

Issue M-5: CrosschainDistributor: Not paying relayer fee when calling xcall to claim tokens to other domains

Source: https://github.com/sherlock-audit/2023-06-tokensoft-judging/issues/143

Found by

0xhacksmithh, AkshaySrivastav, Czar102, GREY-HAWK-REACH, Vagner, jkoppel, kutugu, n33k, pengun, pep7siup, qbs, r0bert, tsvetanovv

Summary

CrosschainDistributor is not paying relayer fee when calling xcall to claim tokens to other domains. The transaction will not be relayed on target chain to finnalize the claim. User will not receive the claimed tokens unless they bump the transaction fee themself.

Vulnerability Detail

In `_settleClaim`, the CrosschainDistributor is using xcall to claim tokens to another domain. But relayer fee is not payed.

https://github.com/sherlock-audit/2023-06-tokensoft/blob/main/contracts/contracts/claim/abstract/CrosschainDistributor.sol#L78

```
id = connext.xcall(           // <------ relayer fee should be payed here
  _recipientDomain, // destination domain
  _recipient, // to
  address(token), // asset
  _recipient, // delegate, only required for self-execution + slippage
  _amount, // amount
  0, // slippage -- assumes no pools on connext
  bytes('') // calldata
);
```

Without the relayer fee, the transaction will not be relayed. The user will need to bump the relayer fee to finnally settle the claim by following the instructions here in the connext doc.

Impact

User will not receive their claimed tokens on target chain.

Code Snippet

https://github.com/sherlock-audit/2023-06-tokensoft/blob/main/contracts/contracts/claim/abstract/CrosschainDistributor.sol#L78

Tool used

Manual Review

Recommendation

Help user bump the transaction fee in Satellite.

Discussion

cr-walker

Hmm, this would be a valid issue in general but Connext is paying for relayer fees in this case, i.e. a zero fee cross-chain transaction is valid in this case (and on https://docs.connext.network/developers/guides/estimating-fees). @LayneHaber any thoughts on validity?

cr-walker

I am marking this as valid since auditors would not know the plan to have Connext pay this fee in this case and we'll change this anyway (it seems strictly better to make the function payable and pass along message value like this:

```
function _settleClaim(
    address _beneficiary,
    address _recipient,
    uint32 _recipientDomain,
    uint256 _amount
  ) internal virtual {
    bytes32 id;
    if (_recipientDomain == 0 || _recipientDomain == domain) {
      token.safeTransfer(_recipient, _amount);
    } else {
      id = connext.xcall{value: msg.value}(
        _recipientDomain, // destination domain
        _recipient, // to
        address(token), // asset
        _recipient, // delegate, only required for self-execution + slippage
        _amount, // amount
        0, // slippage -- assumes no pools on connext
        bytes('') // calldata
      );
    }
```

```
    emit CrosschainClaim(id, _beneficiary, _recipient, _recipientDomain,
    ↪  _amount);
  }
```

LayneHaber

Agree that it is strictly better to have the fees be an option on the contract itself. We can always pass in `0` and the `xcall` will not fail, which works for our distribution but likely not for others.

Shogoki

Is there a hint to this behaviour in the connext docs to this behaviour?

LayneHaber

docs on the relayer fee behavior can be found here -- this specifically outlines bumping fees, but that implies if the fee is low enough the `xcall` goes through, it's just not processed on the destination chain.

there is no documentation on fee-sponsoring though, and agree that we should make this payable!

LayneHaber

Fixed: https://github.com/SoftDAO/contracts/pull/8

cr-walker

Closing because fix has been merged into repo.

cr-walker

Reopening - I think I'm supposed to leave this open until the fix has been reviewed!

maarcweiss

Fixed by sending the value directly from the `_settleClaim` function.

Issue M-6: Loss of funds during user adjusting

Source: https://github.com/sherlock-audit/2023-06-tokensoft-judging/issues/195

Found by

0xMAKEOUTHILL, Yuki

Summary

Adjusting a user's total claimable value not working correctly

Vulnerability Detail

Whenever the owner is adjusting user's total claimable value, the `records[beneficiary].total` is decreased or increased by `uint256 diff = uint256(amount > 0 ? amount : -amount);`.

However some assumptions made are not correct. **Scenario**:

1. User has bought 200 FOO tokens for example.
2. In `PriceTierVestingSale_2_0.sol` he calls the `initializeDistributionRecord` which sets his `records[beneficiary].total` to the purchased amount || 200. So **records[beneficiary].total = 200**
3. After that the owner decides to adjust his `records[beneficiary].total` to 300. So **records[beneficiary].total = 300**
4. User decides to claim his claimable amount which should be equal to 300. He calls the `claim` function in `PriceTierVestingSale_2_0.sol`.

```
function claim(
    address beneficiary // the address that will receive tokens
  ) external validSaleParticipant(beneficiary) nonReentrant {
    uint256 claimableAmount = getClaimableAmount(beneficiary);
    uint256 purchasedAmount = getPurchasedAmount(beneficiary);

    // effects
    uint256 claimedAmount = super._executeClaim(beneficiary, purchasedAmount);

    // interactions
    super._settleClaim(beneficiary, claimedAmount);
  }
```

As we can see here the `_executeClaim` is called with the `purchasedAmount` of the user which is still 200.

```
function _executeClaim(
    address beneficiary,
    uint256 _totalAmount
  ) internal virtual returns (uint256) {
    uint120 totalAmount = uint120(_totalAmount);

    // effects
    if (records[beneficiary].total != totalAmount) {
      // re-initialize if the total has been updated
      _initializeDistributionRecord(beneficiary, totalAmount);
    }

    uint120 claimableAmount = uint120(getClaimableAmount(beneficiary));
    require(claimableAmount > 0, 'Distributor: no more tokens claimable right
↪   now');

    records[beneficiary].claimed += claimableAmount;
    claimed += claimableAmount;

    return claimableAmount;
  }
```

Now check the `if` statement:

```
if (records[beneficiary].total != totalAmount) {
     // re-initialize if the total has been updated
     _initializeDistributionRecord(beneficiary, totalAmount);
   }
```

The point of this is if the `total` of the user has been adjusted, to re-initialize to the corresponding amount, but since it's updated by the input value which is 200, **records[beneficiary].total = 200** , the user will lose the 100 added from the owner during the `adjust`

Impact

Loss of funds for the user and the protocol

Code Snippet

https://github.com/sherlock-audit/2023-06-tokensoft/blob/main/contracts/contracts/claim/PriceTierVestingSale_2_0.sol#L75-L109

https://github.com/sherlock-audit/2023-06-tokensoft/blob/main/contracts/contracts/claim/abstract/AdvancedDistributor.sol#L105-L131

https://github.com/sherlock-audit/2023-06-tokensoft/blob/main/contracts/contracts/claim/abstract/Distributor.sol#L66-L84

Tool used

Manual Review

Recommendation

I am not sure if it is enough to just set it the following way:

```
if (records[beneficiary].total != totalAmount) {
     // re-initialize if the total has been updated
     `--` _initializeDistributionRecord(beneficiary, totalAmount);
    `++` _initializeDistributionRecord(beneficiary, records[beneficiary].total);
   }
```

Think of different scenarios if it is done that way and also keep in mind that the same holds for the decrease of `records[beneficiary].total` by `adjust`

Discussion

MAKEOUTHILL6

Escalate I think this issue is wrongly excluded as judge thinks it's a type of an admin error. However I believe it should be addressed, because the functionality of this `adjust` function is broken for `PriceTierVestingSale_2_0.sol` regardless of any admin actions taken. The function just doesn't work right and as intended.

sherlock-admin2

> Escalate I think this issue is wrongly excluded as judge thinks it's a type of an admin error. However I believe it should be addressed, because the functionality of this `adjust` function is broken for `PriceTierVestingSale_2_0.sol` regardless of any admin actions taken. The function just doesn't work right and as intended.

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

Shogoki

I do not think the adjust function is intended to be called for the PriceTierVestingSale Distributor, as it does not really make a lot of sense imho. That

is why i think it is an admin error if it is called for this kind of Distributor.

cr-walker

@Shogoki - Alas, I think this is a valid issue because we might want to adjust distributors that refer to sales.

Real world use cases:

- A user participated in a sale but shouldn't have: their distribution record needs to be adjusted downward
- A user participated in a sale and complains a lot about the project: adjust their distribution record upward

hrishibhat

Result: Medium Has duplicates Considering this a valid medium based on the issue and Sponsor comments

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- MAKEOUTHILL6: accepted

cr-walker

Fixed by https://github.com/SoftDAO/contracts/pull/16

maarcweiss

Fixed by accounting for the adjustment of the user. This has been done by getting the `records[beneficiary].total` instead of the `purchasedAmount` which was used before.