SHERLOCK

SHERLOCK SECURITY REVIEW FOR

Prepared for:	Truflation
Prepared by:	Sherlock
Lead Security Expert:	IllIllI
Dates Audited:	January 4 - January 8, 2024
Prepared on:	January 26, 2024

Introduction

The definitive reference point for RWA, Indexes, & Inflation.

Scope

Repository: truflation/truflation-contracts

Branch: main

Commit: ec0862848d9461f6d681e423865aab7ec0027923

For the detailed scope, see the contest details.

Commit hash after fix review

Repository: truflation/truflation-contracts

Branch: main

Commit: c2ba03d5c1ff4d7c974b1e34ef4e3335bb263973

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
3	2

Issues not fixed or acknowledged

Medium	High
0	0

Security experts who found valid issues

zzykxx	jerseyjoewalcott	KupiaSec
s1ce	aslanbek	CL001
zraxx	Kow	ydlee
fnanni	carrotsmuggler	Krace
mstpr-brainbot	0xLogos	Bauer
rvierdiiev	HonorLt	IllIIII
UbiquitousComputing	cawfree	VAD37
nobody2018	IvanFitro	dany.armstrong90
bughuntoor	asauditor	ZdravkoHr.
unforgiven	ubl4nk	0xmystery

Issue H-1: Users can fully drain the `TrufVesting` contract

Source: https://github.com/sherlock-audit/2023-12-truflation-judging/issues/1

Found by

0xLogos, CL001, HonorLt, IvanFitro, KupiaSec, asauditor, bughuntoor, carrotsmuggler, cawfree, fnanni, mstpr-brainbot, s1ce, ubl4nk, unforgiven, ydlee, zzykxx

Summary

Due to flaw in the logic in `claimable` any arbitrary user can drain all the funds within the contract.

Vulnerability Detail

A user's claimable is calculated in the following way:

1. Up until start time it is 0.
2. Between start time and cliff time it's equal to `initialRelease`.
3. After cliff time, it linearly increases until the full period ends.

However, if we look at the code, when we are at stage 2., it always returns `initialRelease`, even if we've already claimed it. This would allow for any arbitrary user to call claim as many times as they wish and every time they'd receive `initialRelease`. Given enough iterations, any user can drain the contract.

```
function claimable(uint256 categoryId, uint256 vestingId, address user)
    public
    view
    returns (uint256 claimableAmount)
{
    UserVesting memory userVesting = userVestings[categoryId][vestingId][user];

    VestingInfo memory info = vestingInfos[categoryId][vestingId];

    uint64 startTime = userVesting.startTime + info.initialReleasePeriod;

    if (startTime > block.timestamp) {
        return 0;
    }

    uint256 totalAmount = userVesting.amount;
    uint256 initialRelease = (totalAmount * info.initialReleasePct) /
↪   DENOMINATOR;

    startTime += info.cliff;

    if (startTime > block.timestamp) {
        return initialRelease;
    }
```

```
function claim(address user, uint256 categoryId, uint256 vestingId, uint256
↪   claimAmount) public {
    if (user != msg.sender && (!categories[categoryId].adminClaimable ||
↪   msg.sender != owner())) {
        revert Forbidden(msg.sender);
    }

    uint256 claimableAmount = claimable(categoryId, vestingId, user);
    if (claimAmount == type(uint256).max) {
        claimAmount = claimableAmount;
    } else if (claimAmount > claimableAmount) {
        revert ClaimAmountExceed();
    }
    if (claimAmount == 0) {
        revert ZeroAmount();
    }

    categories[categoryId].totalClaimed += claimAmount;
    userVestings[categoryId][vestingId][user].claimed += claimAmount;
    trufToken.safeTransfer(user, claimAmount);

    emit Claimed(categoryId, vestingId, user, claimAmount);
}
```

Impact

Any user can drain the contract

Code Snippet

https://github.com/sherlock-audit/2023-12-truflation/blob/main/truflation-contracts/src/token/TrufVesting.sol#L176C1-L182C10

Tool used

Manual Review

Recommendation

change the if check to the following

```
if (startTime > block.timestamp) {
    if (initialRelease > userVesting.claimed) {
    return initialRelease - userVesting.claimed;
    }
    else { return 0; }
}
```

PoC

```
function test_cliffVestingDrain() public {
    _setupVestingPlan();
    uint256 categoryId = 2;
    uint256 vestingId = 0;
    uint256 stakeAmount = 10e18;
    uint256 duration = 30 days;

    vm.startPrank(owner);

    vesting.setUserVesting(categoryId, vestingId, alice, 0, stakeAmount);

    vm.warp(block.timestamp + 11 days);     // warping 11 days, because initial
↪   release period is 10 days
                                            // and cliff is at 20 days. We need
↪   to be in the middle
    vm.startPrank(alice);
    assertEq(trufToken.balanceOf(alice), 0);
    vesting.claim(alice, categoryId, vestingId, type(uint256).max);

    uint256 balance = trufToken.balanceOf(alice);
    assertEq(balance, stakeAmount * 5 / 100);  // Alice should be able to have
↪   claimed just 5% of the vesting

    for (uint i; i < 39; i++ ){
        vesting.claim(alice, categoryId, vestingId, type(uint256).max);
    }
    uint256 newBalance = trufToken.balanceOf(alice);   // Alice has claimed 2x
↪   the amount she was supposed to be vested.
    assertEq(newBalance, stakeAmount * 2);             // In fact she can keep
↪   on doing this to drain the whole contract
}
```

Discussion

sherlock-admin2

1 comment(s) were left on this issue during the judging contest.

Shaheen commented:

> Valid High. Solid Catch

ryuheimat

https://github.com/truflation/truflation-contracts/pull/3

Fixed `claimable` function

mstpr

Fix LGTM

ryuheimat

Merged

Issue H-2: `cancelVesting` will potentially not give users unclaimed, vested funds, even if giveUnclaimed = true

Source: https://github.com/sherlock-audit/2023-12-truflation-judging/issues/192

Found by

s1ce, zraxx, zzykxx

Summary

The purpose of `cancelVesting` is to cancel a vesting grant and potentially give users unclaimed but vested funds in the event that `giveUnclaimed = true`. However, due to a bug, in the event that the user had staked / locked funds, they will potentially not received the unclaimed / vested funds even if `giveUnclaimed = true`.

Vulnerability Detail

Here's the cancelVesting function in TrufVesting:

```
function cancelVesting(uint256 categoryId, uint256 vestingId, address user, bool
 ↪  giveUnclaimed)
        external
        onlyOwner
{
        UserVesting memory userVesting =
 ↪  userVestings[categoryId][vestingId][user];

        if (userVesting.amount == 0) {
            revert UserVestingDoesNotExists(categoryId, vestingId, user);
        }

        if (userVesting.startTime + vestingInfos[categoryId][vestingId].period
 ↪  <= block.timestamp) {
            revert AlreadyVested(categoryId, vestingId, user);
        }

        uint256 lockupId = lockupIds[categoryId][vestingId][user];

        if (lockupId != 0) {
            veTRUF.unstakeVesting(user, lockupId - 1, true);
            delete lockupIds[categoryId][vestingId][user];
            userVesting.locked = 0;
        }
        VestingCategory storage category = categories[categoryId];

        uint256 claimableAmount = claimable(categoryId, vestingId, user);
        if (giveUnclaimed && claimableAmount != 0) {
            trufToken.safeTransfer(user, claimableAmount);

            userVesting.claimed += claimableAmount;
            category.totalClaimed += claimableAmount;
            emit Claimed(categoryId, vestingId, user, claimableAmount);
        }

        uint256 unvested = userVesting.amount - userVesting.claimed;

        delete userVestings[categoryId][vestingId][user];

        category.allocated -= unvested;

        emit CancelVesting(categoryId, vestingId, user, giveUnclaimed);
}
```

First, consider the following code:

```
uint256 lockupId = lockupIds[categoryId][vestingId][user];

if (lockupId != 0) {
            veTRUF.unstakeVesting(user, lockupId - 1, true);
            delete lockupIds[categoryId][vestingId][user];
            userVesting.locked = 0;
}
```

First the locked / staked funds will essentially be un-staked. The following line of code: `userVesting.locked = 0;` exists because there is a call to `uint256 claimableAmount = claimable(categoryId, vestingId, user);` afterwards, and in the event that there were locked funds that were unstaked, these funds should now potentially be claimable if they are vested (but if locked is not set to 0, then the vested funds will potentially not be deemed claimable by the `claimable` function).

However, because `userVesting` is `memory` rather than `storage`, this doesn't end up happening (so `userVesting.locked = 0;` is actually a bug). This means that if a user is currently staking all their funds (so all their funds are locked), and `cancelVesting` is called, then they will not receive any funds back even if `giveUnclaimed = true`. This is because the `claimable` function (which will access the unaltered `userVestings[categoryId][vestingId][user]`) will still think that all the funds are currently locked, even though they are not as they have been forcibly unstaked.

Impact

When `cancelVesting` is called, a user may not receive their unclaimed, vested funds.

Code Snippet

https://github.com/sherlock-audit/2023-12-truflation/blob/main/truflation-contracts/src/token/TrufVesting.sol#L348-L388

Tool used

Manual Review

Recommendation

Change `userVesting.locked = 0;` to
`userVestings[categoryId][vestingId][user].locked = 0;`

Discussion

ryuheimat

Fixed. Updated `userVesting` type to storage type to fix
https://github.com/sherlock-audit/2023-12-truflation-judging/issues/192
Considered initial release period and cliff for vesting end validation to fix
https://github.com/sherlock-audit/2023-12-truflation-judging/issues/11

Extra: Added `admin` permission, and allow several admins to be able to set vesting info and user vestings. https://github.com/truflation/truflation-contracts/pull/4

mstpr

Fix and new admin functionality LGTM

ryuheimat

> Fix and new admin functionality LGTM

Thank you, merging PR.

Issue M-1: TrufVesting.cancelVesting calculates end of vesting incorrectly

Source: https://github.com/sherlock-audit/2023-12-truflation-judging/issues/11

Found by

UbiquitousComputing, fnanni, nobody2018, rvierdiiev

Summary

TrufVesting.cancelVesting calculates end of vesting incorrectly and because of that owner can't stop vesting for the account.

Vulnerability Detail

`TrufVesting.cancelVesting` function allows owner to stop vesting for some account.

In case if vesting already finished, then function reverts.

The problem is that `vestingInfos[categoryId][vestingId].period` is just period for token distribution after cliff.

As you can see in `claimable` function the vesting starts on `startTime`, then when initialReleasePeriod has passed, then `initialRelease` is distributed. Then after cliff has passed, only then final distribution starts.

Thus check if vesting already finished is incorrect in the `TrufVesting.cancelVesting` and it doesn't allow owner to cancel vesting when it is still going.

Impact

Ongoing vesting can't be canceled.

Code Snippet

Provided above

Tool used

Manual Review

Recommendation

This check should be correct.

```
if (userVesting.startTime +
↪   vestingInfos[categoryId][vestingId].initialReleasePeriod +
↪   vestingInfos[categoryId][vestingId].cliff +
↪   vestingInfos[categoryId][vestingId].period <= block.timestamp) {
            revert AlreadyVested(categoryId, vestingId, user);
}
```

Discussion

ryuheimat

https://github.com/truflation/truflation-contracts/pull/4

mstpr

Nice catch by the Watson! Fix LGTM

ryuheimat

Merged

Issue M-2: When migrating the owner users will lose their rewards

Source: https://github.com/sherlock-audit/2023-12-truflation-judging/issues/80

Found by

0xmystery, Bauer, IIIIIII, Krace, VAD37, ZdravkoHr., dany.armstrong90, mstpr-brainbot, s1ce

Summary

When a user migrates the owner due to a lost private key, the rewards belonging to the previous owner remain recorded in their account and cannot be claimed, resulting in the loss of user rewards.

Vulnerability Detail

According to the documentation, `migrateUser()` is used when a user loses their private key to migrate the old vesting owner to a new owner.

```
/**
 * @notice Migrate owner of vesting. Used when user lost his private key
 * @dev Only admin can migrate users vesting
 * @param categoryId Category id
 * @param vestingId Vesting id
 * @param prevUser previous user address
 * @param newUser new user address
 */
```

In this function, the protocol calls `migrateVestingLock()` to obtain a new ID.

```
if (lockupId != 0) {
        newLockupId = veTRUF.migrateVestingLock(prevUser, newUser, lockupId - 1)
 ↪  + 1;
        lockupIds[categoryId][vestingId][newUser] = newLockupId;
        delete lockupIds[categoryId][vestingId][prevUser];

        newVesting.locked = prevVesting.locked;
    }
```

However, in the `migrateVestingLock()` function, the protocol calls `stakingRewards.withdraw()` to withdraw the user's stake, burning points. In the `withdraw()` function, the protocol first calls `updateReward()` to update the user's rewards and records them in the user's account.

```
function withdraw(address user, uint256 amount) public updateReward(user)
↪   onlyOperator {
    if (amount == 0) {
        revert ZeroAmount();
    }
    _totalSupply -= amount;
    _balances[user] -= amount;
    emit Withdrawn(user, amount);
}
```

However, `stakingRewards.withdraw()` is called with the old owner as a parameter, meaning that the rewards will be updated on the old account.

```
uint256 points = oldLockup.points;
      stakingRewards.withdraw(oldUser, points);
      _burn(oldUser, points);
```

As mentioned earlier, the old owner has lost their private key and cannot claim the rewards, resulting in the loss of these rewards.

Impact

The user's rewards are lost

Code Snippet

https://github.com/sherlock-audit/2023-12-truflation/blob/main/truflation-contracts/src/token/TrufVesting.sol#L329

Tool used

Manual Review

Recommendation

When migrating the owner, the rewards belonging to the previous owner should be transferred to the new owner.

Discussion

sherlock-admin2

1 comment(s) were left on this issue during the judging contest.

0xLogos commented:

> loss of private key is extreme case, so i think better do not add additional complexity to the code

ryuheimat

https://github.com/truflation/truflation-contracts/pull/10

Added `to` param to `getReward` function, and call it when migrating.

nevillehuang

@securitygrid I considered the same exact point during judging, but considering the protocol has an explicit function just to migrate users for private keys, this constitute at the very least medium for me, because it is intended protocol functionality to fully migrate users position.

If not, why not say the vested position is also users responsibility too, so I think your argument is moot.

securitygrid

Escalate Loss of the private key should be an uncommon scenario. Therefore the problem should be M.

sherlock-admin2

> Escalate Loss of the private key should be an uncommon scenario. Therefore the problem should be M.

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.

nevillehuang

Since this directly contradicts the intended functionality of the function of transferring ALL of the users intended vested position and rewards, which means likely a definite loss of funds for them, I think this should remain high severity.

Czar102

This is a backup mechanism, and users won't intentionally lose their private keys because the funds can be retrieved.

I'd classify it as a loss of intended functionality and not a loss of funds (because the funds can still be used by the pre-migration address owner), hence planning to consider this a medium severity issue.

nevillehuang

@Czar102 I don't understand your following statement, if they lose their private keys how can they use the funds? Whatever it is I will take it into account the special case of losing private keys in the future.

> because the funds can still be used by the pre-migration address owner

Czar102

I understand the confusion. The past owner remains the owner of the funds. I detached the address identity (with whoever may still own it, even if it's no one) from the new owner.

sleepriverfish

@Czar102 Both users who lost their private keys and the new owner are unable to use these funds.

Czar102

@sleepriverfish I'm not following. From my understanding, the report states that the rewards are sent to the old address.

nevillehuang

@Czar102 For your info, I still don't get what your are pointing to, but I accept the severity downgrade only because of my own understanding that private key loss is an exceptional case

Czar102

Result: Medium Has duplicates

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- securitygrid: accepted

mstpr

Fix LGTM

ryuheimat

Merged

Issue M-3: Ended locks can be extended

Source: https://github.com/sherlock-audit/2023-12-truflation-judging/issues/82

Found by

Kow, aslanbek, bughuntoor, jerseyjoewalcott, mstpr-brainbot, unforgiven, zzykxx

Summary

When a lock period ends, it can be extended. If the new extension 'end' is earlier than the current block.timestamp, the user will have a lock that can be unstaked at any time."

Vulnerability Detail

When the lock period ends, the owner of the expired lock can extend it to set a new lock end that is earlier than the current block.timestamp. By doing so, the lock owner can create a lock that is unstakeable at any time.

This is doable because there are no checks in the extendLock function that checks whether the lock is already ended or not.

PoC:

```
function test_ExtendLock_AlreadyEnded() external {
        uint256 amount = 100e18;
        uint256 duration = 5 days;

        _stake(amount, duration, alice, alice);

        // 5 days later, lock is ended for Alice
        skip(5 days + 1);

        (,, uint128 _ends,,) = veTRUF.lockups(alice, 0);

        // Alice's lock is indeed ended
        assertTrue(_ends < block.timestamp, "lock is ended");

        // 36 days passed
        skip(36 days);

        // Alice extends her already finished lock 30 more days
        vm.prank(alice);
        veTRUF.extendLock(0, 30 days);
        (,,_ends,,) = veTRUF.lockups(alice, 0);

        // Alice's lock can be easily unlocked right away
        assertTrue(_ends < block.timestamp, "lock is ended");

        // Alice unstakes her lock, basically alice can unstake her lock anytime
↪   she likes
        vm.prank(alice);
        veTRUF.unstake(0);
    }
```

Impact

The owner of the lock will achieve points that he can unlock anytime. This is clearly a gaming of the system and shouldn't be acceptable behaviour. A locker having a "lock" that can be unstaked anytime will be unfair for the other lockers. Considering this, I'll label this as high.

Code Snippet

https://github.com/sherlock-audit/2023-12-truflation/blob/37ddbb69e0c7fb6510f1ec99162fd9172ec44733/truflation-contracts/src/token/VotingEscrowTruf.sol#L333-L366

Tool used

Manual Review

Recommendation

Do not let extension of locks that are already ended.

Discussion

ryuheimat

https://github.com/truflation/truflation-contracts/pull/7 Disable to extend lock if already expired.

sherlock-admin2

1 comment(s) were left on this issue during the judging contest.

Shaheen commented:

> Valid issue but Medium. Good Finding

securitygrid

Escalate I didn't sumbit this issue because I think this is not an issue: no funds loss, no core function broken. The reason is: Alice is still in the game and has not called `unstake` to exit the game.

nevillehuang

@ryuheimat I think @securitygrid has a point, and this seems similar to this issue here, might have slipped my judgement

https://github.com/sherlock-audit/2023-12-truflation-judging/issues/169

However both this and #169 indeed does break the "locking" mechanism, given it is expected that funds should be locked within escrows instead of allowing users to simply unlock immediately. I believe both should be at least medium/high severity issues, and could possibly be considered to be duplicates

sherlock-admin2

> Escalate I didn't sumbit this issue because I think this is not an issue: no funds loss, no core function broken. The reason is: Alice is still in the game and has not called `unstake` to exit the game.

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.

detectiveking123

This is definitely a valid issue, but it is unclear to me whether it should be labeled as medium or high.

Czar102

I believe this is a valid medium. Core functionality – rewarding points for locking tokens – is broken.

From my understanding, someone must have locked the tokens in the past and can receive points retroactively, meaning that they effectively can achieve flexible duration locks. This shouldn't be possible and is an important property of the system.

Czar102

Also, I don't think #169 is a duplicate. Can you explain in more detail why would that be the case? @nevillehuang

nevillehuang

@Czar102 because they have the same impact, that is there is no “lock” required. I.e. User can gain rewards and retain voting power while still being able to unstake at any time.

Czar102

I see now. So effectively, the extension of the lock mentioned here doesn't matter since the identity locking receives points anyway?

nevillehuang

@Czar102 yes as long as the lock expires, the user will have no incentive to continue locking, and even if they do, they can unlock the lock at anytime as mentioned here. So I believe they should be duplicated as valid Medium severity.

Czar102

Planning to duplicate these and make the severity Medium.

aslanbekaibimov

@Czar102

> I see now. So effectively, the extension of the lock mentioned here doesn't matter since the identity locking receives points anyway?

It does matter. #82 describes how the user is able to have voting power **greater** than their original one, while always being able to withdraw. The fix is straightforward - to add a check for the lock's expiration.

On the other hand, #169 only allows users to **keep** their original voting power from the original lock. The fix of #82 also does not fix #169.

0xunforgiven

I don't believe they are duplicates. the only common thing between #82 and #169 is a precondition for the issues. they both happens when a lock expires. but impact and fix and bug place is totally different for them.

#82 explains that after lock expires users can call extend lock and extend their lock. so the issue is in extend function and it can be fixed by simple check in extend function, but this fix won't fix the #169.

#169 explains that after the lock expires users still have voting power and receive rewards. so the issue is in voting mechanism and reward distribution. this issue can't be fixed easily with current design. fixing this will not fix the #82.

multiple wardens reported these two as separate issues while some wardens missed one of them. issues can't be concluded from each other.

mstpr

Fix LGTM

ryuheimat

> mstpr

Merging...

Czar102

Removing locks after they expire will both remove the voting power (locks don't exist anymore) and make it impossible to retroactively extend the lock (since any ended lock won't exist anymore). This is why I think these are duplicates.

0xunforgiven

@Czar102 regarding:

> Removing locks after they expire

in current code only the stacker can remove their own lock. so why should they remove their expired lock? this was their attack to not remove the expired lock and receive the rewards and voting power. he wouldn't remove his own lock(he is the one performing the attack) and as result receive rewards for his expired lock.

the sponsor in the https://github.com/sherlock-audit/2023-12-truflation-judging/issues/82#issuecomment-1882784243 says that their fix for extending expire lock is this:

> Disable to extend lock if already expired.

so issue #169 still exists in the code.

Czar102

I agree it still exists. I still think these issues are duplicates, though instead of fixing the root cause #169, the sponsor decided to mitigate the implication of #169 described here, in #82.

Please note that the sponsor acknowledged an unfixed duplicate of this issue #169, which has a different impact.

Czar102

Result: Medium Has duplicates

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- securitygrid: accepted

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.