BAILSEC.IO

OFFICE@BAILSEC.IO

X: @BAILSECURITY

TG: @HELLOATBAILSEC

FINAL REPORT

Trustswap Incident

Post Mortem

October 2024

Scenario Description

A deployment issue in TrustSwap's StakingPool smart contract implementation was exploited, resulting in an unauthorized reward amplification. The incident specifically targeted the contract's initialization state, allowing the attacker to manipulate the reward distribution mechanism to receive double the intended reward amount for a specific staking position. The theft was possible due to the incorrect initialization of the smart contract.

Impact:

- Type: One-time reward theft
- Vector: Contract initialization vulnerability
- Result: 2x reward multiplication
- Affected Component: StakingPool smart contract

Mitigation:

Trustswap will purchase the stolen tokens on the market and re-supply them via simple transfer to the corresponding Stakingpool contracts.

Timeline of Actions

On september 30 around 2pm UTC, Bailsec was informed about attacks which have been executed on Trustswap's StakingPool implementation by Decurity. Shortly afterwards a war room was opened by SEAL 911 with all relevant parties. After a quick escalation phase, Bailsec has swiftly identified the issue of uninitialized contracts and prompted the Trustswap team to initialize the contracts, which has been done. Around 4 hours later, another implementation was exploited which was due to the fact that one transaction from Trustswap's development team did not go through. Subsequently, further implementations were initialized without any further damage.

Technical Background

The StakingPool contract from TrustSwap has been exploited for a one-time reward theft due to the fact that **the implementation has not been initialized,** which has allowed the attacker to disable the reentrancy guard. This exact issue has been described in the audit report two times:

First: Lack of support for transfer-tax tokens results in multiple ground-breaking issues

Recommendations	Consider simply following the before-after pattern. A reentrancy guard is mandatory.
Comments / Resolution	Resolved, the before/after pattern has been implemented for the following spots: addPool deposit It is important to call initializePoolV2 directly after initialization to ensure that no one can reenter in this function to disable the reentrancy guard to then reenter in another function which is now unguarded.

Second: Arbitrary staking and reward tokens may allow for reentrancy occasions

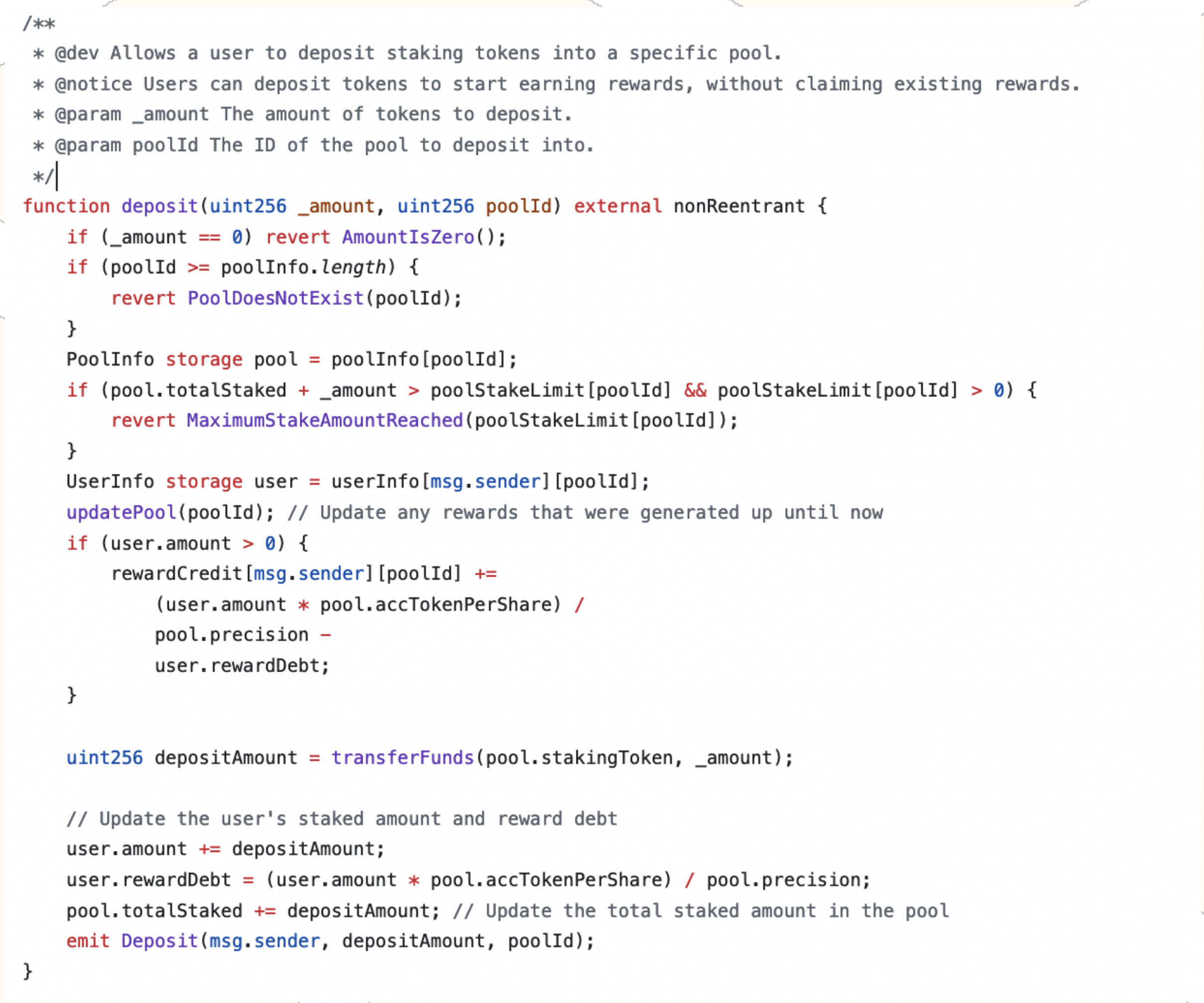
Recommendations	Consider implementing a reentrancy guard on every external function.
Comments / Resolution	Resolved, a reentrancyGuard has been implemented on: addPool stopReward deposit withdraw claimReward withdrawEmptyPool It is important to call initializePoolV2 directly after deployment to ensure that no one can reenter in this

Unfortunately, due to recent transitions in the development team, the new TrustSwap engineers missed the opportunity to review the audit report before the deployment of the contracts.

The smart contract itself is (considered it is initialized correctly) bug-free. The reason why a reentrancy guard was mandatory instead of the simple CEI pattern is due to the fact that a **before-after pattern in the token transfer** was incorporated in an effort to support transfer-tax tokens (see first issue).

Root-cause explained:

The deposit function incorporates a pattern which casts rewards that have not yet paid out within the deposit function (this was even implemented after the initial audit but **our initial audit still covers the possibility of such scenarios in the recommendations**):



```
/**
 * @dev Allows a user to deposit staking tokens into a specific pool.
 * @notice Users can deposit tokens to start earning rewards, without claiming existing rewards.
 * @param _amount The amount of tokens to deposit.
 * @param poolId The ID of the pool to deposit into.
 */
function deposit(uint256 _amount, uint256 poolId) external nonReentrant {
    if (_amount == 0) revert AmountIsZero();
    if (poolId >= poolInfo.length) {
        revert PoolDoesNotExist(poolId);
    }
    PoolInfo storage pool = poolInfo[poolId];
    if (pool.totalStaked + _amount > poolStakeLimit[poolId] && poolStakeLimit[poolId] > 0) {
        revert MaximumStakeAmountReached(poolStakeLimit[poolId]);
    }
    UserInfo storage user = userInfo[msg.sender][poolId];
    updatePool(poolId); // Update any rewards that were generated up until now
    if (user.amount > 0) {
        rewardCredit[msg.sender][poolId] +=
            (user.amount * pool.accTokenPerShare) /
            pool.precision -
            user.rewardDebt;
    }

    uint256 depositAmount = transferFunds(pool.stakingToken, _amount);

    // Update the user's staked amount and reward debt
    user.amount += depositAmount;
    user.rewardDebt = (user.amount * pool.accTokenPerShare) / pool.precision;
    pool.totalStaked += depositAmount; // Update the total staked amount in the pool
    emit Deposit(msg.sender, depositAmount, poolId);
}
```

As one can see, in the scenario where user.amount > 0, it will add rewards to the rewardCredit mapping. Afterwards, funds are transferred in, using the **before-after pattern** to support transfer-tax tokens. This practice **inherently violates the CEI pattern**: https://bailsec.io/tpost/gxcih1xoy1-checks-effects-interactions, which was also multiple times reported in the audit.

This is a standard practice to support transfer-tax tokens and with this practice it becomes mandatory to include a reentrancy guard because the function **inherently violates the CEI pattern**, again we will refer to the audit document:

Recommendations	Consider simply following the before-after pattern. A reentrancy guard is mandatory.

This means, without a reentrancy guard one could reenter upon the transfer while the **rewardDebt** is not updated and the contract is in a faulty state.

After we have elaborated the root cause of the issue, we will explain the flow how to double claim rewards:

1. Create a pool with a stakingToken being a dummy reentrancy token and a valid rewardToken while providing a specific amount of reward tokens which is paid out for staking this token.

2. Deposit the token and accrue rewards

3. Wait some time until rewards are accrued

4. Trigger again the deposit function which increases the rewardCredit mapping. Upon token transfer, re enter into the initializePoolV2 function (which should be been invoked directly after deployment), this will reset the locked state of the contract.

5. Reenter into the claimReward function which now **again** calculates the rewards which have been already added to the rewardCredit mapping. This is working because the rewardDebt has not yet been updated.

6. Pay out 2x of the reward token amount which has been initially deposited

The fact that the exploiter lost a large part of the exploited funds due to a sandwich attack in the swap because of an invalid minAmountOut parameter indicates that the exploiter was not a sophisticated hacker. **He was just capable of reading our audit report and exploited the fact that unfortunately our recommendations were not implemented.**

The audit report can be found here and was published 9 months ago, clearly highlighting this issue **two times**.

Audit Report:

https://github.com/bailsec/BailSec/blob/main/Bailsec%20-%20Trustswap%20StakingPool%20Audit(%2BResolution)%2BExtension.pdf

The lack of initialization was due to the fact that Trustswap went through internal changes and hired a new engineer who was responsible for deploying the contract 6 months after the audit was published.

Unfortunately, the fact that this contract must be initialized via initializePoolV2 was not internally communicated with the new developer by the previous development team.

Relevant Transactions

https://bscscan.com/tx/0x94cd8b88969bcf1ca03bb6db0df465e46cfa8dc8dded25e0497c34
73e888f573

https://bscscan.com/tx/0x88d228aa4a78895d852fbd11f8f66dcdcab4dca15d7f7f0d8a02693fc
018a976

https://bscscan.com/tx/0x5fb704be48965fa0ccb5793fda2dd43a3c880ca8985da65d3623d1
094b2c9649

https://bscscan.com/tx/0x064b04822f069bf7b78c250a352468487d6cd71018d7465add8561
02b42cbade

https://etherscan.io/tx/0x83952d998cc562f40d0a58b76d563a16f3064ddb116e7b1b4e40298
ca80499b8

https://basescan.org/tx/0x5856c934ffebe9dbb74b9e5732f9f6c7ce29928b80f5318f96d73eaa
6435b685

Prevention Techniques

TrustSwap has established a comprehensive security partnership with BailSec through a 12-month retainer agreement. This strategic collaboration represents a significant enhancement to their security infrastructure.

Scope of Services

The retainer agreement with BailSec includes, but is not limited to:

- Deployment oversight and verification
- Pre-deployment security reviews
- On-demand security consultations
- Emergency response support

Commitment Duration

- Initial Term: Minimum of 12 months
- Service Model: Monthly retainer basis
- Availability: Continuous security coverage

This partnership demonstrates TrustSwap's ongoing commitment to maintaining the highest standards of security for their protocol and community.