Pangolin Staking Positions Audit Report

Aug 30, 2022

Table of Contents

Summary

This report has been prepared for Pangolin Staking Positions Audit Report smart contract, to discover issues and vulnerabilities in the source code of their Smart Contract as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Static Analysis and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

Overview

Project Summary

Project Name	**Pangolin Staking Positions**
Codebase	**https://github.com/pangolindex/exchange-contracts**
Commit	**7c01b8f562e2095a36a3689cd1e587ef630b242c**
Language	**Solidity**

Audit Summary

Delivery Date	**Aug 30, 2022**
Audit Methodology	**Static Analysis, Manual Review**
Total Isssues	**5**

WP-H1: Add liquidity to the PNG-WAVAX pool in compoundToPoolZero() or compound() with msg.value is not handled properly

High

Issue Description

https://github.com/pangolindex/exchange-contracts/blob/6739cd2ed98903c39f8173eff7565c56e4d97456/contracts/staking-positions/PangoChef.sol#L641-L687

```
function _addLiquidity(
    Pool storage pool,
    uint256 rewardAmount,
    uint256 maxPairAmount
) private returns (uint256 poolTokenAmount) {
    address poolToken = pool.tokenOrRecipient;
    address rewardPair = pool.rewardPair;

    // Get token amounts from the pool.
    (uint256 reserve0, uint256 reserve1, ) =
IPangolinPair(poolToken).getReserves();

    // Get the reward token's pair's amount from the reserves.
    ERC20 tmpRewardsToken = rewardsToken;
    uint256 pairAmount = address(tmpRewardsToken) < rewardPair
        ? (reserve1 * rewardAmount) / reserve0
        : (reserve0 * rewardAmount) / reserve1;

    // Ensure slippage is not above the limit.
    if (pairAmount > maxPairAmount) revert HighSlippage();

    // Non-zero message value signals desire to pay with native token.
    if (msg.value > 0) {
        // Ensure reward pair is native token.
        if (rewardPair != wrappedNativeToken) revert InvalidToken();

        // Ensure consistent slippage control.
        if (msg.value != maxPairAmount) revert InvalidAmount();
668
669             // Wrap the native token.
670             IWAVAX(rewardPair).deposit{ value: pairAmount }();
671
672             // Refund user.
673             unchecked {
674                 uint256 refundAmount = msg.value - pairAmount;
675                 if (refundAmount != 0) SafeTransferLib.safeTransferETH(msg.sender,
refundAmount);
676             }
677         }
678
679         // Transfer reward pair tokens from the user to the pair contract.
680         ERC20(rewardPair).safeTransferFrom(msg.sender, poolToken, pairAmount);
681
682         // Transfer reward tokens from the contract to the pair contract.
683         tmpRewardsToken.safeTransfer(poolToken, rewardAmount);
684
685         // Mint liquidity tokens to the PangoChef and return the amount minted.
686         poolTokenAmount = IPangolinPair(poolToken).mint(address(this));
687     }
```

When the user `compoundToPoolZero()` or `compound()` , a certain amount of native tokens can be sent as `msg.value` to be paired with the `rewardsToken` (`PNG`) and add liquidity to the `PNG-WAVAX` pool.

The user can also choose not to pay with the native token, but with `WAVAX` . The contract will then pull the funds from the user directly to the `PNG-WAVAX` pool (L680).

However, the current implementation is not handling the native token payment (`msg.value`) correctly: the native tokens will be wrapped (L670), but the wrapped native tokens will not be sent to the `poolToken` contract.

Instead, it will continue to pull funds from the user's wallet to the `PNG-WAVAX` pool (L680).

As a result, the users who pay with `msg.value` will be paying double the amount of native tokens.

Recommendation

Change to:

```
function _addLiquidity(
    Pool storage pool,
    uint256 rewardAmount,
    uint256 maxPairAmount
) private returns (uint256 poolTokenAmount) {
    address poolToken = pool.tokenOrRecipient;
    address rewardPair = pool.rewardPair;

    // Get token amounts from the pool.
    (uint256 reserve0, uint256 reserve1, ) =
IPangolinPair(poolToken).getReserves();

    // Get the reward token's pair's amount from the reserves.
    ERC20 tmpRewardsToken = rewardsToken;
    uint256 pairAmount = address(tmpRewardsToken) < rewardPair
        ? (reserve1 * rewardAmount) / reserve0
        : (reserve0 * rewardAmount) / reserve1;

    // Ensure slippage is not above the limit.
    if (pairAmount > maxPairAmount) revert HighSlippage();

    // Non-zero message value signals desire to pay with native token.
    if (msg.value > 0) {
        // Ensure reward pair is native token.
        if (rewardPair != wrappedNativeToken) revert InvalidToken();

        // Ensure consistent slippage control.
        if (msg.value != maxPairAmount) revert InvalidAmount();

        // Wrap the native token.
        IWAVAX(rewardPair).deposit{ value: pairAmount }();
        // Transfer reward pair tokens from this contract to the pair contract.
        ERC20(rewardPair).safeTransfer(poolToken, pairAmount);

        // Refund user.
        unchecked {
            uint256 refundAmount = msg.value - pairAmount;
            if (refundAmount != 0) SafeTransferLib.safeTransferETH(msg.sender,
refundAmount);
```

```
        }
    } else {
        // Transfer reward pair tokens from the user to the pair contract.
        ERC20(rewardPair).safeTransferFrom(msg.sender, poolToken, pairAmount);
    }

    // Transfer reward tokens from the contract to the pair contract.
    tmpRewardsToken.safeTransfer(poolToken, rewardAmount);

    // Mint liquidity tokens to the PangoChef and return the amount minted.
    poolTokenAmount = IPangolinPair(poolToken).mint(address(this));
}
```

Status

✓ Fixed

WP-M2: Insufficient slippage control in _addLiquidity() with maxPairAmount

Medium

Issue Description

When the user `compoundToPoolZero()` or `compound()` , a certain amount of other tokens (native or ERC20 tokens) can be paired with the `rewardsToken` (`PNG`) and add liquidity to the `PNG` -paired pool.

According to the comments, the `maxPairAmount` parameter is used for `slippage control` .

While it does put a upper limit to the amount of paired tokens to be spent for adding liquidity, which is useful to some extend, to prevent overspending, but we believe it's insufficient to prevent MEV.

https://github.com/pangolindex/exchange-contracts/blob/6739cd2ed98903c39f8173eff7565c56e4d97456/contracts/staking-positions/PangoChef.sol#L641-L687

```
641  function _addLiquidity(
642      Pool storage pool,
643      uint256 rewardAmount,
644      uint256 maxPairAmount
645  ) private returns (uint256 poolTokenAmount) {
646      address poolToken = pool.tokenOrRecipient;
647      address rewardPair = pool.rewardPair;
648
649      // Get token amounts from the pool.
650      (uint256 reserve0, uint256 reserve1, ) =
     IPangolinPair(poolToken).getReserves();
651
652      // Get the reward token's pair's amount from the reserves.
653      ERC20 tmpRewardsToken = rewardsToken;
654      uint256 pairAmount = address(tmpRewardsToken) < rewardPair
655          ? (reserve1 * rewardAmount) / reserve0
656          : (reserve0 * rewardAmount) / reserve1;
657
658      // Ensure slippage is not above the limit.
```

```
    if (pairAmount > maxPairAmount) revert HighSlippage();

    // Non-zero message value signals desire to pay with native token.
    if (msg.value > 0) {
        // Ensure reward pair is native token.
        if (rewardPair != wrappedNativeToken) revert InvalidToken();

        // Ensure consistent slippage control.
        if (msg.value != maxPairAmount) revert InvalidAmount();

        // Wrap the native token.
        IWAVAX(rewardPair).deposit{ value: pairAmount }();

        // Refund user.
        unchecked {
            uint256 refundAmount = msg.value - pairAmount;
            if (refundAmount != 0) SafeTransferLib.safeTransferETH(msg.sender,
refundAmount);
        }
    }

    // Transfer reward pair tokens from the user to the pair contract.
    ERC20(rewardPair).safeTransferFrom(msg.sender, poolToken, pairAmount);

    // Transfer reward tokens from the contract to the pair contract.
    tmpRewardsToken.safeTransfer(poolToken, rewardAmount);

    // Mint liquidity tokens to the PangoChef and return the amount minted.
    poolTokenAmount = IPangolinPair(poolToken).mint(address(this));
}
```

PoC

Given:

- `poolToken` : `PNG-USDC` ;
- Current market price of `PNG` is: `10 USDC` ;
- Current `PNG-USDC` pool reserves: `10 PNG` and `100 USDC` . (We use a lower liquidity as an example for easier manipulation.The attack vector works for pools with regular liquidity as well.)

1. Alice called `stake()` to compounding `PNG-USDC` with `10k PNG` and `100k USDC` as `maxPairAmount` ;
2. Attacker frontrunned Alice's `stake()` transaction and skewed the price of `PNG` in `PNG-USDC` pool to `1 USDC` ;
3. By the time Alice's `stake()` was minted, only `10k USDC` was added (at `1 USDC per PNG`);
4. Attacker backrunned Alice's `stake()` transaction and bought most of the newly added `PNG` tokens at a huge discount.

Recommendation

Consider adding a new parameter `minPairAmount` for slippage control.

WP-N3: Misleading varibale names

Issue Description

1. “user” sounds like the user’s address, while it’s actually a struct that storing the user’s information in the pool:

https://github.com/pangolindex/exchange-contracts/blob/6739cd2ed98903c39f8173eff7565c56e4d97456/contracts/staking-positions/PangoChef.sol#L586-L628

```
function _harvestWithoutReset(uint256 poolId) private returns (uint256 reward) {
    // Create a storage pointer for the pool and the user.
    Pool storage pool = pools[poolId];
    User storage user = pool.users[msg.sender];

    // Ensure pool is ERC20 type.
    _onlyERC20Pool(pool);

    // Update pool summations that govern the reward distribution from pool to
users.
    _updateRewardSummations(poolId, pool);

    // Pool zero should instead use `compound()`.
    if (poolId == 0) revert InvalidType();

    // Increment lock count on pool zero if this pool was not already locking it.
    _incrementLockOnPoolZero(user);

    // Get the rewards accrued by the user, then delete the user stash.
    reward = _userPendingRewards(poolId, pool, user);
    user.stashedRewards = 0;

    // Ensure there are sufficient rewards to use in compounding.
    if (reward == 0) revert NoEffect();

    // Increment the previousValues to not reset the staking duration. In the
proofs,
    // previousValues was regarding combining positions, however we are not
combining positions
    // here. Consider this trick as combining with a null position. This allows us
to not reset
```

```
    // the staking duration but exclude any rewards before block time.
    uint256 userBalance = user.valueVariables.balance;
    user.previousValues += uint152(userBalance * (block.timestamp -
user.lastUpdate));

    // Snapshot the lastUpdate and summations.
    _snapshotRewardSummations(pool, user);

    // Emit the harvest event, even though it will not be transferred to the user.
    emit Withdrawn(poolId, msg.sender, 0, reward);

    // Get extra rewards from rewarder.
    IRewarder rewarder = pool.rewarder;
    if (address(rewarder) != address(0)) {
        rewarder.onReward(poolId, msg.sender, msg.sender, reward, userBalance);
    }
}
```

Recommendation

Consider renaming to `poolUser` .

2. “pool.rewardPair” sounds like the address of the pair, while it’s actually the address of the token that paired the ”rewardToken”

https://github.com/pangolindex/exchange-contracts/blob/6739cd2ed98903c39f8173eff7565c56e4d97456/contracts/staking-positions/PangoChef.sol#L88-L105

```
struct Pool {
    // The address of the token when poolType is ERC_20, or the recipient address
when poolType
    // is RELAYER_POOL.
    address tokenOrRecipient;
    // The type of the pool, which determines which actions can be performed on
it.
    PoolType poolType;
    // An external contract that distributes additional rewards.
    IRewarder rewarder;
    // The address that is paired with PNG. It is zero address if the pool token
is not a
```

```
    // liquidity pool token, or if the liquidity pool do not have PNG as one of the reserves.
    address rewardPair;
    // Two variables that specify the total shares (i.e.: “value”) in the pool.
    ValueVariables valueVariables;
    // Summations incremented on every action on the pool.
    RewardSummations rewardSummationsStored;
    // The mapping from addresses of the users of the pool to their properties.
    mapping(address => User) users;
}
```

https://github.com/pangolindex/exchange-contracts/blob/6739cd2ed98903c39f8173eff7565c56e4d97456/contracts/staking-positions/PangoChef.sol#L641-L647

```
    function _addLiquidity(
        Pool storage pool,
        uint256 rewardAmount,
        uint256 maxPairAmount
    ) private returns (uint256 poolTokenAmount) {
        address poolToken = pool.tokenOrRecipient;
        address rewardPair = pool.rewardPair;
```

Recommendation

Consider renaming to `rewardPairedToken` .

Status

ⓘ Acknowledged

WP-G4: Existence of pool.rewarder should be checked before the IRewarder.onReward() call for consistency and gas saving

Gas

Issue Description

The rewarder may not exists and other methods like _stake() (L503), _withdraw() (L574), _harvestWithoutReset() (L625) will check the existence of rewarder before calling IRewarder.onReward().

https://github.com/pangolindex/exchange-contracts/blob/6739cd2ed98903c39f8173eff7565c56e4d97456/contracts/staking-positions/PangoChef.sol#L513-L578

```
513  function _withdraw(uint256 poolId, uint256 amount) private {
514      // Create a storage pointer for the pool and the user.
515      Pool storage pool = pools[poolId];
516      User storage user = pool.users[msg.sender];
517
518      // Ensure pool is ERC20 type.
519      _onlyERC20Pool(pool);
520
521      // Update pool summations that govern the reward distribution from pool to users.
522      _updateRewardSummations(poolId, pool);
523
524      // Ensure pool zero is not locked.
525      // Decrement lock count on pool zero if this pool was locking it.
526      _decrementLockOnPoolZero(poolId, user);
527
528      // Get position balance and ensure sufficient balance exists.
529      ValueVariables storage userValueVariables = user.valueVariables;
530      uint256 oldBalance = userValueVariables.balance;
531      if (amount > oldBalance) revert InsufficientBalance();
532
533      // Before everything else, get the rewards accrued by the user, then delete the user stash.
534      uint256 reward = _userPendingRewards(poolId, pool, user);
535      user.stashedRewards = 0;
```

```

    // Ensure we are either withdrawing something or claiming rewards.
    if (amount == 0 && reward == 0) revert NoEffect();

    uint256 remaining;
    unchecked {
        // Get the remaining balance in the position.
        remaining = oldBalance - amount;

        // Decrement the withdrawn amount from totalStaked.
        ValueVariables storage poolValueVariables = pool.valueVariables;
        poolValueVariables.balance -= uint104(amount);

        // Update sumOfEntryTimes.
        uint256 newEntryTimes = block.timestamp * remaining;
        poolValueVariables.sumOfEntryTimes = uint152(
            poolValueVariables.sumOfEntryTimes +
                newEntryTimes -
                userValueVariables.sumOfEntryTimes
        );

        // Decrement the withdrawn amount from user balance, and update the user
entry times.
        userValueVariables.balance = uint104(remaining);
        userValueVariables.sumOfEntryTimes = uint152(newEntryTimes);
    }

    // Reset the previous values, as we have restarted the staking duration.
    user.previousValues = 0;

    // Snapshot the lastUpdate and summations.
    _snapshotRewardSummations(pool, user);

    // Transfer withdrawn tokens.
    if (reward != 0) rewardsToken.safeTransfer(msg.sender, reward);
    if (amount != 0) ERC20(pool.tokenOrRecipient).safeTransfer(msg.sender,
amount);
    emit Withdrawn(poolId, msg.sender, amount, reward);

    // Get extra rewards from rewarder if it is not an emergency exit.
    IRewarder rewarder = pool.rewarder;
    if (address(rewarder) != address(0)) {
        rewarder.onReward(poolId, msg.sender, msg.sender, reward, remaining);
    }
}
```

By contrast, `_emergencyExit()` will call `pool.rewarder` without checking if `address(pool.rewarder) != address(0)` .

We believe that's inconsistent and will waste some gas if `rewarder == address(0)` .

https://github.com/pangolindex/exchange-contracts/blob/6739cd2ed98903c39f8173eff7565c56e4d97456/contracts/staking-positions/PangoChef.sol#L694-L756

```
function _emergencyExit(uint256 poolId, bool withdrawStake) private {
    // Create storage pointers for the pool and the user.
    Pool storage pool = pools[poolId];
    User storage user = pool.users[msg.sender];

    // Ensure pool is ERC20 type.
    _onlyERC20Pool(pool);

    // Decrement lock count on pool zero if this pool was locking it.
    _decrementLockOnPoolZero(poolId, user);

    // Create storage pointers for the value variables.
    ValueVariables storage poolValueVariables = pool.valueVariables;
    ValueVariables storage userValueVariables = user.valueVariables;

    // Decrement the state variables pertaining to total value calculation.
    uint104 balance = userValueVariables.balance;
    if (balance == 0) revert NoEffect();
    unchecked {
        poolValueVariables.balance -= balance;
        poolValueVariables.sumOfEntryTimes -= userValueVariables.sumOfEntryTimes;
    }

    // Simply delete the user information.
    delete pools[poolId].users[msg.sender];

    // Transfer stake from contract to user and emit the associated event.
    if (withdrawStake) {
        ERC20(pool.tokenOrRecipient).safeTransfer(msg.sender, balance);
        emit Withdrawn(poolId, msg.sender, balance, 0);
    // Still try withdrawing, but do a non-reverting low-level call.
    } else {
        (bool success, bytes memory returndata) = pool.tokenOrRecipient.call(
            abi.encodeWithSelector(ERC20.transfer.selector, msg.sender, balance)
        );
        if (success && returndata.length > 0 && abi.decode(returndata, (bool))) {
            emit Withdrawn(poolId, msg.sender, balance, 0);
        }
    }

    {
        // Do a low-level call for rewarder. If external function reverts, only the external
        // contract reverts. To prevent DOS, this function (_emergencyExit) must never revert
        // unless `balance == 0`. This can still return true if rewarder is not a contract.
        (bool success, ) = address(pool.rewarder).call(
            abi.encodeWithSelector(
                IRewarder.onReward.selector,
                poolId,
                msg.sender,
                msg.sender,
                0,
                0
            )
        );

        // Record last failed Rewarder calls. This can be used for slashing rewards by a
        // non-malicious Rewarder just in case it reverts due to some bug. If rewarder is
        // correctly written, this statement should never execute. We also do not care if
        // `success` is `true` due to rewarder not being a contract. A non-contract rewarder
        // only means that it is unset. So it does not matter if we record or not.
        if (!success) lastTimeRewarderCallFailed[poolId][msg.sender] = block.timestamp;
    }
}
```

Recommendation

Consider checking if `address(pool.rewarder) != address(0)` before doing the low-level call for rewarder.

Status

ⓘ **Acknowledged**

WP-G5: Immutable variables should not be copied to memory

Gas

Issue Description

https://docs.soliditylang.org/en/v0.8.16/contracts.html?highlight=immutable#constant-and-immutable-state-variables

> Immutable variables are evaluated once at construction time and their value is copied to all the places in the code where they are accessed.

Coping immutable variables to memory is a waste of gas. For example:

https://github.com/pangolindex/exchange-contracts/blob/328067a5af7bbc67360ca9a0c5bda7c350234371/contracts/staking-positions/PangoChefFunding.sol#L81-L82

```
    /** @notice The reward token that is distributed to stakers. */
    ERC20 public immutable rewardsToken;
```

https://github.com/pangolindex/exchange-contracts/blob/328067a5af7bbc67360ca9a0c5bda7c350234371/contracts/staking-positions/PangoChef.sol#L647-L697

```
    function _addLiquidity(
        Pool storage pool,
        uint256 rewardAmount,
        Slippage memory slippage
    ) private returns (uint256 poolTokenAmount) {
        address poolToken = pool.tokenOrRecipient;
        address rewardPair = pool.rewardPair;

        // Get token amounts from the pool.
        (uint256 reserve0, uint256 reserve1, ) =
IPangolinPair(poolToken).getReserves();

        // Get the reward token’s pair’s amount from the reserves.
        ERC20 tmpRewardsToken = rewardsToken;
        uint256 pairAmount = address(tmpRewardsToken) < rewardPair
            ? (reserve1 * rewardAmount) / reserve0
            : (reserve0 * rewardAmount) / reserve1;

        // Ensure slippage is not above the limit.
        if (pairAmount > slippage.maxPairAmount) revert HighSlippage();
        if (pairAmount < slippage.minPairAmount) revert HighSlippage();

        // Non-zero message value signals desire to pay with native token.
        if (msg.value > 0) {
            // Ensure reward pair is native token.
            if (rewardPair != wrappedNativeToken) revert InvalidToken();

            // Ensure consistent slippage control.
            if (msg.value != slippage.maxPairAmount) revert InvalidAmount();

            // Wrap the native token.
            IWAVAX(rewardPair).deposit{ value: pairAmount }();

            // Transfer reward pair tokens from this contract to the pair
contract.
            ERC20(rewardPair).safeTransfer(poolToken, pairAmount);

            // Refund user.
            unchecked {
                uint256 refundAmount = msg.value - pairAmount;
                if (refundAmount != 0) SafeTransferLib.safeTransferETH(msg.sender,
refundAmount);
            }
        } else {
            // Transfer reward pair tokens from the user to the pair contract.
            ERC20(rewardPair).safeTransferFrom(msg.sender, poolToken, pairAmount);
        }

        // Transfer reward tokens from the contract to the pair contract.
        tmpRewardsToken.safeTransfer(poolToken, rewardAmount);

        // Mint liquidity tokens to the PangoChef and return the amount minted.
        poolTokenAmount = IPangolinPair(poolToken).mint(address(this));
    }
```

Recommendation

Using `rewardsToken` directly and remove `tmpRewardsToken` .

Status

✓ **Fixed**

Appendix

Timeliness of content

The content contained in the report is current as of the date appearing on the report and is subject to change without notice, unless indicated otherwise by WatchPug; however, WatchPug does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following publication.

Disclaimer