Guard

Smart contracts security assessment

Final report

Tariff: Standard

HLAW

December 2024

0xguard.com

hello@0xguard.com

Contents

Introduction

The report has been prepared for **HLAW**.

The audited project consists of the following contracts.

The HLAWToken ERC20 token to be minted and burned (without approval) by the HLAWExchange contract.

The HLAWExchange contract to allow users exchange between HLAW and DAI/WPLS pair tokens. Additional fees up to 10% may be applied, fees are split and transferred to reward pools and referral contract. DAI/WPLS tokens are deposited to external PulseX Farm contract, and any external rewards from that farm are redirected to the HLAWStaking contract as incentive rewards.

The HLAWStaking contract to provide staking pools with different reward tokens, but shared incentive reward pool (reward token of the PulseX Farm) and instant reward pool (HLAW reward token).

The HLAWReferral contract to provide referral model for the HLAWExchange operations.

The HLAWZapper to ease user experience for obtaining DAI/WPLS pair tokens.

The HLAWRewardPool, HLAWIncRewardPool, HLAWInstantRewardPool contracts are used to hold only specific ERC20 tokens, which are approved to the HLAWStaking address for `type(uint256).max` amount.

The HLAWPrizePool contract holds HLAW tokens to be distributed to the list of addresses selected by the `signer` address.

The HLAWBoost contract holds HLAW tokens to be distributed as instant reward of the HLAWStaking.

The md5 sums of the files under investigation:

59a353fa439532ce6484f3e329f1e2d0 - HLAWToken.sol

484515fc8d9e6cc8fa16152019564219 - HLAWStaking.sol

036bc8cb18151767569d17129d8de6c1 - HLAWExchange.sol

094a4a4866eec765bb8bca00a6f89903 - HLAWReferral.sol

b23b698cad58ed3bc04de14431035f6d - HLAWZapper.sol

725d7f6fe089599b5b8fff7161d61093 - HLAWRewardPool.sol

41de1cedb05e8537333b753ba04d5bc1 - HLAWPrizePool.sol

6976d0d1bc7698bca16a19a6d1b2462e - HLAWInstantRewardPool.sol

8f7e03ce35fa177f72ec811260ec13f9 - HLAWIncRewardPool.sol

af99dab484cd9aedc95765c443f134ea - HLAWBoost.sol

Report Update

The contracts' code was updated according to this report.

The md5 sums of the updated files:

3efa5718eff7d1f8a39c22565ac9e70e - HLAWStaking.sol

8d9e1b00303cc7f562d48514fa5999b6 - HLAWExchange.sol

d472487e4c6c763863f5f2cbcfaa9004 - HLAWZapper.sol

ff368e17a617499bda071fcc0d84c7f2 - HLAWPrizePool.sol

Report Update #2

The contracts' code was updated according to this report.

The md5 sums of the updated files:

a95755a961876861a47789e06ad06cf7 - HLAWToken.sol

fc9f31f372ded3e28854a4c03c01c7a4 - HLAWStaking.sol

8e756f12fc5de6bf89b765d56bc241b0 - HLAWExchange.sol

58e7575a409853a2c4423f233d3903f7 - HLAWReferral.sol

3342ad30caa7e5a069771edfbf7d4cf4 - HLAWZapper.sol

45f52e02f868965af1637d2a23ac676f - HLAWRewardPool.sol

f27c82252ae1b110f6f5da2be28aa9c9 - HLAWPrizePool.sol

0f40b62b9ef2326db0d787c072dd594c - HLAWInstantRewardPool.sol

37f6209c427b7993a5a9a3de0907ae82 - HLAWIncentiveRewardPool.sol

32bf52f3f13e07cceaa993f65e559633 - HLAWBoost.sol

Report Update #3

The contracts' code was updated according to this report.

The md5 sums of the updated files:

7fc7ef0fbd90bb908fd299910fd500f7 - HLAWExchange.sol

99f22947d7d78daf4255636965485af6 - HLAWReferral.sol

7d27187c23265da7d4a2508cfe0d9cc5 - HLAWZapper.sol

aee7556363e2862d99c4934fb5cd3230 - HLAWPrizePool.sol

Name	HLAW
Audit date	2024-10-15 - 2024-12-07
Language	Solidity
Platform	Pulse Chain

Contracts checked

Name	Address
HLAWStaking.sol	0x55d1C8e1F854f12F47F3FB98268fF883FA7369D6
HLAWZapper.sol	0x9B24A6Bf2A27ca8b4Af226A9c7e0F80a718deEba
HLAWToken	0xd197D8ECCA9b5B35d5f3b42cb874E2819f315224
HLAWReferral.sol	0xd910E9310c10664A2493670617974CF80Ca5489D
HLAWRewardPool.sol	0x1225e657844584f623C0cC6896Ca9e56Fc6fB6b5
HLAWIncentiveRewardPool.sol	0xbF2D4eA035c1b6530f14B4cD63ef9528Ee0871A2
HLAWInstantRewardPool.sol	0xb37D077e5e6A2720503809b6fAf076D78867EC32
HLAWBoost.sol	0x0e45b05D2d32a94650124F08576Abae86f104971
HLAWPrizePool.sol	0x873debA13845E6f79542b753f6CC4B2EbE861D80
HLAWExchange.sol	0xa224C402Edf26E3027416d85f25d4D7B5b43E015
All audited contracts	

Procedure

We perform our audit according to the following procedure:

Automated analysis

- Scanning the project's smart contracts with several publicly available automated Solidity analysis tools
- Manual verification (reject or confirm) all the issues found by the tools

Manual audit

- Manually analyze smart contracts for security vulnerabilities
- Smart contracts' logic check

Known vulnerabilities checked

Title	Check result
Unencrypted Private Data On-Chain	passed
Code With No Effects	passed
Message call with hardcoded gas amount	not passed
Typographical Error	passed
DoS With Block Gas Limit	passed
Presence of unused variables	passed
Incorrect Inheritance Order	passed
Requirement Violation	passed
Weak Sources of Randomness from Chain Attributes	passed
Shadowing State Variables	passed
Incorrect Constructor Name	passed
Block values as a proxy for time	passed
Authorization through tx.origin	passed
DoS with Failed Call	passed
Delegatecall to Untrusted Callee	passed
Use of Deprecated Solidity Functions	passed
Assert Violation	passed
State Variable Default Visibility	passed
Reentrancy	passed

Unprotected SELFDESTRUCT Instruction	passed
Unprotected Ether Withdrawal	passed
Unchecked Call Return Value	passed
Floating Pragma	passed
Outdated Compiler Version	passed
Integer Overflow and Underflow	passed
Function Default Visibility	passed

Classification of issue severity

High severity	High severity issues can cause a significant or full loss of funds, change of contract ownership, major interference with contract logic. Such issues require immediate attention.
Medium severity	Medium severity issues do not pose an immediate risk, but can be detrimental to the client's reputation if exploited. Medium severity issues may lead to a contract failure and can be fixed by modifying the contract state or redeployment. Such issues require attention.
Low severity	Low severity issues do not cause significant destruction to the contract's functionality. Such issues are recommended to be taken into consideration.

Issues

High severity issues

1. Stake is not open for public use (HLAWStaking.sol)
Status: Fixed

The function `stake` checks and adds user's address to the `existingUser` mapping, but the `addUser` function is restricted from public use. So only the HLAWExchange contract can call the stake function by using `HLAWExchange.buy(amount, referral, true)`.

```
    function stake(uint256 _pid, uint256 _amount) public nonReentrant {
        . . .
        if (!existingUser[msg.sender]) {
            addUser(msg.sender);
        }
    }

    function addUser(address user) public {
        require(
            msg.sender == address(hlawExchange) || msg.sender == address(this),
            "Only hlaw exchange and staking contract can call addUser"
        );
        if (!existingUser[user]) {
            existingUser[user] = true;
            totalUsers += 1;
        }
    }
```

Recommendation: Split the `addUser` to external and internal functions.

2. Pools are not updated (HLAWStaking.sol)

Status: Fixed

The function `_updatePool` is never used, calling it without `()` doesn't invocate function.

```
    function claimIncentives() public nonReentrant {
        . . .
        if (autoUpdate && lastUpdateTime + updateInterval <= block.timestamp) {
            _updatePool;
        }
    }
```

Recommendation: Use `_updatePool()` in all affected functions.

3. Different token amounts are added up (HLAWStaking.sol)

Status: Fixed

The variable `totalStakedTokens` is used to store total staked amount of all staking pools, containing different ERC20 tokens. The `totalStakedTokens` is also used to calculate pending

instant rewards, incentive rewards, and boost rewards. Users of pools with staking token with higher decimals or pools with higher total supply will receive higher rewards, regardless actual value of staked tokens or pool's allocation.

The same applies to user's total staked amount in the `pendingIncentives` and `pendingInstantRewards` functions.

```
function stake(uint256 _pid, uint256 _amount) public nonReentrant {
    . . .
    totalStakedTokens = totalStakedTokens.add(_amount);
}

function distributeInstantRewardDividends(uint256 _amount) external {
    . . .
    uint256 instantRewardDistributionAmount = _amount.mul(MULTIPLIER).div(
        totalStakedTokens
    );
    accInstantRewardsPerShare = accInstantRewardsPerShare.add(
        instantRewardDistributionAmount
    );
    . . .
}

function pendingInstantRewards(
    address _user
) public view returns (uint256) {
    uint256 _totalStaked = 0;
    for (uint256 _pid = 1; _pid <= poolLength(); _pid++) {
        UserInfo storage user = userInfo[_pid][_user];
        _totalStaked = _totalStaked.add(user.amount);
    }

    I2Reward storage i2Reward = i2RewardInfo[_user];

    uint256 _pendingInstantReward = accInstantRewardsPerShare
        .sub(i2Reward.lastAccInstantRewardsPerShare)
        .mul(_totalStaked)
        .div(MULTIPLIER);
```

```
        return _pendingInstantReward;
    }

    function pendingIncentives(address _user) public view returns (uint256) {
        uint256 _totalStaked = 0;
        for (uint256 _pid = 1; _pid <= poolLength(); _pid++) {
            UserInfo storage user = userInfo[_pid][_user];
            _totalStaked = _totalStaked.add(user.amount);
        }

        I2Reward storage i2Reward = i2RewardInfo[_user];

        uint256 _pendingIncentive = accIncentiveRewardsPerShare
            .sub(i2Reward.lastAccIncentiveRewardsPerShare)
            .mul(_totalStaked)
            .div(MULTIPLIER);

        return _pendingIncentive;
    }
```

Recommendation: Fix the rewarding logic.

Comment from HLAW team: The contract's `initializePool` function only allows for one single pool to be added. This makes it impossible for an issue to arise.

Update: The issue is marked as FIXED according to the team comment. Any possible reuse or rework of the contract's code should be aware of this issue.

4. Possible lack of rewards (HLAWStaking.sol)

Status: Fixed

Pools of the HLAWStaking contract receive their primary rewards from external `pool.rewardsPool` address. Each pool has its own reward rate, and the total pool's reward is split between all tokens staked in that pool. The HLAWStaking contract doesn't check if the reward pools have sufficient tokens for rewards, and the claiming function nullifies user's rewards if there's not enough reward tokens available.

```
    function pendingRewards(
        uint256 _pid,
        address _user
    ) public view returns (uint256) {
        . . .

        uint256 beforeRewards = user.amount.mul(
            pool.accRewardsPerShare.sub(user.lastAccRewardsPerShare)
        );
        uint256 timeDiff = getTimeDiff(pool.lastCalcTime, block.timestamp);
        uint256 afterRewards = (pool.allocPoint)
            .mul(timeDiff)
            .mul(user.amount)
            .mul(MULTIPLIER)
            .div(SECONDS_PER_YEAR)
            .div(pool.totalStaked);

        return beforeRewards.add(afterRewards).div(MULTIPLIER);
    }

    function safeRewardTransfer(
        uint256 _pid,
        address _to,
        uint256 _amount
    ) internal {
        PoolInfo storage pool = poolInfo[_pid];
        uint256 rewardBal = pool.dripToken.balanceOf(pool.rewardsPool);
        if (_amount > rewardBal) {
            pool.dripToken.safeTransferFrom(pool.rewardsPool, _to, rewardBal);
        } else {
            pool.dripToken.safeTransferFrom(pool.rewardsPool, _to, _amount);
        }
    }
```

Recommendation: Ensure available rewards or avoid nullifying user's pending rewards.

5. Incorrect slippage calculation (HLAWZapper.sol)

Status: Fixed

The UniswapV2 router's slippage parameter is calculated incorrectly as

`pulseRouter.getAmountOut` always return instant amount that is calculated from current reserves, so setting any `amountOutMin` less that `getAmountOut` is an equivalent to use 100% slippage.

```
function _swapPls(
    uint256 _amount,
    uint256 _slippage
) internal returns (uint256, uint256) {
    . . .
    // Calculate slippage
    uint256 expectedAmount = getExpectedAmountOut(
        _amount,
        address(wpls),
        address(dai)
    );
    uint256 slippage = (expectedAmount * _slippage) / denominator;
    . . .
}

function _swapTokens(
    address _curr,
    uint256 _amount,
    uint256 _slippage
) internal returns (uint256, uint256) {
    . . .
    // Calculate slippage
    uint256 expectedAmount = getExpectedAmountOut(
        _amount,
        _curr,
        address(wpls)
    );
    uint256 slippage = (expectedAmount * _slippage) / denominator;
    . . .
}
```

Recommendation: Use absolute off-chain value received as parameter.

6. Incorrect user address (HLAWReferral.sol)

Status: Fixed

The `hlawExchange` contract calls the `addReward` with the user's referrer address. The `addReward` function increases the `rewardsEarned` amount for the referrer of the referrer of the user.

```
contract HLAWExchange is ReentrancyGuard, Ownable {
    function buy(
        uint256 amount,
        address referral,
        bool autoStake
    ) public nonReentrant {
        hlawReferral.addReward(
            hlawReferral.getReferrer(msg.sender),
            toReferral
        );
    }
}

contract HLAWReferral is Ownable, ReentrancyGuard {
    function addReward(address user, uint256 amount) external {
        require(
            msg.sender == hlawExchange,
            "Only the HLAW Exchange Contract can call addReward."
        );

        address referrer = referrers[user];
        if (referrer != address(0)) {
            refStats[referrer].rewardsEarned += amount;

            for (uint256 i = 0; i < referralDetails[referrer].length; i++) {
                if (referralDetails[referrer][i].user == user) {
                    referralDetails[referrer][i].rewardGenerated += amount;
                    break;
                }
            }

            emit RewardAdded(user, amount);
        }
    }
```

```
}
```

Recommendation: Fix either HLAWReferral or HLAWExchange contracts.

7. distributeWinners function can be replayed (HLAWPrizePool.sol)

Status: Fixed

The `distributeWinners` is not secured against replaying as it only requires the session to be ended, but the session's `paid` parameter is never checked.

```
    /**
     * @dev Distributes HLAW tokens to multiple winners.
     */

    function distributeWinners() external {
        require(
            msg.sender == signer || msg.sender == owner(),
            "Only signer or owner can distributeWinners"
        );
        require(
            sessions[sessionIndex].ended == true,
            "The session has not ended yet."
        );

        address[] memory winnersSubset = new address[](10);
        uint256[] memory amountsSubset = new uint256[](10);

        for (uint256 k = 0; k < 10; k++) {
            winnersSubset[k] = (k > sessions[sessionIndex].winners.length - 1)
                ? address(0)
                : sessions[sessionIndex].winners[k];
            amountsSubset[k] = (k > sessions[sessionIndex].numOfWinners - 1)
                ? 0
                : (sessions[sessionIndex].prizePercents[k] *
                    hlawToken.balanceOf(address(this))) / denominator;
        }

        for (uint256 i = 0; i < sessions[sessionIndex].numOfWinners; i++) {
            if (amountsSubset[i] == 0 || winnersSubset[i] == address(0)) {
```

```
                continue;
            }

            hlawToken.transfer(winnersSubset[i], amountsSubset[i]);
            rewardsPaid += amountsSubset[i];
        }

        sessions[sessionIndex].paid = true;
        lastRewardTime = block.timestamp;
        sessionIndex++;

        emit WinnersDistributed(winnersSubset, amountsSubset, sessionIndex - 1);
    }
```

Recommendation: Include safety check to prevent replicability.

8. No tests provided (All audited contracts)

Status: Partially fixed

For the project, there were no testing offered except single test for `HLAWStaking.distributeInstantRewardDividends` function. We advise covering the core business logic scenarios with integration tests with adequate coverage until we can no longer ensure the project works in line with the documentation, as the audit uncovered several high severity issues and erroneous operation of many methods.

Medium severity issues

1. Cumulative pending reward is useless (HLAWStaking.sol)

Status: Fixed

The function `allPendingRewards` sums up all user's pending rewards from different pools, that have different reward tokens.

```
    function allPendingRewards(
        address _user
    ) public view returns (uint256[] memory) {
        uint256 length = poolLength();
```

```
        uint256[] memory rewards = new uint256[](length);

        for (uint256 _pid = 1; _pid <= length; _pid++) {
            rewards[_pid - 1] = pendingRewards(_pid, _user);
        }
        return (rewards);
    }
```

Recommendation: Remove `allPendingRewards` function or fix its logic.

2. Migration is complicated (HLAWStaking.sol)

Status: Fixed

Possible migration via the function `migrate` transfers staked user's funds and calls for `migrationAddress.migrate`, but that call doesn't contain information about the user, who is calling the HLAWStaking contract.

```
    function migrate(uint256 _pid) public nonReentrant {
        require(migration, "Migration must be enabled");

        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][msg.sender];

        require(user.amount > 0, "unstake: no tokens staked");

        calcAccRewardsPerShare(_pid);

        pool.totalStaked = pool.totalStaked.sub(user.amount);
        totalStakedTokens = totalStakedTokens.sub(user.amount);
        uint256 transferAmount = user.amount;
        user.amount = 0;

        pool.stakingToken.safeTransfer(
            address(migrationAddress),
            transferAmount
        );
        migrationAddress.migrate(transferAmount);

        emit Migrated(msg.sender, _pid, transferAmount);
```

```
    }
```

Recommendation: Consider including all information into the external migration call.

3. Disabled address is approved (HLAWZapper.sol)

Status: Fixed

Disabling token address with the `setCurr` also approves the token to `pulseRouter`.

```
    function setCurr(address _curr, bool _allowed) public onlyOwner {
        zapCurrs[_curr] = _allowed;
        IERC20(_curr).approve(address(pulseRouter), type(uint256).max);
    }
```

Recommendation: Consider using `SafeERC20.forceApprove(IERC20(_curr), address(pulseRouter), _allowed ? type(uint256).max : 0);`

Low severity issues

1. Token is mintable (HLAWToken)

Status: Open

The HLAW token can be minted and burned without permission from the holder's address. The `hlawExchange` getter returns the address of authorized contract that can mint and burn tokens. This address should be the address of the HLAWExchange contract, otherwise token is not secure and should not be used.

The `hlawExchange` address is set during the contract initialization and can't be updated later.

2. Incorrect condition (HLAWPrizePool.sol)

Status: Open

Session struct includes `startTime`, `endsAt`, `initialLength`, `lengthIncrement`, `maxLength` parameters. The `addBuyer` extends the session as `max(startTime + initialLength, min(endsAt + lengthIncrement, timestamp + maxLength))`. The session can be never ending, if the `addBuyer` is called before `session.endsAt`.

```
    /**
     * @dev Deposits HLAW to prize pool contract.
     * @param user The user of the deposit.
     * @param amount The amount being deposited.
     */

    function addBuyer(address user, uint256 amount) external {
        require(
            msg.sender == hlawExchange,
            "Only HLAW Exchange can use deposit function."
        );

        if (
            sessions[sessionIndex].ended ||
            sessions[sessionIndex].startTime == 0 ||
            sessions[sessionIndex].startTime > block.timestamp
        ) {
            _addBuyer(user, amount);
            return;
        }

        if (block.timestamp >= sessions[sessionIndex].endsAt) {
            sessions[sessionIndex].ended = true;
            isSessionActive = false;
            _selectWinners();
            _calculatePrizeAmounts();

            emit SessionEnded(sessionIndex);

            if (sessions[sessionIndex].autoPay) {
                _autoDistribute();
            }
            _addBuyer(user, amount);
        } else if (amount >= sessions[sessionIndex].minBuySize) {
            if (!sessions[sessionIndex].forceEnd) {
                uint256 initialEndAt = sessions[sessionIndex].startTime +
                    sessions[sessionIndex].initialLength;
                uint256 newEndAt = Math.min(
                    sessions[sessionIndex].endsAt +
                        sessions[sessionIndex].lengthIncrement,
                    block.timestamp + sessions[sessionIndex].maxLength
```

```
                );
                sessions[sessionIndex].endsAt = Math.max(
                    initialEndAt,
                    newEndAt
                );
            }
            _addBuyer(user, amount);
        }
    }
```

Recommendation: Consider using

```
                uint256 newEndAt = Math.min(
                    sessions[sessionIndex].endsAt +
                        sessions[sessionIndex].lengthIncrement,
                    sessions[sessionIndex].startTime + sessions[sessionIndex].maxLength
                );
```

Dev response: This sessions system is actually supposed to be this way, where it is possible for it to never end.

Conclusion

HLAW HLAWStaking.sol, HLAWZapper.sol, HLAWToken, HLAWReferral.sol, HLAWRewardPool.sol, HLAWIncentiveRewardPool.sol, HLAWInstantRewardPool.sol, HLAWBoost.sol, HLAWPrizePool.sol, HLAWExchange.sol, All audited contracts contracts were audited. 8 high, 3 medium, 2 low severity issues were found.
7 high, 3 medium severity issues have been fixed in the update.

The source code of the HLAWStaking and HLAWExchange contracts was verified by comparing it with the deployed bytecode.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability)set forth in the Services Agreement, or the scope of services, and terms and conditions provided to the Company in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without 0xGuard prior written consent.

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts 0xGuard to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Slither output

```
INFO:Detectors:
HLAWStaking.autoStake(uint256,address,uint256) (contracts/HLAWStaking.sol#766-792) uses
arbitrary from in transferFrom:
pool.stakingToken.safeTransferFrom(address(hlawExchange),address(this),_amount)
(contracts/HLAWStaking.sol#787)
HLAWStaking._claimInstantRewards(address) (contracts/HLAWStaking.sol#1053-1068) uses
arbitrary from in transferFrom:
hlawToken.safeTransferFrom(instantRewardPool,_user,pendingInstantReward) (contracts/
HLAWStaking.sol#1066)
HLAWStaking._claimIncentives(address,uint256,uint256,bool) (contracts/
HLAWStaking.sol#1070-1137) uses arbitrary from in transferFrom:
incToken.safeTransferFrom(incRewardPool,_user,_pendingIncentives) (contracts/
HLAWStaking.sol#1106)
HLAWStaking._claimIncentives(address,uint256,uint256,bool) (contracts/
HLAWStaking.sol#1070-1137) uses arbitrary from in transferFrom:
incToken.safeTransferFrom(incRewardPool,address(this),feeAmount) (contracts/
HLAWStaking.sol#1109)
HLAWStaking.safeRewardTransfer(uint256,address,uint256) (contracts/
HLAWStaking.sol#1174-1187) uses arbitrary from in transferFrom:
pool.dripToken.safeTransferFrom(pool.rewardsPool,_to,_amount) (contracts/
HLAWStaking.sol#1186)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#arbitrary-from-
in-transferfrom
INFO:Detectors:
HLAWPrizePool.distributeWinners() (contracts/HLAWPrizePool.sol#140-166) ignores return
value by hlawToken.transfer(winnersSubset[i],amountsSubset[i]) (contracts/
HLAWPrizePool.sol#157)
HLAWPrizePool._autoDistribute() (contracts/HLAWPrizePool.sol#279-302) ignores return
value by hlawToken.transfer(winnersSubset[i],amountsSubset[i]) (contracts/
HLAWPrizePool.sol#293)
HLAWReferral.claimReward() (contracts/HLAWReferral.sol#100-112) ignores return value by
hlawToken.transfer(msg.sender,refStats[msg.sender].rewardsEarned) (contracts/
HLAWReferral.sol#106)
HeartsLaw.withdrawTokens(address) (contracts/HLAWToken.sol#58-64) ignores return value
by token.transfer(address(msg.sender),amount) (contracts/HLAWToken.sol#63)
HLAWZapper.withdrawTokens(address,uint256) (contracts/HLAWZapper.sol#151-161) ignores
return value by IERC20(_tokenAddress).transfer(address(msg.sender),_amount) (contracts/
HLAWZapper.sol#160)
```

```
HLAWZapper.zapTokens(address,uint256,uint256,uint256,uint256) (contracts/
HLAWZapper.sol#184-252) ignores return value by
IERC20(_curr).transferFrom(msg.sender,address(this),_currAmount) (contracts/
HLAWZapper.sol#209)
HLAWZapper.zapTokens(address,uint256,uint256,uint256,uint256) (contracts/
HLAWZapper.sol#184-252) ignores return value by
IERC20(_curr).transferFrom(msg.sender,address(treasury),feeAmount) (contracts/
HLAWZapper.sol#211-215)
HLAWZapper.zapTokens(address,uint256,uint256,uint256,uint256) (contracts/
HLAWZapper.sol#184-252) ignores return value by dai.transfer(msg.sender,daiRefund)
(contracts/HLAWZapper.sol#241)
HLAWZapper.zapTokens(address,uint256,uint256,uint256,uint256) (contracts/
HLAWZapper.sol#184-252) ignores return value by
IERC20(_curr).transfer(msg.sender,currRefund) (contracts/HLAWZapper.sol#247)
HLAWZapper.zapPls(uint256) (contracts/HLAWZapper.sol#254-303) ignores return value by
dai.transfer(msg.sender,daiRemaining) (contracts/HLAWZapper.sol#298)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-
transfer
INFO:Detectors:
HLAWStaking.migrationAddress (contracts/HLAWStaking.sol#362) is never initialized. It
is used in:
        - HLAWStaking.migrate(uint256) (contracts/HLAWStaking.sol#587-611)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-
state-variables
INFO:Detectors:
HLAWExchange.constructor(address) (contracts/HLAWExchange.sol#197-217) uses a Boolean
constant improperly:
        -fees[true].instantRewardFee = 1000 (contracts/HLAWExchange.sol#213)
HLAWExchange.constructor(address) (contracts/HLAWExchange.sol#197-217) uses a Boolean
constant improperly:
        -fees[false].instantRewardFee = 1000 (contracts/HLAWExchange.sol#208)
HLAWExchange.constructor(address) (contracts/HLAWExchange.sol#197-217) uses a Boolean
constant improperly:
        -fees[true].referralFee = 0 (contracts/HLAWExchange.sol#216)
HLAWExchange.constructor(address) (contracts/HLAWExchange.sol#197-217) uses a Boolean
constant improperly:
        -fees[false].referralFee = 1000 (contracts/HLAWExchange.sol#211)
HLAWExchange.constructor(address) (contracts/HLAWExchange.sol#197-217) uses a Boolean
constant improperly:
        -fees[true].teamFee = 4000 (contracts/HLAWExchange.sol#214)
HLAWExchange.constructor(address) (contracts/HLAWExchange.sol#197-217) uses a Boolean
```

```
constant improperly:
        -fees[false].teamFee = 2000 (contracts/HLAWExchange.sol#209)
HLAWExchange.constructor(address) (contracts/HLAWExchange.sol#197-217) uses a Boolean
constant improperly:
        -fees[true].prizePoolFee = 1000 (contracts/HLAWExchange.sol#215)
HLAWExchange.constructor(address) (contracts/HLAWExchange.sol#197-217) uses a Boolean
constant improperly:
        -fees[true].rewardPoolFee = 4000 (contracts/HLAWExchange.sol#212)
HLAWExchange.constructor(address) (contracts/HLAWExchange.sol#197-217) uses a Boolean
constant improperly:
        -fees[false].rewardPoolFee = 4000 (contracts/HLAWExchange.sol#207)
HLAWExchange.constructor(address) (contracts/HLAWExchange.sol#197-217) uses a Boolean
constant improperly:
        -fees[false].prizePoolFee = 2000 (contracts/HLAWExchange.sol#210)
HLAWExchange.buy(uint256,address,bool) (contracts/HLAWExchange.sol#225-344) uses a
Boolean constant improperly:
        -toReferral = (fee * fees[false].referralFee) / FEE_DENOMINATOR (contracts/
HLAWExchange.sol#289)
HLAWExchange.buy(uint256,address,bool) (contracts/HLAWExchange.sol#225-344) uses a
Boolean constant improperly:
        -toInstantReward = (fee * fees[false].instantRewardFee) / FEE_DENOMINATOR
(contracts/HLAWExchange.sol#286)
HLAWExchange.buy(uint256,address,bool) (contracts/HLAWExchange.sol#225-344) uses a
Boolean constant improperly:
        -toRewardPool = (fee * fees[false].rewardPoolFee) / FEE_DENOMINATOR (contracts/
HLAWExchange.sol#285)
HLAWExchange.buy(uint256,address,bool) (contracts/HLAWExchange.sol#225-344) uses a
Boolean constant improperly:
        -toPrizePool = (fee * fees[false].prizePoolFee) / FEE_DENOMINATOR (contracts/
HLAWExchange.sol#288)
HLAWExchange.buy(uint256,address,bool) (contracts/HLAWExchange.sol#225-344) uses a
Boolean constant improperly:
        -toTeam = (fee * fees[false].teamFee) / FEE_DENOMINATOR (contracts/
HLAWExchange.sol#287)
HLAWExchange.sell(uint256) (contracts/HLAWExchange.sol#346-433) uses a Boolean constant
improperly:
        -toPrizePool = (fee * fees[true].prizePoolFee) / FEE_DENOMINATOR (contracts/
HLAWExchange.sol#365)
HLAWExchange.sell(uint256) (contracts/HLAWExchange.sol#346-433) uses a Boolean constant
improperly:
        -toRewardPool = (fee * fees[true].rewardPoolFee) / FEE_DENOMINATOR (contracts/
HLAWExchange.sol#362)
```

```
HLAWExchange.sell(uint256) (contracts/HLAWExchange.sol#346-433) uses a Boolean constant
improperly:
        -toTeam = (fee * fees[true].teamFee) / FEE_DENOMINATOR (contracts/
HLAWExchange.sol#364)
HLAWExchange.sell(uint256) (contracts/HLAWExchange.sol#346-433) uses a Boolean constant
improperly:
        -toReferral = (fee * fees[true].referralFee) / FEE_DENOMINATOR (contracts/
HLAWExchange.sol#366)
HLAWExchange.sell(uint256) (contracts/HLAWExchange.sol#346-433) uses a Boolean constant
improperly:
        -toInstantReward = (fee * fees[true].instantRewardFee) / FEE_DENOMINATOR
(contracts/HLAWExchange.sol#363)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#misuse-of-a-
boolean-constant
INFO:Detectors:
HLAWExchange.sell(uint256) (contracts/HLAWExchange.sol#346-433) performs a
multiplication on the result of a division:
        - fee = (amount * totalSellFee) / FEE_DENOMINATOR (contracts/
HLAWExchange.sol#358)
        - toRewardPool = (fee * fees[true].rewardPoolFee) / FEE_DENOMINATOR (contracts/
HLAWExchange.sol#362)
HLAWExchange.sell(uint256) (contracts/HLAWExchange.sol#346-433) performs a
multiplication on the result of a division:
        - fee = (amount * totalSellFee) / FEE_DENOMINATOR (contracts/
HLAWExchange.sol#358)
        - toInstantReward = (fee * fees[true].instantRewardFee) / FEE_DENOMINATOR
(contracts/HLAWExchange.sol#363)
HLAWExchange.sell(uint256) (contracts/HLAWExchange.sol#346-433) performs a
multiplication on the result of a division:
        - fee = (amount * totalSellFee) / FEE_DENOMINATOR (contracts/
HLAWExchange.sol#358)
        - toTeam = (fee * fees[true].teamFee) / FEE_DENOMINATOR (contracts/
HLAWExchange.sol#364)
HLAWExchange.sell(uint256) (contracts/HLAWExchange.sol#346-433) performs a
multiplication on the result of a division:
        - fee = (amount * totalSellFee) / FEE_DENOMINATOR (contracts/
HLAWExchange.sol#358)
        - toPrizePool = (fee * fees[true].prizePoolFee) / FEE_DENOMINATOR (contracts/
HLAWExchange.sol#365)
HLAWExchange.sell(uint256) (contracts/HLAWExchange.sol#346-433) performs a
multiplication on the result of a division:
```

```
        - fee = (amount * totalSellFee) / FEE_DENOMINATOR (contracts/
HLAWExchange.sol#358)
        - toReferral = (fee * fees[true].referralFee) / FEE_DENOMINATOR (contracts/
HLAWExchange.sol#366)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-
multiply
INFO:Detectors:
HLAWStaking._claim(uint256,address) (contracts/HLAWStaking.sol#1032-1051) uses a
dangerous strict equality:
        - pendingAmount == 0 (contracts/HLAWStaking.sol#1038)
HLAWStaking._distributeInstantRewardDividends(uint256) (contracts/
HLAWStaking.sol#1139-1150) uses a dangerous strict equality:
        - totalStakedTokens == 0 || _claimedAmount == 0 (contracts/
HLAWStaking.sol#1140)
HLAWStaking.distributeBoost(uint256) (contracts/HLAWStaking.sol#836-850) uses a
dangerous strict equality:
        - totalStakedTokens == 0 (contracts/HLAWStaking.sol#840)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-
strict-equalities
INFO:Detectors:
Reentrancy in HLAWPrizePool._autoDistribute() (contracts/HLAWPrizePool.sol#279-302):
        External calls:
        - hlawToken.transfer(winnersSubset[i],amountsSubset[i]) (contracts/
HLAWPrizePool.sol#293)
        State variables written after the call(s):
        - rewardsPaid += amountsSubset[i] (contracts/HLAWPrizePool.sol#294)
        HLAWPrizePool.rewardsPaid (contracts/HLAWPrizePool.sol#41) can be used in cross
function reentrancies:
        - HLAWPrizePool._autoDistribute() (contracts/HLAWPrizePool.sol#279-302)
        - HLAWPrizePool.distributeWinners() (contracts/HLAWPrizePool.sol#140-166)
        - HLAWPrizePool.rewardsPaid (contracts/HLAWPrizePool.sol#41)
        - sessionIndex ++ (contracts/HLAWPrizePool.sol#299)
        HLAWPrizePool.sessionIndex (contracts/HLAWPrizePool.sol#44) can be used in
cross function reentrancies:
        - HLAWPrizePool._autoDistribute() (contracts/HLAWPrizePool.sol#279-302)
        - HLAWPrizePool._calculatePrizeAmounts() (contracts/HLAWPrizePool.sol#264-273)
        - HLAWPrizePool._selectWinners() (contracts/HLAWPrizePool.sol#248-258)
        - HLAWPrizePool.addBuyer(address,uint256) (contracts/HLAWPrizePool.sol#198-233)
        - HLAWPrizePool.distributeWinners() (contracts/HLAWPrizePool.sol#140-166)
        - HLAWPrizePool.getSessionInfo(uint256) (contracts/HLAWPrizePool.sol#350-364)
        - HLAWPrizePool.sessionIndex (contracts/HLAWPrizePool.sol#44)
```

```
        - HLAWPrizePool.startSession(uint256,uint256,uint256,uint256,uint256,uint256[],u
int256,bool) (contracts/HLAWPrizePool.sol#78-119)
        - sessions[sessionIndex].paid = true (contracts/HLAWPrizePool.sol#297)
        HLAWPrizePool.sessions (contracts/HLAWPrizePool.sol#48) can be used in cross
function reentrancies:
        - HLAWPrizePool._autoDistribute() (contracts/HLAWPrizePool.sol#279-302)
        - HLAWPrizePool._calculatePrizeAmounts() (contracts/HLAWPrizePool.sol#264-273)
        - HLAWPrizePool._selectWinners() (contracts/HLAWPrizePool.sol#248-258)
        - HLAWPrizePool.addBuyer(address,uint256) (contracts/HLAWPrizePool.sol#198-233)
        - HLAWPrizePool.distributeWinners() (contracts/HLAWPrizePool.sol#140-166)
        - HLAWPrizePool.forceEnd(uint256,uint256) (contracts/HLAWPrizePool.sol#127-134)
        - HLAWPrizePool.getSessionInfo(uint256) (contracts/HLAWPrizePool.sol#350-364)
        - HLAWPrizePool.sessions (contracts/HLAWPrizePool.sol#48)
        - HLAWPrizePool.startSession(uint256,uint256,uint256,uint256,uint256,uint256[],u
int256,bool) (contracts/HLAWPrizePool.sol#78-119)
Reentrancy in HLAWStaking._autoStake(uint256,address,uint256) (contracts/
HLAWStaking.sol#1152-1172):
        External calls:
        - _claimIncentives(_user,_pid,_amount,true) (contracts/HLAWStaking.sol#1167)
                -
pulseRouter.swapExactTokensForTokens(amount,0,path,address(this),block.timestamp)
(contracts/HLAWStaking.sol#1241)
                - pulseRouter.swapExactTokensForTokens(wplsGained /
2,0,path2,address(this),block.timestamp) (contracts/HLAWStaking.sol#1252)
                - (None,None,lpTokenGained) =
pulseRouter.addLiquidity(address(wplsToken),address(daiToken),wplsGained /
2,daiGained,0,0,address(this),block.timestamp) (contracts/HLAWStaking.sol#1258)
                - hlawExchange.buyFromFee(lpTokenGained) (contracts/
HLAWStaking.sol#1264)
        State variables written after the call(s):
        - _updatePool() (contracts/HLAWStaking.sol#1170)
                - lastUpdateTime = lastUpdateTime.add(updateInterval) (contracts/
HLAWStaking.sol#1213)
        HLAWStaking.lastUpdateTime (contracts/HLAWStaking.sol#388) can be used in cross
function reentrancies:
        - HLAWStaking._autoStake(uint256,address,uint256) (contracts/
HLAWStaking.sol#1152-1172)
        - HLAWStaking._updatePool() (contracts/HLAWStaking.sol#1207-1221)
        - HLAWStaking.autoStake(uint256,address,uint256) (contracts/
HLAWStaking.sol#766-792)
        - HLAWStaking.lastUpdateTime (contracts/HLAWStaking.sol#388)
```

```
        - _updatePool() (contracts/HLAWStaking.sol#1170)
                - pool.allocPoint = _allocPoint (contracts/HLAWStaking.sol#1228)
                - pool.active = _active (contracts/HLAWStaking.sol#1229)
                - pool.accRewardsPerShare =
pool.accRewardsPerShare.add(newRewardsPerShare) (contracts/HLAWStaking.sol#1200-1202)
                - pool.lastCalcTime = block.timestamp (contracts/HLAWStaking.sol#1204)
        HLAWStaking.poolInfo (contracts/HLAWStaking.sol#381) can be used in cross
function reentrancies:
        - HLAWStaking._autoStake(uint256,address,uint256) (contracts/
HLAWStaking.sol#1152-1172)
        - HLAWStaking._claim(uint256,address) (contracts/HLAWStaking.sol#1032-1051)
        - HLAWStaking._claimIncentives(address,uint256,uint256,bool) (contracts/
HLAWStaking.sol#1070-1137)
        - HLAWStaking._set(uint256,uint256,bool) (contracts/HLAWStaking.sol#1223-1230)
        - HLAWStaking._updatePool() (contracts/HLAWStaking.sol#1207-1221)
        - HLAWStaking.autoStake(uint256,address,uint256) (contracts/
HLAWStaking.sol#766-792)
        - HLAWStaking.calcAccRewardsPerShare(uint256) (contracts/
HLAWStaking.sol#1189-1205)
        - HLAWStaking.getPidInfo(uint256) (contracts/HLAWStaking.sol#953-976)
        - HLAWStaking.initializePool(IERC20,IERC20,address,uint256) (contracts/
HLAWStaking.sol#709-743)
        - HLAWStaking.pendingRewards(uint256,address) (contracts/
HLAWStaking.sol#869-897)
        - HLAWStaking.safeRewardTransfer(uint256,address,uint256) (contracts/
HLAWStaking.sol#1174-1187)
        - HLAWStaking.set(uint256,uint256,bool) (contracts/HLAWStaking.sol#745-758)
Reentrancy in HLAWStaking._claimIncentives(address,uint256,uint256,bool) (contracts/
HLAWStaking.sol#1070-1137):
        External calls:
        - hlawGained = _incentiveSwap(feeAmount.sub(toTeam)) (contracts/
HLAWStaking.sol#1113)
                -
pulseRouter.swapExactTokensForTokens(amount,0,path,address(this),block.timestamp)
(contracts/HLAWStaking.sol#1241)
                - pulseRouter.swapExactTokensForTokens(wplsGained /
2,0,path2,address(this),block.timestamp) (contracts/HLAWStaking.sol#1252)
                - (None,None,lpTokenGained) =
pulseRouter.addLiquidity(address(wplsToken),address(daiToken),wplsGained /
2,daiGained,0,0,address(this),block.timestamp) (contracts/HLAWStaking.sol#1258)
                - hlawExchange.buyFromFee(lpTokenGained) (contracts/
HLAWStaking.sol#1264)
```

```
        - _autoStake(1,treasuryFeeReceiver,toTreasury) (contracts/HLAWStaking.sol#1133)
                -
pulseRouter.swapExactTokensForTokens(amount,0,path,address(this),block.timestamp)
(contracts/HLAWStaking.sol#1241)
                - pulseRouter.swapExactTokensForTokens(wplsGained /
2,0,path2,address(this),block.timestamp) (contracts/HLAWStaking.sol#1252)
                - (None,None,lpTokenGained) =
pulseRouter.addLiquidity(address(wplsToken),address(daiToken),wplsGained /
2,daiGained,0,0,address(this),block.timestamp) (contracts/HLAWStaking.sol#1258)
                - hlawExchange.buyFromFee(lpTokenGained) (contracts/
HLAWStaking.sol#1264)
        State variables written after the call(s):
        - _autoStake(1,treasuryFeeReceiver,toTreasury) (contracts/HLAWStaking.sol#1133)
                - i2Reward.lastAccInstantRewardsPerShare = accInstantRewardsPerShare
(contracts/HLAWStaking.sol#1057)
                - i2Reward.redeemedInstant =
i2Reward.redeemedInstant.add(pendingInstantReward) (contracts/HLAWStaking.sol#1063)
                - i2Reward.lastAccIncentiveRewardsPerShare =
accIncentiveRewardsPerShare (contracts/HLAWStaking.sol#1089)
                - i2Reward.redeemedIncentive =
i2Reward.redeemedIncentive.add(_pendingIncentives) (contracts/HLAWStaking.sol#1104)
        HLAWStaking.i2RewardInfo (contracts/HLAWStaking.sol#383) can be used in cross
function reentrancies:
        - HLAWStaking._claimIncentives(address,uint256,uint256,bool) (contracts/
HLAWStaking.sol#1070-1137)
        - HLAWStaking._claimInstantRewards(address) (contracts/
HLAWStaking.sol#1053-1068)
        - HLAWStaking.getIncentiveInfo(address) (contracts/HLAWStaking.sol#982-993)
        - HLAWStaking.getInstantRewardInfo(address) (contracts/
HLAWStaking.sol#995-1002)
        - HLAWStaking.pendingIncentives(address) (contracts/HLAWStaking.sol#899-911)
        - HLAWStaking.pendingInstantRewards(address) (contracts/
HLAWStaking.sol#913-929)
        - _autoStake(1,treasuryFeeReceiver,toTreasury) (contracts/HLAWStaking.sol#1133)
                - pool.allocPoint = _allocPoint (contracts/HLAWStaking.sol#1228)
                - pool.accRewardsPerShare =
pool.accRewardsPerShare.add(newRewardsPerShare) (contracts/HLAWStaking.sol#1200-1202)
                - pool.active = _active (contracts/HLAWStaking.sol#1229)
                - pool.totalStaked = pool.totalStaked.add(_stakeAmount) (contracts/
HLAWStaking.sol#1078)
                - pool.lastCalcTime = block.timestamp (contracts/HLAWStaking.sol#1204)
```

```
                - pool.totalRewards = pool.totalRewards.add(pendingAmount) (contracts/
HLAWStaking.sol#1043)
                - pool.totalStaked = pool.totalStaked.sub(_stakeAmount) (contracts/
HLAWStaking.sol#1082)
        HLAWStaking.poolInfo (contracts/HLAWStaking.sol#381) can be used in cross
function reentrancies:
        - HLAWStaking._autoStake(uint256,address,uint256) (contracts/
HLAWStaking.sol#1152-1172)
        - HLAWStaking._claim(uint256,address) (contracts/HLAWStaking.sol#1032-1051)
        - HLAWStaking._claimIncentives(address,uint256,uint256,bool) (contracts/
HLAWStaking.sol#1070-1137)
        - HLAWStaking._set(uint256,uint256,bool) (contracts/HLAWStaking.sol#1223-1230)
        - HLAWStaking._updatePool() (contracts/HLAWStaking.sol#1207-1221)
        - HLAWStaking.autoStake(uint256,address,uint256) (contracts/
HLAWStaking.sol#766-792)
        - HLAWStaking.calcAccRewardsPerShare(uint256) (contracts/
HLAWStaking.sol#1189-1205)
        - HLAWStaking.getPidInfo(uint256) (contracts/HLAWStaking.sol#953-976)
        - HLAWStaking.initializePool(IERC20,IERC20,address,uint256) (contracts/
HLAWStaking.sol#709-743)
        - HLAWStaking.pendingRewards(uint256,address) (contracts/
HLAWStaking.sol#869-897)
        - HLAWStaking.safeRewardTransfer(uint256,address,uint256) (contracts/
HLAWStaking.sol#1174-1187)
        - HLAWStaking.set(uint256,uint256,bool) (contracts/HLAWStaking.sol#745-758)
        - _autoStake(1,treasuryFeeReceiver,toTreasury) (contracts/HLAWStaking.sol#1133)
                - totalIncRewarded = totalIncRewarded.add(_pendingIncentives)
(contracts/HLAWStaking.sol#1097)
        HLAWStaking.totalIncRewarded (contracts/HLAWStaking.sol#394) can be used in
cross function reentrancies:
        - HLAWStaking._claimIncentives(address,uint256,uint256,bool) (contracts/
HLAWStaking.sol#1070-1137)
        - HLAWStaking.totalIncRewarded (contracts/HLAWStaking.sol#394)
        - _autoStake(1,treasuryFeeReceiver,toTreasury) (contracts/HLAWStaking.sol#1133)
                - totalStakedTokens = totalStakedTokens.add(_stakeAmount) (contracts/
HLAWStaking.sol#1080)
                - totalStakedTokens = totalStakedTokens.sub(_stakeAmount) (contracts/
HLAWStaking.sol#1084)
        HLAWStaking.totalStakedTokens (contracts/HLAWStaking.sol#387) can be used in
cross function reentrancies:
        - HLAWStaking._claimIncentives(address,uint256,uint256,bool) (contracts/
HLAWStaking.sol#1070-1137)
```

```
        - HLAWStaking._distributeInstantRewardDividends(uint256) (contracts/
HLAWStaking.sol#1139-1150)
        - HLAWStaking.distributeBoost(uint256) (contracts/HLAWStaking.sol#836-850)
        - HLAWStaking.distributeIncentiveDividends(uint256) (contracts/
HLAWStaking.sol#817-834)
        - HLAWStaking.distributeInstantRewardDividends(uint256) (contracts/
HLAWStaking.sol#794-815)
        - HLAWStaking.totalStakedTokens (contracts/HLAWStaking.sol#387)
        - _autoStake(1,treasuryFeeReceiver,toTreasury) (contracts/HLAWStaking.sol#1133)
                - user.totalRedeemed = user.totalRedeemed.add(pendingAmount) (contracts/
HLAWStaking.sol#1042)
                - user.amount = user.amount.add(_stakeAmount) (contracts/
HLAWStaking.sol#1079)
                - user.lastRewardTime = block.timestamp (contracts/
HLAWStaking.sol#1047)
                - user.amount = user.amount.sub(_stakeAmount) (contracts/
HLAWStaking.sol#1083)
                - user.lastAccRewardsPerShare = pool.accRewardsPerShare (contracts/
HLAWStaking.sol#1048)
        HLAWStaking.userInfo (contracts/HLAWStaking.sol#382) can be used in cross
function reentrancies:
        - HLAWStaking._autoStake(uint256,address,uint256) (contracts/
HLAWStaking.sol#1152-1172)
        - HLAWStaking._claim(uint256,address) (contracts/HLAWStaking.sol#1032-1051)
        - HLAWStaking._claimIncentives(address,uint256,uint256,bool) (contracts/
HLAWStaking.sol#1070-1137)
        - HLAWStaking.autoStake(uint256,address,uint256) (contracts/
HLAWStaking.sol#766-792)
        - HLAWStaking.getUserInfo(uint256,address) (contracts/HLAWStaking.sol#931-951)
        - HLAWStaking.pendingIncentives(address) (contracts/HLAWStaking.sol#899-911)
        - HLAWStaking.pendingInstantRewards(address) (contracts/
HLAWStaking.sol#913-929)
        - HLAWStaking.pendingRewards(uint256,address) (contracts/
HLAWStaking.sol#869-897)
Reentrancy in HLAWPrizePool.addBuyer(address,uint256) (contracts/
HLAWPrizePool.sol#198-233):
        External calls:
        - _autoDistribute() (contracts/HLAWPrizePool.sol#218)
                - hlawToken.transfer(winnersSubset[i],amountsSubset[i]) (contracts/
HLAWPrizePool.sol#293)
        State variables written after the call(s):
```

```
        - _addBuyer(user,amount) (contracts/HLAWPrizePool.sol#220)
                - buyers[buyersIndex] =
Buyers({index:buyersIndex,buyer:user,amount:amount,timestamp:block.timestamp})
(contracts/HLAWPrizePool.sol#240)
        HLAWPrizePool.buyers (contracts/HLAWPrizePool.sol#49) can be used in cross
function reentrancies:
        - HLAWPrizePool._addBuyer(address,uint256) (contracts/
HLAWPrizePool.sol#239-242)
        - HLAWPrizePool._selectWinners() (contracts/HLAWPrizePool.sol#248-258)
        - HLAWPrizePool.buyers (contracts/HLAWPrizePool.sol#49)
        - HLAWPrizePool.getBuyersInRange(uint256,uint256) (contracts/
HLAWPrizePool.sol#328-344)
        - HLAWPrizePool.getLastBuyers(uint256) (contracts/HLAWPrizePool.sol#309-320)
        - _addBuyer(user,amount) (contracts/HLAWPrizePool.sol#220)
                - buyersIndex ++ (contracts/HLAWPrizePool.sol#241)
        HLAWPrizePool.buyersIndex (contracts/HLAWPrizePool.sol#45) can be used in cross
function reentrancies:
        - HLAWPrizePool._addBuyer(address,uint256) (contracts/
HLAWPrizePool.sol#239-242)
        - HLAWPrizePool._selectWinners() (contracts/HLAWPrizePool.sol#248-258)
        - HLAWPrizePool.buyersIndex (contracts/HLAWPrizePool.sol#45)
        - HLAWPrizePool.getBuyersInRange(uint256,uint256) (contracts/
HLAWPrizePool.sol#328-344)
        - HLAWPrizePool.getLastBuyers(uint256) (contracts/HLAWPrizePool.sol#309-320)
Reentrancy in HLAWStaking.autoStake(uint256,address,uint256) (contracts/
HLAWStaking.sol#766-792):
        External calls:
        - _claimIncentives(_user,_pid,_amount,true) (contracts/HLAWStaking.sol#785)
                -
pulseRouter.swapExactTokensForTokens(amount,0,path,address(this),block.timestamp)
(contracts/HLAWStaking.sol#1241)
                - pulseRouter.swapExactTokensForTokens(wplsGained /
2,0,path2,address(this),block.timestamp) (contracts/HLAWStaking.sol#1252)
                - (None,None,lpTokenGained) =
pulseRouter.addLiquidity(address(wplsToken),address(daiToken),wplsGained /
2,daiGained,0,0,address(this),block.timestamp) (contracts/HLAWStaking.sol#1258)
                - hlawExchange.buyFromFee(lpTokenGained) (contracts/
HLAWStaking.sol#1264)
        State variables written after the call(s):
        - _updatePool() (contracts/HLAWStaking.sol#790)
                - lastUpdateTime = lastUpdateTime.add(updateInterval) (contracts/
HLAWStaking.sol#1213)
```

```
        HLAWStaking.lastUpdateTime (contracts/HLAWStaking.sol#388) can be used in cross
function reentrancies:
        - HLAWStaking._autoStake(uint256,address,uint256) (contracts/
HLAWStaking.sol#1152-1172)
        - HLAWStaking._updatePool() (contracts/HLAWStaking.sol#1207-1221)
        - HLAWStaking.autoStake(uint256,address,uint256) (contracts/
HLAWStaking.sol#766-792)
        - HLAWStaking.lastUpdateTime (contracts/HLAWStaking.sol#388)
        - _updatePool() (contracts/HLAWStaking.sol#790)
                - pool.allocPoint = _allocPoint (contracts/HLAWStaking.sol#1228)
                - pool.active = _active (contracts/HLAWStaking.sol#1229)
                - pool.accRewardsPerShare =
pool.accRewardsPerShare.add(newRewardsPerShare) (contracts/HLAWStaking.sol#1200-1202)
                - pool.lastCalcTime = block.timestamp (contracts/HLAWStaking.sol#1204)
        HLAWStaking.poolInfo (contracts/HLAWStaking.sol#381) can be used in cross
function reentrancies:
        - HLAWStaking._autoStake(uint256,address,uint256) (contracts/
HLAWStaking.sol#1152-1172)
        - HLAWStaking._claim(uint256,address) (contracts/HLAWStaking.sol#1032-1051)
        - HLAWStaking._claimIncentives(address,uint256,uint256,bool) (contracts/
HLAWStaking.sol#1070-1137)
        - HLAWStaking._set(uint256,uint256,bool) (contracts/HLAWStaking.sol#1223-1230)
        - HLAWStaking._updatePool() (contracts/HLAWStaking.sol#1207-1221)
        - HLAWStaking.autoStake(uint256,address,uint256) (contracts/
HLAWStaking.sol#766-792)
        - HLAWStaking.calcAccRewardsPerShare(uint256) (contracts/
HLAWStaking.sol#1189-1205)
        - HLAWStaking.getPidInfo(uint256) (contracts/HLAWStaking.sol#953-976)
        - HLAWStaking.initializePool(IERC20,IERC20,address,uint256) (contracts/
HLAWStaking.sol#709-743)
        - HLAWStaking.pendingRewards(uint256,address) (contracts/
HLAWStaking.sol#869-897)
        - HLAWStaking.safeRewardTransfer(uint256,address,uint256) (contracts/
HLAWStaking.sol#1174-1187)
        - HLAWStaking.set(uint256,uint256,bool) (contracts/HLAWStaking.sol#745-758)
Reentrancy in HLAWStaking.claimIncentives() (contracts/HLAWStaking.sol#571-577):
        External calls:
        - _claimIncentives(msg.sender,1,0,true) (contracts/HLAWStaking.sol#572)
                -
pulseRouter.swapExactTokensForTokens(amount,0,path,address(this),block.timestamp)
(contracts/HLAWStaking.sol#1241)
```

```
                - pulseRouter.swapExactTokensForTokens(wplsGained /
2,0,path2,address(this),block.timestamp) (contracts/HLAWStaking.sol#1252)
                - (None,None,lpTokenGained) =
pulseRouter.addLiquidity(address(wplsToken),address(daiToken),wplsGained /
2,daiGained,0,0,address(this),block.timestamp) (contracts/HLAWStaking.sol#1258)
                - hlawExchange.buyFromFee(lpTokenGained) (contracts/
HLAWStaking.sol#1264)
        State variables written after the call(s):
        - _updatePool() (contracts/HLAWStaking.sol#575)
                - lastUpdateTime = lastUpdateTime.add(updateInterval) (contracts/
HLAWStaking.sol#1213)
        HLAWStaking.lastUpdateTime (contracts/HLAWStaking.sol#388) can be used in cross
function reentrancies:
        - HLAWStaking._autoStake(uint256,address,uint256) (contracts/
HLAWStaking.sol#1152-1172)
        - HLAWStaking._updatePool() (contracts/HLAWStaking.sol#1207-1221)
        - HLAWStaking.autoStake(uint256,address,uint256) (contracts/
HLAWStaking.sol#766-792)
        - HLAWStaking.lastUpdateTime (contracts/HLAWStaking.sol#388)
        - _updatePool() (contracts/HLAWStaking.sol#575)
                - pool.allocPoint = _allocPoint (contracts/HLAWStaking.sol#1228)
                - pool.active = _active (contracts/HLAWStaking.sol#1229)
                - pool.accRewardsPerShare =
pool.accRewardsPerShare.add(newRewardsPerShare) (contracts/HLAWStaking.sol#1200-1202)
                - pool.lastCalcTime = block.timestamp (contracts/HLAWStaking.sol#1204)
        HLAWStaking.poolInfo (contracts/HLAWStaking.sol#381) can be used in cross
function reentrancies:
        - HLAWStaking._autoStake(uint256,address,uint256) (contracts/
HLAWStaking.sol#1152-1172)
        - HLAWStaking._claim(uint256,address) (contracts/HLAWStaking.sol#1032-1051)
        - HLAWStaking._claimIncentives(address,uint256,uint256,bool) (contracts/
HLAWStaking.sol#1070-1137)
        - HLAWStaking._set(uint256,uint256,bool) (contracts/HLAWStaking.sol#1223-1230)
        - HLAWStaking._updatePool() (contracts/HLAWStaking.sol#1207-1221)
        - HLAWStaking.autoStake(uint256,address,uint256) (contracts/
HLAWStaking.sol#766-792)
        - HLAWStaking.calcAccRewardsPerShare(uint256) (contracts/
HLAWStaking.sol#1189-1205)
        - HLAWStaking.getPidInfo(uint256) (contracts/HLAWStaking.sol#953-976)
        - HLAWStaking.initializePool(IERC20,IERC20,address,uint256) (contracts/
HLAWStaking.sol#709-743)
```

```
        - HLAWStaking.pendingRewards(uint256,address) (contracts/
HLAWStaking.sol#869-897)
        - HLAWStaking.safeRewardTransfer(uint256,address,uint256) (contracts/
HLAWStaking.sol#1174-1187)
        - HLAWStaking.set(uint256,uint256,bool) (contracts/HLAWStaking.sol#745-758)
Reentrancy in HLAWReferral.claimReward() (contracts/HLAWReferral.sol#100-112):
        External calls:
        - hlawToken.transfer(msg.sender,refStats[msg.sender].rewardsEarned) (contracts/
HLAWReferral.sol#106)
        State variables written after the call(s):
        - refStats[msg.sender].rewardsClaimed += refStats[msg.sender].rewardsEarned
(contracts/HLAWReferral.sol#109-110)
        HLAWReferral.refStats (contracts/HLAWReferral.sol#33) can be used in cross
function reentrancies:
        - HLAWReferral.addReward(address,uint256) (contracts/HLAWReferral.sol#79-98)
        - HLAWReferral.refStats (contracts/HLAWReferral.sol#33)
        - HLAWReferral.setReferral(address,address) (contracts/HLAWReferral.sol#56-73)
        - refStats[msg.sender].rewardsEarned = 0 (contracts/HLAWReferral.sol#111)
        HLAWReferral.refStats (contracts/HLAWReferral.sol#33) can be used in cross
function reentrancies:
        - HLAWReferral.addReward(address,uint256) (contracts/HLAWReferral.sol#79-98)
        - HLAWReferral.refStats (contracts/HLAWReferral.sol#33)
        - HLAWReferral.setReferral(address,address) (contracts/HLAWReferral.sol#56-73)
Reentrancy in HLAWPrizePool.distributeWinners() (contracts/HLAWPrizePool.sol#140-166):
        External calls:
        - hlawToken.transfer(winnersSubset[i],amountsSubset[i]) (contracts/
HLAWPrizePool.sol#157)
        State variables written after the call(s):
        - rewardsPaid += amountsSubset[i] (contracts/HLAWPrizePool.sol#158)
        HLAWPrizePool.rewardsPaid (contracts/HLAWPrizePool.sol#41) can be used in cross
function reentrancies:
        - HLAWPrizePool._autoDistribute() (contracts/HLAWPrizePool.sol#279-302)
        - HLAWPrizePool.distributeWinners() (contracts/HLAWPrizePool.sol#140-166)
        - HLAWPrizePool.rewardsPaid (contracts/HLAWPrizePool.sol#41)
        - sessionIndex ++ (contracts/HLAWPrizePool.sol#163)
        HLAWPrizePool.sessionIndex (contracts/HLAWPrizePool.sol#44) can be used in
cross function reentrancies:
        - HLAWPrizePool._autoDistribute() (contracts/HLAWPrizePool.sol#279-302)
        - HLAWPrizePool._calculatePrizeAmounts() (contracts/HLAWPrizePool.sol#264-273)
        - HLAWPrizePool._selectWinners() (contracts/HLAWPrizePool.sol#248-258)
        - HLAWPrizePool.addBuyer(address,uint256) (contracts/HLAWPrizePool.sol#198-233)
```

```
        - HLAWPrizePool.distributeWinners() (contracts/HLAWPrizePool.sol#140-166)
        - HLAWPrizePool.getSessionInfo(uint256) (contracts/HLAWPrizePool.sol#350-364)
        - HLAWPrizePool.sessionIndex (contracts/HLAWPrizePool.sol#44)
        - HLAWPrizePool.startSession(uint256,uint256,uint256,uint256,uint256,uint256[],u
int256,bool) (contracts/HLAWPrizePool.sol#78-119)
        - sessions[sessionIndex].paid = true (contracts/HLAWPrizePool.sol#161)
        HLAWPrizePool.sessions (contracts/HLAWPrizePool.sol#48) can be used in cross
function reentrancies:
        - HLAWPrizePool._autoDistribute() (contracts/HLAWPrizePool.sol#279-302)
        - HLAWPrizePool._calculatePrizeAmounts() (contracts/HLAWPrizePool.sol#264-273)
        - HLAWPrizePool._selectWinners() (contracts/HLAWPrizePool.sol#248-258)
        - HLAWPrizePool.addBuyer(address,uint256) (contracts/HLAWPrizePool.sol#198-233)
        - HLAWPrizePool.distributeWinners() (contracts/HLAWPrizePool.sol#140-166)
        - HLAWPrizePool.forceEnd(uint256,uint256) (contracts/HLAWPrizePool.sol#127-134)
        - HLAWPrizePool.getSessionInfo(uint256) (contracts/HLAWPrizePool.sol#350-364)
        - HLAWPrizePool.sessions (contracts/HLAWPrizePool.sol#48)
        - HLAWPrizePool.startSession(uint256,uint256,uint256,uint256,uint256,uint256[],u
int256,bool) (contracts/HLAWPrizePool.sol#78-119)
Reentrancy in HLAWExchange.init(address,address,address) (contracts/
HLAWExchange.sol#552-566):
        External calls:
        - IERC20(hlawToken).approve(_hlawStaking,type()(uint256).max) (contracts/
HLAWExchange.sol#561)
        - incToken.approve(address(_hlawStaking),type()(uint256).max) (contracts/
HLAWExchange.sol#562)
        State variables written after the call(s):
        - initialized = true (contracts/HLAWExchange.sol#564)
        HLAWExchange.initialized (contracts/HLAWExchange.sol#162) can be used in cross
function reentrancies:
        - HLAWExchange.init(address,address,address) (contracts/
HLAWExchange.sol#552-566)
Reentrancy in HLAWIncentiveRewardPool.init(address) (contracts/
HLAWIncentiveRewardPool.sol#19-26):
        External calls:
        - incToken.approve(hlawStaking,type()(uint256).max) (contracts/
HLAWIncentiveRewardPool.sol#23)
        State variables written after the call(s):
        - initialized = true (contracts/HLAWIncentiveRewardPool.sol#25)
        HLAWIncentiveRewardPool.initialized (contracts/HLAWIncentiveRewardPool.sol#15)
can be used in cross function reentrancies:
        - HLAWIncentiveRewardPool.init(address) (contracts/
HLAWIncentiveRewardPool.sol#19-26)
```

```
Reentrancy in HLAWInstantRewardPool.init(address) (contracts/
HLAWInstantRewardPool.sol#20-27):
        External calls:
        - hlawToken.approve(hlawStaking,type()(uint256).max) (contracts/
HLAWInstantRewardPool.sol#24)
        State variables written after the call(s):
        - initialized = true (contracts/HLAWInstantRewardPool.sol#26)
        HLAWInstantRewardPool.initialized (contracts/HLAWInstantRewardPool.sol#14) can
be used in cross function reentrancies:
        - HLAWInstantRewardPool.init(address) (contracts/
HLAWInstantRewardPool.sol#20-27)
Reentrancy in HLAWRewardPool.init(address) (contracts/HLAWRewardPool.sol#20-27):
        External calls:
        - hlawToken.approve(hlawStaking,type()(uint256).max) (contracts/
HLAWRewardPool.sol#24)
        State variables written after the call(s):
        - initialized = true (contracts/HLAWRewardPool.sol#26)
        HLAWRewardPool.initialized (contracts/HLAWRewardPool.sol#14) can be used in
cross function reentrancies:
        - HLAWRewardPool.init(address) (contracts/HLAWRewardPool.sol#20-27)
Reentrancy in HLAWStaking.migrate(uint256) (contracts/HLAWStaking.sol#587-611):
        External calls:
        - migrationAddress.migrate(msg.sender,transferAmount) (contracts/
HLAWStaking.sol#599)
        State variables written after the call(s):
        - pool.totalStaked = pool.totalStaked.sub(user.amount) (contracts/
HLAWStaking.sol#601)
        HLAWStaking.poolInfo (contracts/HLAWStaking.sol#381) can be used in cross
function reentrancies:
        - HLAWStaking._autoStake(uint256,address,uint256) (contracts/
HLAWStaking.sol#1152-1172)
        - HLAWStaking._claim(uint256,address) (contracts/HLAWStaking.sol#1032-1051)
        - HLAWStaking._claimIncentives(address,uint256,uint256,bool) (contracts/
HLAWStaking.sol#1070-1137)
        - HLAWStaking._set(uint256,uint256,bool) (contracts/HLAWStaking.sol#1223-1230)
        - HLAWStaking._updatePool() (contracts/HLAWStaking.sol#1207-1221)
        - HLAWStaking.autoStake(uint256,address,uint256) (contracts/
HLAWStaking.sol#766-792)
        - HLAWStaking.calcAccRewardsPerShare(uint256) (contracts/
HLAWStaking.sol#1189-1205)
        - HLAWStaking.getPidInfo(uint256) (contracts/HLAWStaking.sol#953-976)
```

```
        - HLAWStaking.initializePool(IERC20,IERC20,address,uint256) (contracts/
HLAWStaking.sol#709-743)
        - HLAWStaking.pendingRewards(uint256,address) (contracts/
HLAWStaking.sol#869-897)
        - HLAWStaking.safeRewardTransfer(uint256,address,uint256) (contracts/
HLAWStaking.sol#1174-1187)
        - HLAWStaking.set(uint256,uint256,bool) (contracts/HLAWStaking.sol#745-758)
        - user.amount = 0 (contracts/HLAWStaking.sol#603)
        HLAWStaking.userInfo (contracts/HLAWStaking.sol#382) can be used in cross
function reentrancies:
        - HLAWStaking._autoStake(uint256,address,uint256) (contracts/
HLAWStaking.sol#1152-1172)
        - HLAWStaking._claim(uint256,address) (contracts/HLAWStaking.sol#1032-1051)
        - HLAWStaking._claimIncentives(address,uint256,uint256,bool) (contracts/
HLAWStaking.sol#1070-1137)
        - HLAWStaking.autoStake(uint256,address,uint256) (contracts/
HLAWStaking.sol#766-792)
        - HLAWStaking.getUserInfo(uint256,address) (contracts/HLAWStaking.sol#931-951)
        - HLAWStaking.pendingIncentives(address) (contracts/HLAWStaking.sol#899-911)
        - HLAWStaking.pendingInstantRewards(address) (contracts/
HLAWStaking.sol#913-929)
        - HLAWStaking.pendingRewards(uint256,address) (contracts/
HLAWStaking.sol#869-897)
Reentrancy in HLAWStaking.stake(uint256,uint256) (contracts/HLAWStaking.sol#478-506):
        External calls:
        - _claimIncentives(msg.sender,_pid,_amount,true) (contracts/
HLAWStaking.sol#493)
                -
pulseRouter.swapExactTokensForTokens(amount,0,path,address(this),block.timestamp)
(contracts/HLAWStaking.sol#1241)
                - pulseRouter.swapExactTokensForTokens(wplsGained /
2,0,path2,address(this),block.timestamp) (contracts/HLAWStaking.sol#1252)
                - (None,None,lpTokenGained) =
pulseRouter.addLiquidity(address(wplsToken),address(daiToken),wplsGained /
2,daiGained,0,0,address(this),block.timestamp) (contracts/HLAWStaking.sol#1258)
                - hlawExchange.buyFromFee(lpTokenGained) (contracts/
HLAWStaking.sol#1264)
        State variables written after the call(s):
        - _updatePool() (contracts/HLAWStaking.sol#498)
                - lastUpdateTime = lastUpdateTime.add(updateInterval) (contracts/
HLAWStaking.sol#1213)
```

```
        HLAWStaking.lastUpdateTime (contracts/HLAWStaking.sol#388) can be used in cross
function reentrancies:
        - HLAWStaking._autoStake(uint256,address,uint256) (contracts/
HLAWStaking.sol#1152-1172)
        - HLAWStaking._updatePool() (contracts/HLAWStaking.sol#1207-1221)
        - HLAWStaking.autoStake(uint256,address,uint256) (contracts/
HLAWStaking.sol#766-792)
        - HLAWStaking.lastUpdateTime (contracts/HLAWStaking.sol#388)
        - _updatePool() (contracts/HLAWStaking.sol#498)
                - pool.allocPoint = _allocPoint (contracts/HLAWStaking.sol#1228)
                - pool.active = _active (contracts/HLAWStaking.sol#1229)
                - pool.accRewardsPerShare =
pool.accRewardsPerShare.add(newRewardsPerShare) (contracts/HLAWStaking.sol#1200-1202)
                - pool.lastCalcTime = block.timestamp (contracts/HLAWStaking.sol#1204)
        HLAWStaking.poolInfo (contracts/HLAWStaking.sol#381) can be used in cross
function reentrancies:
        - HLAWStaking._autoStake(uint256,address,uint256) (contracts/
HLAWStaking.sol#1152-1172)
        - HLAWStaking._claim(uint256,address) (contracts/HLAWStaking.sol#1032-1051)
        - HLAWStaking._claimIncentives(address,uint256,uint256,bool) (contracts/
HLAWStaking.sol#1070-1137)
        - HLAWStaking._set(uint256,uint256,bool) (contracts/HLAWStaking.sol#1223-1230)
        - HLAWStaking._updatePool() (contracts/HLAWStaking.sol#1207-1221)
        - HLAWStaking.autoStake(uint256,address,uint256) (contracts/
HLAWStaking.sol#766-792)
        - HLAWStaking.calcAccRewardsPerShare(uint256) (contracts/
HLAWStaking.sol#1189-1205)
        - HLAWStaking.getPidInfo(uint256) (contracts/HLAWStaking.sol#953-976)
        - HLAWStaking.initializePool(IERC20,IERC20,address,uint256) (contracts/
HLAWStaking.sol#709-743)
        - HLAWStaking.pendingRewards(uint256,address) (contracts/
HLAWStaking.sol#869-897)
        - HLAWStaking.safeRewardTransfer(uint256,address,uint256) (contracts/
HLAWStaking.sol#1174-1187)
        - HLAWStaking.set(uint256,uint256,bool) (contracts/HLAWStaking.sol#745-758)
Reentrancy in HLAWStaking.unstake(uint256,uint256) (contracts/HLAWStaking.sol#508-533):
        External calls:
        - _claimIncentives(msg.sender,_pid,_amount,false) (contracts/
HLAWStaking.sol#524)
                -
pulseRouter.swapExactTokensForTokens(amount,0,path,address(this),block.timestamp)
```

```
(contracts/HLAWStaking.sol#1241)
                - pulseRouter.swapExactTokensForTokens(wplsGained /
2,0,path2,address(this),block.timestamp) (contracts/HLAWStaking.sol#1252)
                - (None,None,lpTokenGained) =
pulseRouter.addLiquidity(address(wplsToken),address(daiToken),wplsGained /
2,daiGained,0,0,address(this),block.timestamp) (contracts/HLAWStaking.sol#1258)
                - hlawExchange.buyFromFee(lpTokenGained) (contracts/
HLAWStaking.sol#1264)
        State variables written after the call(s):
        - _updatePool() (contracts/HLAWStaking.sol#529)
                - lastUpdateTime = lastUpdateTime.add(updateInterval) (contracts/
HLAWStaking.sol#1213)
        HLAWStaking.lastUpdateTime (contracts/HLAWStaking.sol#388) can be used in cross
function reentrancies:
        - HLAWStaking._autoStake(uint256,address,uint256) (contracts/
HLAWStaking.sol#1152-1172)
        - HLAWStaking._updatePool() (contracts/HLAWStaking.sol#1207-1221)
        - HLAWStaking.autoStake(uint256,address,uint256) (contracts/
HLAWStaking.sol#766-792)
        - HLAWStaking.lastUpdateTime (contracts/HLAWStaking.sol#388)
        - _updatePool() (contracts/HLAWStaking.sol#529)
                - pool.allocPoint = _allocPoint (contracts/HLAWStaking.sol#1228)
                - pool.active = _active (contracts/HLAWStaking.sol#1229)
                - pool.accRewardsPerShare =
pool.accRewardsPerShare.add(newRewardsPerShare) (contracts/HLAWStaking.sol#1200-1202)
                - pool.lastCalcTime = block.timestamp (contracts/HLAWStaking.sol#1204)
        HLAWStaking.poolInfo (contracts/HLAWStaking.sol#381) can be used in cross
function reentrancies:
        - HLAWStaking._autoStake(uint256,address,uint256) (contracts/
HLAWStaking.sol#1152-1172)
        - HLAWStaking._claim(uint256,address) (contracts/HLAWStaking.sol#1032-1051)
        - HLAWStaking._claimIncentives(address,uint256,uint256,bool) (contracts/
HLAWStaking.sol#1070-1137)
        - HLAWStaking._set(uint256,uint256,bool) (contracts/HLAWStaking.sol#1223-1230)
        - HLAWStaking._updatePool() (contracts/HLAWStaking.sol#1207-1221)
        - HLAWStaking.autoStake(uint256,address,uint256) (contracts/
HLAWStaking.sol#766-792)
        - HLAWStaking.calcAccRewardsPerShare(uint256) (contracts/
HLAWStaking.sol#1189-1205)
        - HLAWStaking.getPidInfo(uint256) (contracts/HLAWStaking.sol#953-976)
        - HLAWStaking.initializePool(IERC20,IERC20,address,uint256) (contracts/
HLAWStaking.sol#709-743)
```

```
        - HLAWStaking.pendingRewards(uint256,address) (contracts/
HLAWStaking.sol#869-897)
        - HLAWStaking.safeRewardTransfer(uint256,address,uint256) (contracts/
HLAWStaking.sol#1174-1187)
        - HLAWStaking.set(uint256,uint256,bool) (contracts/HLAWStaking.sol#745-758)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-
vulnerabilities-1
INFO:Detectors:
HLAWStaking.setUpdateSettings(uint256,uint256,uint256,bool) (contracts/
HLAWStaking.sol#675-697) contains a tautology or contradiction:
        - require(bool,string)(updateThreshold >= 0 && updateThreshold <= 100,Maximum
of 100.) (contracts/HLAWStaking.sol#686-689)
HLAWStaking.setUpdateSettings(uint256,uint256,uint256,bool) (contracts/
HLAWStaking.sol#675-697) contains a tautology or contradiction:
        - require(bool,string)(updateRatio >= 0 && updateRatio <= 100,Maximum of 100.)
(contracts/HLAWStaking.sol#685)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#tautology-or-
contradiction
INFO:Detectors:
HLAWBoost.constructor(address,address) (contracts/HLAWBoost.sol#26-31) ignores return
value by hlawToken.approve(address(hlawStaking),type()(uint256).max) (contracts/
HLAWBoost.sol#29)
HLAWExchange.constructor(address) (contracts/HLAWExchange.sol#197-217) ignores return
value by daiWplsLpToken.approve(address(pulseFarm),type()(uint256).max) (contracts/
HLAWExchange.sol#200)
HLAWExchange.constructor(address) (contracts/HLAWExchange.sol#197-217) ignores return
value by wplsToken.approve(address(pulseRouter),type()(uint256).max) (contracts/
HLAWExchange.sol#201)
HLAWExchange.constructor(address) (contracts/HLAWExchange.sol#197-217) ignores return
value by daiToken.approve(address(pulseRouter),type()(uint256).max) (contracts/
HLAWExchange.sol#202)
HLAWExchange.constructor(address) (contracts/HLAWExchange.sol#197-217) ignores return
value by incToken.approve(address(pulseRouter),type()(uint256).max) (contracts/
HLAWExchange.sol#203)
HLAWExchange.buy(uint256,address,bool) (contracts/HLAWExchange.sol#225-344) ignores
return value by (refStaked,None,None,None) = hlawStaking.getUserInfo(PID,referral)
(contracts/HLAWExchange.sol#239)
HLAWExchange.buy(uint256,address,bool) (contracts/HLAWExchange.sol#225-344) ignores
return value by (None,None,totalStaked,None,None,None) = hlawStaking.getPidInfo(PID)
(contracts/HLAWExchange.sol#262)
HLAWExchange.buy(uint256,address,bool) (contracts/HLAWExchange.sol#225-344) ignores
```

```
return value by (None,None,totalStaked_scope_0,None,None,None) =
hlawStaking.getPidInfo(PID) (contracts/HLAWExchange.sol#296)
HLAWExchange.buy(uint256,address,bool) (contracts/HLAWExchange.sol#225-344) ignores
return value by (refStaked_scope_1,None,None,None) =
hlawStaking.getUserInfo(PID,hlawReferral.getReferrer(msg.sender)) (contracts/
HLAWExchange.sol#315)
HLAWExchange.sell(uint256) (contracts/HLAWExchange.sol#346-433) ignores return value by
(None,None,totalStaked,None,None,None) = hlawStaking.getPidInfo(PID) (contracts/
HLAWExchange.sol#373)
HLAWExchange.sell(uint256) (contracts/HLAWExchange.sol#346-433) ignores return value by
(None,None,totalStaked_scope_0,None,None,None) = hlawStaking.getPidInfo(PID) (contracts/
HLAWExchange.sol#410)
HLAWExchange.buyFromFee(uint256) (contracts/HLAWExchange.sol#441-482) ignores return
value by (None,None,totalStaked,None,None,None) = hlawStaking.getPidInfo(PID)
(contracts/HLAWExchange.sol#459)
HLAWExchange._incentiveSwap(uint256) (contracts/HLAWExchange.sol#503-544) ignores
return value by (None,None,lpTokenGained) =
pulseRouter.addLiquidity(address(wplsToken),address(daiToken),wplsGained /
2,daiGained,0,0,address(this),block.timestamp) (contracts/HLAWExchange.sol#529)
HLAWExchange.init(address,address,address) (contracts/HLAWExchange.sol#552-566) ignores
return value by IERC20(hlawToken).approve(_hlawStaking,type()(uint256).max) (contracts/
HLAWExchange.sol#561)
HLAWExchange.init(address,address,address) (contracts/HLAWExchange.sol#552-566) ignores
return value by incToken.approve(address(_hlawStaking),type()(uint256).max) (contracts/
HLAWExchange.sol#562)
HLAWIncentiveRewardPool.init(address) (contracts/HLAWIncentiveRewardPool.sol#19-26)
ignores return value by incToken.approve(hlawStaking,type()(uint256).max) (contracts/
HLAWIncentiveRewardPool.sol#23)
HLAWInstantRewardPool.init(address) (contracts/HLAWInstantRewardPool.sol#20-27) ignores
return value by hlawToken.approve(hlawStaking,type()(uint256).max) (contracts/
HLAWInstantRewardPool.sol#24)
HLAWRewardPool.init(address) (contracts/HLAWRewardPool.sol#20-27) ignores return value
by hlawToken.approve(hlawStaking,type()(uint256).max) (contracts/HLAWRewardPool.sol#24)
HLAWStaking.constructor(uint256,address,address,address,address,address,address,address,
address) (contracts/HLAWStaking.sol#444-470) ignores return value by
wplsToken.approve(address(pulseRouter),type()(uint256).max) (contracts/
HLAWStaking.sol#466)
HLAWStaking.constructor(uint256,address,address,address,address,address,address,address,
address) (contracts/HLAWStaking.sol#444-470) ignores return value by
daiToken.approve(address(pulseRouter),type()(uint256).max) (contracts/
HLAWStaking.sol#467)
```

```
HLAWStaking.constructor(uint256,address,address,address,address,address,address,address,
address) (contracts/HLAWStaking.sol#444-470) ignores return value by
incToken.approve(address(pulseRouter),type()(uint256).max) (contracts/
HLAWStaking.sol#468)
HLAWStaking.constructor(uint256,address,address,address,address,address,address,address,
address) (contracts/HLAWStaking.sol#444-470) ignores return value by
daiWplsLpToken.approve(address(hlawExchange),type()(uint256).max) (contracts/
HLAWStaking.sol#469)
HLAWStaking._incentiveSwap(uint256) (contracts/HLAWStaking.sol#1232-1281) ignores
return value by (None,None,lpTokenGained) =
pulseRouter.addLiquidity(address(wplsToken),address(daiToken),wplsGained /
2,daiGained,0,0,address(this),block.timestamp) (contracts/HLAWStaking.sol#1258)
HLAWZapper.constructor() (contracts/HLAWZapper.sol#109-144) ignores return value by
IERC20(0x95B303987A60C71504D99Aa1b13B4DA07b0790ab).approve(address(pulseRouter),type()
(uint256).max) (contracts/HLAWZapper.sol#118-121)
HLAWZapper.constructor() (contracts/HLAWZapper.sol#109-144) ignores return value by
IERC20(0x2fa878Ab3F87CC1C9737Fc071108F904c0B0C95d).approve(address(pulseRouter),type()
(uint256).max) (contracts/HLAWZapper.sol#122-125)
HLAWZapper.constructor() (contracts/HLAWZapper.sol#109-144) ignores return value by
IERC20(0x02DcdD04e3F455D838cd1249292C58f3B79e3C3C).approve(address(pulseRouter),type()
(uint256).max) (contracts/HLAWZapper.sol#126-129)
HLAWZapper.constructor() (contracts/HLAWZapper.sol#109-144) ignores return value by
IERC20(0xefD766cCb38EaF1dfd701853BFCe31359239F305).approve(address(pulseRouter),type()
(uint256).max) (contracts/HLAWZapper.sol#130-133)
HLAWZapper.constructor() (contracts/HLAWZapper.sol#109-144) ignores return value by
IERC20(0x0Cb6F5a34ad42ec934882A05265A7d5F59b51A2f).approve(address(pulseRouter),type()
(uint256).max) (contracts/HLAWZapper.sol#134-137)
HLAWZapper.constructor() (contracts/HLAWZapper.sol#109-144) ignores return value by
IERC20(0x15D38573d2feeb82e7ad5187aB8c1D52810B1f07).approve(address(pulseRouter),type()
(uint256).max) (contracts/HLAWZapper.sol#138-141)
HLAWZapper.setCurr(address,bool) (contracts/HLAWZapper.sol#163-172) ignores return
value by IERC20(_curr).approve(address(pulseRouter),0) (contracts/HLAWZapper.sol#167)
HLAWZapper.setCurr(address,bool) (contracts/HLAWZapper.sol#163-172) ignores return
value by IERC20(_curr).approve(address(pulseRouter),type()(uint256).max) (contracts/
HLAWZapper.sol#170)
HLAWZapper.zapTokens(address,uint256,uint256,uint256,uint256) (contracts/
HLAWZapper.sol#184-252) ignores return value by (None,None,lpTokensGained) =
pulseRouter.addLiquidityETH{value: plsAmount}
(address(dai),daiAmount,0,0,address(msg.sender),block.timestamp) (contracts/
HLAWZapper.sol#228-230)
HLAWZapper.zapPls(uint256) (contracts/HLAWZapper.sol#254-303) ignores return value by
```

```
(None,None,lpTokensGained) = pulseRouter.addLiquidityETH{value: plsAmount}
(address(dai),daiAmount,0,0,msg.sender,block.timestamp) (contracts/
HLAWZapper.sol#280-282)
HLAWZapper._swapPls(uint256,uint256) (contracts/HLAWZapper.sol#305-325) ignores return
value by pulseRouter.swapExactETHForTokens{value: _amount}
(_amountOutMin,path,address(this),block.timestamp) (contracts/HLAWZapper.sol#317-322)
HLAWZapper._swapTokens(address,uint256,uint256,uint256,uint256) (contracts/
HLAWZapper.sol#327-389) ignores return value by pulseRouter.swapExactTokensForETH(_amoun
t,_amountOutMin1,path,address(this),block.timestamp) (contracts/HLAWZapper.sol#342-348)
HLAWZapper._swapTokens(address,uint256,uint256,uint256,uint256) (contracts/
HLAWZapper.sol#327-389) ignores return value by
pulseRouter.swapExactETHForTokens{value: plsGained / 2}
(_amountOutMin2,path2,address(this),block.timestamp) (contracts/HLAWZapper.sol#359-364)
HLAWZapper._swapTokens(address,uint256,uint256,uint256,uint256) (contracts/
HLAWZapper.sol#327-389) ignores return value by
pulseRouter.swapExactTokensForETH(_amount /
2,_amountOutMin3,path_scope_0,address(this),block.timestamp) (contracts/
HLAWZapper.sol#376-382)
HLAWZapper.getExpectedAmountOut(uint256,address,address) (contracts/
HLAWZapper.sol#391-412) ignores return value by (reserve0,reserve1,None) =
IPulsePair(pairAddress).getReserves() (contracts/HLAWZapper.sol#398-399)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
HLAWZapper.setFeeAndTreasury(uint256,address) (contracts/HLAWZapper.sol#174-182) should
emit an event for:
        - convenienceFee = _newFee (contracts/HLAWZapper.sol#179)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-
arithmetic
INFO:Detectors:
HLAWExchange.constructor(address)._hlawToken (contracts/HLAWExchange.sol#197) lacks a
zero-check on :
                - hlawToken = _hlawToken (contracts/HLAWExchange.sol#198)
HLAWPrizePool.constructor(address,address)._hlawExchange (contracts/
HLAWPrizePool.sol#63) lacks a zero-check on :
                - hlawExchange = _hlawExchange (contracts/HLAWPrizePool.sol#65)
HLAWStaking.constructor(uint256,address,address,address,address,address,address,address,
address)._rewardPool (contracts/HLAWStaking.sol#448) lacks a zero-check on :
                - rewardPool = _rewardPool (contracts/HLAWStaking.sol#459)
HLAWStaking.constructor(uint256,address,address,address,address,address,address,address,
address)._prizePool (contracts/HLAWStaking.sol#449) lacks a zero-check on :
                - prizePool = _prizePool (contracts/HLAWStaking.sol#460)
```

```
HLAWStaking.constructor(uint256,address,address,address,address,address,address,address,
address)._incRewardPool (contracts/HLAWStaking.sol#450) lacks a zero-check on :
                - incRewardPool = _incRewardPool (contracts/HLAWStaking.sol#461)
HLAWStaking.constructor(uint256,address,address,address,address,address,address,address,
address)._instantRewardPool (contracts/HLAWStaking.sol#451) lacks a zero-check on :
                - instantRewardPool = _instantRewardPool (contracts/
HLAWStaking.sol#462)
HLAWStaking.constructor(uint256,address,address,address,address,address,address,address,
address)._teamFee (contracts/HLAWStaking.sol#452) lacks a zero-check on :
                - teamFeeReceiver = _teamFee (contracts/HLAWStaking.sol#463)
HLAWStaking.constructor(uint256,address,address,address,address,address,address,address,
address)._treasuryFee (contracts/HLAWStaking.sol#453) lacks a zero-check on :
                - treasuryFeeReceiver = _treasuryFee (contracts/HLAWStaking.sol#464)
HLAWStaking.setReceivers(address,address,address)._newPrizePool (contracts/
HLAWStaking.sol#628) lacks a zero-check on :
                - prizePool = _newPrizePool (contracts/HLAWStaking.sol#636)
HeartsLaw.init(address)._hlawExchange (contracts/HLAWToken.sol#17) lacks a zero-check
on :
                - hlawExchange = _hlawExchange (contracts/HLAWToken.sol#19)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-
address-validation
INFO:Detectors:
HLAWPrizePool.distributeWinners() (contracts/HLAWPrizePool.sol#140-166) has external
calls inside a loop: amountsSubset[k] = (sessions[sessionIndex].prizePercents[k] *
hlawToken.balanceOf(address(this))) / denominator (contracts/HLAWPrizePool.sol#149)
HLAWPrizePool.distributeWinners() (contracts/HLAWPrizePool.sol#140-166) has external
calls inside a loop: hlawToken.transfer(winnersSubset[i],amountsSubset[i]) (contracts/
HLAWPrizePool.sol#157)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-
a-loop
INFO:Detectors:
Reentrancy in HLAWPrizePool._autoDistribute() (contracts/HLAWPrizePool.sol#279-302):
        External calls:
        - hlawToken.transfer(winnersSubset[i],amountsSubset[i]) (contracts/
HLAWPrizePool.sol#293)
        State variables written after the call(s):
        - lastRewardTime = block.timestamp (contracts/HLAWPrizePool.sol#298)
Reentrancy in HLAWStaking._claimIncentives(address,uint256,uint256,bool) (contracts/
HLAWStaking.sol#1070-1137):
        External calls:
        - hlawGained = _incentiveSwap(feeAmount.sub(toTeam)) (contracts/
HLAWStaking.sol#1113)
```

```
                -
pulseRouter.swapExactTokensForTokens(amount,0,path,address(this),block.timestamp)
(contracts/HLAWStaking.sol#1241)
                - pulseRouter.swapExactTokensForTokens(wplsGained /
2,0,path2,address(this),block.timestamp) (contracts/HLAWStaking.sol#1252)
                - (None,None,lpTokenGained) =
pulseRouter.addLiquidity(address(wplsToken),address(daiToken),wplsGained /
2,daiGained,0,0,address(this),block.timestamp) (contracts/HLAWStaking.sol#1258)
                - hlawExchange.buyFromFee(lpTokenGained) (contracts/
HLAWStaking.sol#1264)
        State variables written after the call(s):
        - _distributeInstantRewardDividends(toInstantReward) (contracts/
HLAWStaking.sol#1127)
                - accInstantRewardsPerShare =
accInstantRewardsPerShare.add(instantRewardDistributionAmount) (contracts/
HLAWStaking.sol#1147)
Reentrancy in HLAWPrizePool.distributeWinners() (contracts/HLAWPrizePool.sol#140-166):
        External calls:
        - hlawToken.transfer(winnersSubset[i],amountsSubset[i]) (contracts/
HLAWPrizePool.sol#157)
        State variables written after the call(s):
        - lastRewardTime = block.timestamp (contracts/HLAWPrizePool.sol#162)
Reentrancy in HLAWStaking.migrate(uint256) (contracts/HLAWStaking.sol#587-611):
        External calls:
        - migrationAddress.migrate(msg.sender,transferAmount) (contracts/
HLAWStaking.sol#599)
        State variables written after the call(s):
        - totalStakedTokens = totalStakedTokens.sub(user.amount) (contracts/
HLAWStaking.sol#602)
Reentrancy in HLAWExchange.pauseBuys(bool) (contracts/HLAWExchange.sol#568-571):
        External calls:
        - require(bool,string)(!
IHLAWPrizePool(hlawStaking.prizePool()).isSessionActive(),Can only pause buys when
there is no active prize session.) (contracts/HLAWExchange.sol#569)
        State variables written after the call(s):
        - buysPaused = _paused (contracts/HLAWExchange.sol#570)
Reentrancy in HLAWStaking.stake(uint256,uint256) (contracts/HLAWStaking.sol#478-506):
        External calls:
        - _claimIncentives(msg.sender,_pid,_amount,true) (contracts/
HLAWStaking.sol#493)
                -
```

```
pulseRouter.swapExactTokensForTokens(amount,0,path,address(this),block.timestamp)
(contracts/HLAWStaking.sol#1241)
                - pulseRouter.swapExactTokensForTokens(wplsGained /
2,0,path2,address(this),block.timestamp) (contracts/HLAWStaking.sol#1252)
                - (None,None,lpTokenGained) =
pulseRouter.addLiquidity(address(wplsToken),address(daiToken),wplsGained /
2,daiGained,0,0,address(this),block.timestamp) (contracts/HLAWStaking.sol#1258)
                - hlawExchange.buyFromFee(lpTokenGained) (contracts/
HLAWStaking.sol#1264)
        State variables written after the call(s):
        - _addUser(msg.sender) (contracts/HLAWStaking.sol#502)
                - existingUser[user] = true (contracts/HLAWStaking.sol#1284)
        - _addUser(msg.sender) (contracts/HLAWStaking.sol#502)
                - totalUsers += 1 (contracts/HLAWStaking.sol#1285)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-
vulnerabilities-2
INFO:Detectors:
Reentrancy in HLAWPrizePool._autoDistribute() (contracts/HLAWPrizePool.sol#279-302):
        External calls:
        - hlawToken.transfer(winnersSubset[i],amountsSubset[i]) (contracts/
HLAWPrizePool.sol#293)
        Event emitted after the call(s):
        - WinnersDistributed(winnersSubset,amountsSubset,sessionIndex - 1) (contracts/
HLAWPrizePool.sol#301)
Reentrancy in HLAWExchange._buyFromFee(uint256) (contracts/HLAWExchange.sol#490-501):
        External calls:
        - pulseFarm.deposit(PID,amount) (contracts/HLAWExchange.sol#494)
        - IHLAWToken(hlawToken).mint(address(this),amount) (contracts/
HLAWExchange.sol#497)
        - hlawStaking.autoStake(PID,hlawStaking.treasuryFeeReceiver(),amount)
(contracts/HLAWExchange.sol#498)
        Event emitted after the call(s):
        - HLAWBoughtFromFeeProtocol(amount) (contracts/HLAWExchange.sol#500)
Reentrancy in HLAWStaking._claimIncentives(address,uint256,uint256,bool) (contracts/
HLAWStaking.sol#1070-1137):
        External calls:
        - hlawGained = _incentiveSwap(feeAmount.sub(toTeam)) (contracts/
HLAWStaking.sol#1113)
                -
pulseRouter.swapExactTokensForTokens(amount,0,path,address(this),block.timestamp)
(contracts/HLAWStaking.sol#1241)
```

```
                - pulseRouter.swapExactTokensForTokens(wplsGained /
2,0,path2,address(this),block.timestamp) (contracts/HLAWStaking.sol#1252)
                - (None,None,lpTokenGained) =
pulseRouter.addLiquidity(address(wplsToken),address(daiToken),wplsGained /
2,daiGained,0,0,address(this),block.timestamp) (contracts/HLAWStaking.sol#1258)
                - hlawExchange.buyFromFee(lpTokenGained) (contracts/
HLAWStaking.sol#1264)
        Event emitted after the call(s):
        - InstantRewardDistributed(_claimedAmount) (contracts/HLAWStaking.sol#1149)
                - _distributeInstantRewardDividends(toInstantReward) (contracts/
HLAWStaking.sol#1127)
Reentrancy in HLAWStaking._claimIncentives(address,uint256,uint256,bool) (contracts/
HLAWStaking.sol#1070-1137):
        External calls:
        - hlawGained = _incentiveSwap(feeAmount.sub(toTeam)) (contracts/
HLAWStaking.sol#1113)
                -
pulseRouter.swapExactTokensForTokens(amount,0,path,address(this),block.timestamp)
(contracts/HLAWStaking.sol#1241)
                - pulseRouter.swapExactTokensForTokens(wplsGained /
2,0,path2,address(this),block.timestamp) (contracts/HLAWStaking.sol#1252)
                - (None,None,lpTokenGained) =
pulseRouter.addLiquidity(address(wplsToken),address(daiToken),wplsGained /
2,daiGained,0,0,address(this),block.timestamp) (contracts/HLAWStaking.sol#1258)
                - hlawExchange.buyFromFee(lpTokenGained) (contracts/
HLAWStaking.sol#1264)
        - _autoStake(1,treasuryFeeReceiver,toTreasury) (contracts/HLAWStaking.sol#1133)
                -
pulseRouter.swapExactTokensForTokens(amount,0,path,address(this),block.timestamp)
(contracts/HLAWStaking.sol#1241)
                - pulseRouter.swapExactTokensForTokens(wplsGained /
2,0,path2,address(this),block.timestamp) (contracts/HLAWStaking.sol#1252)
                - (None,None,lpTokenGained) =
pulseRouter.addLiquidity(address(wplsToken),address(daiToken),wplsGained /
2,daiGained,0,0,address(this),block.timestamp) (contracts/HLAWStaking.sol#1258)
                - hlawExchange.buyFromFee(lpTokenGained) (contracts/
HLAWStaking.sol#1264)
        Event emitted after the call(s):
        - Claim(_user,_pid,pendingAmount) (contracts/HLAWStaking.sol#1050)
                - _autoStake(1,treasuryFeeReceiver,toTreasury) (contracts/
HLAWStaking.sol#1133)
```

```
        - IncentiveClaimed(_user,_pendingIncentives) (contracts/HLAWStaking.sol#1136)
        - IncentiveClaimed(_user,_pendingIncentives) (contracts/HLAWStaking.sol#1136)
                - _autoStake(1,treasuryFeeReceiver,toTreasury) (contracts/
HLAWStaking.sol#1133)
        - InstantRewardClaimed(_user,pendingInstantReward) (contracts/
HLAWStaking.sol#1067)
                - _autoStake(1,treasuryFeeReceiver,toTreasury) (contracts/
HLAWStaking.sol#1133)
        - InstantRewardDistributed(_claimedAmount) (contracts/HLAWStaking.sol#1149)
                - _autoStake(1,treasuryFeeReceiver,toTreasury) (contracts/
HLAWStaking.sol#1133)
Reentrancy in HLAWExchange._incentiveSwap(uint256) (contracts/
HLAWExchange.sol#503-544):
        External calls:
        -
pulseRouter.swapExactTokensForTokens(amount,0,path,address(this),block.timestamp)
(contracts/HLAWExchange.sol#512)
        - pulseRouter.swapExactTokensForTokens(wplsGained /
2,0,path2,address(this),block.timestamp) (contracts/HLAWExchange.sol#523)
        - (None,None,lpTokenGained) =
pulseRouter.addLiquidity(address(wplsToken),address(daiToken),wplsGained /
2,daiGained,0,0,address(this),block.timestamp) (contracts/HLAWExchange.sol#529)
        - _buyFromFee(lpTokenGained) (contracts/HLAWExchange.sol#532)
                - pulseFarm.deposit(PID,amount) (contracts/HLAWExchange.sol#494)
                - IHLAWToken(hlawToken).mint(address(this),amount) (contracts/
HLAWExchange.sol#497)
                - hlawStaking.autoStake(PID,hlawStaking.treasuryFeeReceiver(),amount)
(contracts/HLAWExchange.sol#498)
        Event emitted after the call(s):
        - HLAWBoughtFromFeeProtocol(amount) (contracts/HLAWExchange.sol#500)
                - _buyFromFee(lpTokenGained) (contracts/HLAWExchange.sol#532)
Reentrancy in HLAWExchange.buyFromFee(uint256) (contracts/HLAWExchange.sol#441-482):
        External calls:
        - pulseFarm.deposit(PID,amount) (contracts/HLAWExchange.sol#455)
        - hlawStaking.distributeIncentiveDividends(amountToDistribute) (contracts/
HLAWExchange.sol#465)
        - _incentiveSwap(amountToTreasury) (contracts/HLAWExchange.sol#469)
                - pulseFarm.deposit(PID,amount) (contracts/HLAWExchange.sol#494)
                - IHLAWToken(hlawToken).mint(address(this),amount) (contracts/
HLAWExchange.sol#497)
                - hlawStaking.autoStake(PID,hlawStaking.treasuryFeeReceiver(),amount)
```

```
(contracts/HLAWExchange.sol#498)
                - pulseRouter.swapExactTokensForTokens(amount,0,path,address(this),block.timestamp)
(contracts/HLAWExchange.sol#512)
                - pulseRouter.swapExactTokensForTokens(wplsGained /
2,0,path2,address(this),block.timestamp) (contracts/HLAWExchange.sol#523)
                - (None,None,lpTokenGained) =
pulseRouter.addLiquidity(address(wplsToken),address(daiToken),wplsGained /
2,daiGained,0,0,address(this),block.timestamp) (contracts/HLAWExchange.sol#529)
        Event emitted after the call(s):
        - HLAWBoughtFromFeeProtocol(amount) (contracts/HLAWExchange.sol#500)
                - _incentiveSwap(amountToTreasury) (contracts/HLAWExchange.sol#469)
Reentrancy in HLAWExchange.buyFromFee(uint256) (contracts/HLAWExchange.sol#441-482):
        External calls:
        - pulseFarm.deposit(PID,amount) (contracts/HLAWExchange.sol#455)
        - hlawStaking.distributeIncentiveDividends(amountToDistribute) (contracts/
HLAWExchange.sol#465)
        - _incentiveSwap(amountToTreasury) (contracts/HLAWExchange.sol#469)
                - pulseFarm.deposit(PID,amount) (contracts/HLAWExchange.sol#494)
                - IHLAWToken(hlawToken).mint(address(this),amount) (contracts/
HLAWExchange.sol#497)
                - hlawStaking.autoStake(PID,hlawStaking.treasuryFeeReceiver(),amount)
(contracts/HLAWExchange.sol#498)
                - pulseRouter.swapExactTokensForTokens(amount,0,path,address(this),block.timestamp)
(contracts/HLAWExchange.sol#512)
                - pulseRouter.swapExactTokensForTokens(wplsGained /
2,0,path2,address(this),block.timestamp) (contracts/HLAWExchange.sol#523)
                - (None,None,lpTokenGained) =
pulseRouter.addLiquidity(address(wplsToken),address(daiToken),wplsGained /
2,daiGained,0,0,address(this),block.timestamp) (contracts/HLAWExchange.sol#529)
        - IHLAWToken(hlawToken).mint(msg.sender,amount) (contracts/
HLAWExchange.sol#479)
        Event emitted after the call(s):
        - HLAWBoughtFromFee(amount) (contracts/HLAWExchange.sol#481)
Reentrancy in HLAWBoost.distributeBoost() (contracts/HLAWBoost.sol#36-46):
        External calls:
        - hlawStaking.distributeBoost(hlawToken.balanceOf(address(this))) (contracts/
HLAWBoost.sol#43)
        Event emitted after the call(s):
        - BoostDistributed(hlawToken.balanceOf(address(this))) (contracts/
HLAWBoost.sol#45)
```

```
Reentrancy in HLAWPrizePool.distributeWinners() (contracts/HLAWPrizePool.sol#140-166):
        External calls:
        - hlawToken.transfer(winnersSubset[i],amountsSubset[i]) (contracts/
HLAWPrizePool.sol#157)
        Event emitted after the call(s):
        - WinnersDistributed(winnersSubset,amountsSubset,sessionIndex - 1) (contracts/
HLAWPrizePool.sol#165)
Reentrancy in HLAWExchange.init(address,address,address) (contracts/
HLAWExchange.sol#552-566):
        External calls:
        - IERC20(hlawToken).approve(_hlawStaking,type()(uint256).max) (contracts/
HLAWExchange.sol#561)
        - incToken.approve(address(_hlawStaking),type()(uint256).max) (contracts/
HLAWExchange.sol#562)
        Event emitted after the call(s):
        - Initialized() (contracts/HLAWExchange.sol#565)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-
vulnerabilities-3
INFO:Detectors:
HLAWZapper.zapTokens(address,uint256,uint256,uint256,uint256) (contracts/
HLAWZapper.sol#184-252) tries to limit the gas of an external call that controls
implicit decoding
        (success,None) = address(msg.sender).call{gas: 30000,value: plsRefund}()
(contracts/HLAWZapper.sol#235)
HLAWZapper.zapPls(uint256) (contracts/HLAWZapper.sol#254-303) tries to limit the gas of
an external call that controls implicit decoding
        (success,None) = address(treasury).call{gas: 30000,value: feeAmount}()
(contracts/HLAWZapper.sol#266)
        (success_scope_0,None) = address(msg.sender).call{gas: 30000,value: plsRefund}
() (contracts/HLAWZapper.sol#291)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#return-bomb
INFO:Detectors:
HLAWBoost.distributeBoost() (contracts/HLAWBoost.sol#36-46) uses timestamp for
comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp >= distributeTime,Distribute time has
not passed.) (contracts/HLAWBoost.sol#38-41)
HLAWExchange._buyFromFee(uint256) (contracts/HLAWExchange.sol#490-501) uses timestamp
for comparisons
        Dangerous comparisons:
        - require(bool,string)(amount > 0,Amount must be greater than 0) (contracts/
HLAWExchange.sol#491)
```

```
HLAWPrizePool.startSession(uint256,uint256,uint256,uint256,uint256,uint256[],uint256,boo
l) (contracts/HLAWPrizePool.sol#78-119) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(startTime >= block.timestamp,Event start must be in the
future.) (contracts/HLAWPrizePool.sol#90)
HLAWPrizePool.forceEnd(uint256,uint256) (contracts/HLAWPrizePool.sol#127-134) uses
timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(_forceEndTime - 86400 >= block.timestamp,Force end must
be atleast 24 hours in the future.) (contracts/HLAWPrizePool.sol#131)
HLAWPrizePool.addBuyer(address,uint256) (contracts/HLAWPrizePool.sol#198-233) uses
timestamp for comparisons
        Dangerous comparisons:
        - sessions[sessionIndex].ended || sessions[sessionIndex].startTime == 0 ||
sessions[sessionIndex].startTime > block.timestamp (contracts/HLAWPrizePool.sol#204)
        - block.timestamp >= sessions[sessionIndex].endsAt (contracts/
HLAWPrizePool.sol#209)
HLAWPrizePool._selectWinners() (contracts/HLAWPrizePool.sol#248-258) uses timestamp for
comparisons
        Dangerous comparisons:
        - buyers[i].buyer != address(0) (contracts/HLAWPrizePool.sol#252)
HLAWStaking.stake(uint256,uint256) (contracts/HLAWStaking.sol#478-506) uses timestamp
for comparisons
        Dangerous comparisons:
        - pendingRewards(_pid,msg.sender) > 0 (contracts/HLAWStaking.sol#487)
        - autoUpdate && lastUpdateTime + updateInterval <= block.timestamp (contracts/
HLAWStaking.sol#497)
HLAWStaking.unstake(uint256,uint256) (contracts/HLAWStaking.sol#508-533) uses timestamp
for comparisons
        Dangerous comparisons:
        - pendingRewards(_pid,msg.sender) > 0 (contracts/HLAWStaking.sol#519)
        - autoUpdate && lastUpdateTime + updateInterval <= block.timestamp (contracts/
HLAWStaking.sol#528)
HLAWStaking.exit(uint256) (contracts/HLAWStaking.sol#535-555) uses timestamp for
comparisons
        Dangerous comparisons:
        - autoUpdate && lastUpdateTime + updateInterval <= block.timestamp (contracts/
HLAWStaking.sol#550)
HLAWStaking.claim(uint256) (contracts/HLAWStaking.sol#557-569) uses timestamp for
comparisons
        Dangerous comparisons:
```

```
        - pendingRewards(_pid,msg.sender) > 0 (contracts/HLAWStaking.sol#559)
        - autoUpdate && lastUpdateTime + updateInterval <= block.timestamp (contracts/
HLAWStaking.sol#566)
HLAWStaking.claimIncentives() (contracts/HLAWStaking.sol#571-577) uses timestamp for
comparisons
        Dangerous comparisons:
        - autoUpdate && lastUpdateTime + updateInterval <= block.timestamp (contracts/
HLAWStaking.sol#574)
HLAWStaking.claimInstantRewards() (contracts/HLAWStaking.sol#579-585) uses timestamp
for comparisons
        Dangerous comparisons:
        - autoUpdate && lastUpdateTime + updateInterval <= block.timestamp (contracts/
HLAWStaking.sol#582)
HLAWStaking.autoStake(uint256,address,uint256) (contracts/HLAWStaking.sol#766-792) uses
timestamp for comparisons
        Dangerous comparisons:
        - pendingRewards(_pid,_user) > 0 (contracts/HLAWStaking.sol#779)
        - autoUpdate && lastUpdateTime + updateInterval <= block.timestamp (contracts/
HLAWStaking.sol#789)
HLAWStaking.pendingRewards(uint256,address) (contracts/HLAWStaking.sol#869-897) uses
timestamp for comparisons
        Dangerous comparisons:
        - user.amount == 0 || block.timestamp < user.lastRewardTime || pool.totalStaked
== 0 (contracts/HLAWStaking.sol#877-879)
HLAWStaking._claim(uint256,address) (contracts/HLAWStaking.sol#1032-1051) uses
timestamp for comparisons
        Dangerous comparisons:
        - pendingAmount == 0 (contracts/HLAWStaking.sol#1038)
HLAWStaking._autoStake(uint256,address,uint256) (contracts/HLAWStaking.sol#1152-1172)
uses timestamp for comparisons
        Dangerous comparisons:
        - pendingRewards(_pid,_user) > 0 (contracts/HLAWStaking.sol#1161)
        - autoUpdate && lastUpdateTime + updateInterval <= block.timestamp (contracts/
HLAWStaking.sol#1169)
HLAWStaking.safeRewardTransfer(uint256,address,uint256) (contracts/
HLAWStaking.sol#1174-1187) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(_amount <= rewardBal,You must wait for the rewardPool
balance to be replenished.) (contracts/HLAWStaking.sol#1181-1184)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-
timestamp
```

```
INFO:Detectors:
HLAWExchange.buy(uint256,address,bool) (contracts/HLAWExchange.sol#225-344) compares to
a boolean constant:
        -hlawStaking.existingUser(msg.sender) == false (contracts/HLAWExchange.sol#233)
HLAWPrizePool.distributeWinners() (contracts/HLAWPrizePool.sol#140-166) compares to a
boolean constant:
        -require(bool,string)(sessions[sessionIndex].ended == true,The session has not
ended yet.) (contracts/HLAWPrizePool.sol#142)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-
equality
INFO:Detectors:
HLAWPrizePool.distributeWinners() (contracts/HLAWPrizePool.sol#140-166) has costly
operations inside a loop:
        - rewardsPaid += amountsSubset[i] (contracts/HLAWPrizePool.sol#158)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#costly-
operations-inside-a-loop
INFO:Detectors:
HLAWExchange.buy(uint256,address,bool) (contracts/HLAWExchange.sol#225-344) has a high
cyclomatic complexity (22).
HLAWExchange.sell(uint256) (contracts/HLAWExchange.sol#346-433) has a high cyclomatic
complexity (14).
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#cyclomatic-
complexity
INFO:Detectors:
Version constraint 0.8.20 contains known severe issues (https://solidity.readthedocs.io/
en/latest/bugs.html)
        - VerbatimInvalidDeduplication
        - FullInlinerNonExpressionSplitArgumentEvaluationOrder
        - MissingSideEffectsOnSelectorAccess.
It is used by:
        - 0.8.20 (contracts/HLAWBoost.sol#2)
        - 0.8.20 (contracts/HLAWExchange.sol#2)
        - 0.8.20 (contracts/HLAWIncentiveRewardPool.sol#2)
        - 0.8.20 (contracts/HLAWInstantRewardPool.sol#2)
        - 0.8.20 (contracts/HLAWPrizePool.sol#2)
        - 0.8.20 (contracts/HLAWRewardPool.sol#2)
        - 0.8.20 (contracts/HLAWStaking.sol#2)
        - 0.8.20 (contracts/HLAWToken.sol#2)
        - 0.8.20 (contracts/HLAWZapper.sol#2)
Version constraint ^0.8.0 contains known severe issues (https://solidity.readthedocs.io/
en/latest/bugs.html)
```

```
        - FullInlinerNonExpressionSplitArgumentEvaluationOrder
        - MissingSideEffectsOnSelectorAccess
        - AbiReencodingHeadOverflowWithStaticArrayCleanup
        - DirtyBytesArrayToStorage
        - DataLocationChangeInInternalOverride
        - NestedCalldataArrayAbiReencodingSizeValidation
        - SignedImmutables
        - ABIDecodeTwoDimensionalArrayMemory
        - KeccakCaching.
It is used by:
        - ^0.8.0 (contracts/HLAWReferral.sol#2)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-
versions-of-solidity
INFO:Detectors:
Low level call in HLAWZapper.zapTokens(address,uint256,uint256,uint256,uint256)
(contracts/HLAWZapper.sol#184-252):
        - (success,None) = address(msg.sender).call{gas: 30000,value: plsRefund}()
(contracts/HLAWZapper.sol#235)
Low level call in HLAWZapper.zapPls(uint256) (contracts/HLAWZapper.sol#254-303):
        - (success,None) = address(treasury).call{gas: 30000,value: feeAmount}()
(contracts/HLAWZapper.sol#266)
        - (success_scope_0,None) = address(msg.sender).call{gas: 30000,value: plsRefund}
() (contracts/HLAWZapper.sol#291)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-
calls
INFO:Detectors:
HLAWExchange (contracts/HLAWExchange.sol#129-638) should inherit from IHLAWExchange
(contracts/HLAWStaking.sol#253-255)
HLAWReferral (contracts/HLAWReferral.sol#9-126) should inherit from IHLAWReferral
(contracts/HLAWExchange.sol#121-127)
HLAWStaking (contracts/HLAWStaking.sol#307-1294) should inherit from IHLAWStaking
(contracts/HLAWBoost.sol#7-9)
HeartsLaw (contracts/HLAWToken.sol#7-65) should inherit from IHLAWToken (contracts/
HLAWExchange.sol#8-12)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-
inheritance
INFO:Detectors:
Parameter HLAWBoost.setSigner(address)._signer (contracts/HLAWBoost.sol#52) is not in
mixedCase
Parameter HLAWBoost.setDistributeTime(uint256,uint256)._startTime (contracts/
HLAWBoost.sol#63) is not in mixedCase
```

```
Parameter HLAWBoost.setDistributeTime(uint256,uint256)._distributeTime (contracts/
HLAWBoost.sol#64) is not in mixedCase
Parameter HLAWExchange.init(address,address,address)._hlawStaking (contracts/
HLAWExchange.sol#552) is not in mixedCase
Parameter HLAWExchange.init(address,address,address)._hlawReferral (contracts/
HLAWExchange.sol#552) is not in mixedCase
Parameter HLAWExchange.init(address,address,address)._rewardPool (contracts/
HLAWExchange.sol#552) is not in mixedCase
Parameter HLAWExchange.pauseBuys(bool)._paused (contracts/HLAWExchange.sol#568) is not
in mixedCase
Parameter HLAWExchange.setFees(bool,uint256,uint256,uint256,uint256,uint256)._isSell
(contracts/HLAWExchange.sol#574) is not in mixedCase
Parameter
HLAWExchange.setFees(bool,uint256,uint256,uint256,uint256,uint256)._rewardPoolFee
(contracts/HLAWExchange.sol#575) is not in mixedCase
Parameter
HLAWExchange.setFees(bool,uint256,uint256,uint256,uint256,uint256)._instantRewardFee
(contracts/HLAWExchange.sol#576) is not in mixedCase
Parameter HLAWExchange.setFees(bool,uint256,uint256,uint256,uint256,uint256)._teamFee
(contracts/HLAWExchange.sol#577) is not in mixedCase
Parameter
HLAWExchange.setFees(bool,uint256,uint256,uint256,uint256,uint256)._prizePoolFee
(contracts/HLAWExchange.sol#578) is not in mixedCase
Parameter
HLAWExchange.setFees(bool,uint256,uint256,uint256,uint256,uint256)._referralFee
(contracts/HLAWExchange.sol#579) is not in mixedCase
Parameter HLAWExchange.setGlobalFees(uint256,uint256,uint256)._totalBuyFee (contracts/
HLAWExchange.sol#607) is not in mixedCase
Parameter HLAWExchange.setGlobalFees(uint256,uint256,uint256)._totalSellFee (contracts/
HLAWExchange.sol#608) is not in mixedCase
Parameter HLAWExchange.setGlobalFees(uint256,uint256,uint256)._treasuryIncFee
(contracts/HLAWExchange.sol#609) is not in mixedCase
Parameter HLAWExchange.updateRefRequirement(uint256)._newRefRequirement (contracts/
HLAWExchange.sol#622) is not in mixedCase
Parameter HLAWExchange.updateMaxRefRequirement(uint256)._newMaximum (contracts/
HLAWExchange.sol#633) is not in mixedCase
Constant HLAWExchange.daiWplsLpToken (contracts/HLAWExchange.sol#148-149) is not in
UPPER_CASE_WITH_UNDERSCORES
Constant HLAWExchange.wplsToken (contracts/HLAWExchange.sol#150-151) is not in
UPPER_CASE_WITH_UNDERSCORES
Constant HLAWExchange.incToken (contracts/HLAWExchange.sol#152-153) is not in
UPPER_CASE_WITH_UNDERSCORES
```

```
Constant HLAWExchange.daiToken (contracts/HLAWExchange.sol#154-155) is not in
UPPER_CASE_WITH_UNDERSCORES
Constant HLAWExchange.pulseRouter (contracts/HLAWExchange.sol#156-157) is not in
UPPER_CASE_WITH_UNDERSCORES
Constant HLAWExchange.pulseFarm (contracts/HLAWExchange.sol#158-159) is not in
UPPER_CASE_WITH_UNDERSCORES
Parameter HLAWIncentiveRewardPool.init(address)._hlawStaking (contracts/
HLAWIncentiveRewardPool.sol#19) is not in mixedCase
Parameter HLAWInstantRewardPool.init(address)._hlawStaking (contracts/
HLAWInstantRewardPool.sol#20) is not in mixedCase
Parameter HLAWPrizePool.forceEnd(uint256,uint256)._sessionId (contracts/
HLAWPrizePool.sol#128) is not in mixedCase
Parameter HLAWPrizePool.forceEnd(uint256,uint256)._forceEndTime (contracts/
HLAWPrizePool.sol#129) is not in mixedCase
Parameter HLAWPrizePool.setSigner(address)._signer (contracts/HLAWPrizePool.sol#173) is
not in mixedCase
Parameter HLAWPrizePool.setDistributeLimits(uint256)._newMaxPercent (contracts/
HLAWPrizePool.sol#185) is not in mixedCase
Parameter HLAWPrizePool.getSessionInfo(uint256)._sessionIndex (contracts/
HLAWPrizePool.sol#350) is not in mixedCase
Constant HLAWPrizePool.denominator (contracts/HLAWPrizePool.sol#46) is not in
UPPER_CASE_WITH_UNDERSCORES
Parameter HLAWReferral.init(address)._hlawExchange (contracts/HLAWReferral.sol#49) is
not in mixedCase
Parameter HLAWRewardPool.init(address)._hlawStaking (contracts/HLAWRewardPool.sol#20)
is not in mixedCase
Parameter HLAWStaking.stake(uint256,uint256)._pid (contracts/HLAWStaking.sol#478) is
not in mixedCase
Parameter HLAWStaking.stake(uint256,uint256)._amount (contracts/HLAWStaking.sol#478) is
not in mixedCase
Parameter HLAWStaking.unstake(uint256,uint256)._pid (contracts/HLAWStaking.sol#508) is
not in mixedCase
Parameter HLAWStaking.unstake(uint256,uint256)._amount (contracts/HLAWStaking.sol#508)
is not in mixedCase
Parameter HLAWStaking.exit(uint256)._pid (contracts/HLAWStaking.sol#535) is not in
mixedCase
Parameter HLAWStaking.claim(uint256)._pid (contracts/HLAWStaking.sol#557) is not in
mixedCase
Parameter HLAWStaking.migrate(uint256)._pid (contracts/HLAWStaking.sol#587) is not in
mixedCase
Parameter HLAWStaking.setBoostContract(address)._newBoostContract (contracts/
```

```
HLAWStaking.sol#619) is not in mixedCase
Parameter HLAWStaking.setReceivers(address,address,address)._newTeam (contracts/
HLAWStaking.sol#626) is not in mixedCase
Parameter HLAWStaking.setReceivers(address,address,address)._newTreasury (contracts/
HLAWStaking.sol#627) is not in mixedCase
Parameter HLAWStaking.setReceivers(address,address,address)._newPrizePool (contracts/
HLAWStaking.sol#628) is not in mixedCase
Parameter
HLAWStaking.setFees(uint256,uint256,uint256,uint256,uint256,uint256)._incClaimFee
(contracts/HLAWStaking.sol#641) is not in mixedCase
Parameter
HLAWStaking.setFees(uint256,uint256,uint256,uint256,uint256,uint256)._rewardPoolFee
(contracts/HLAWStaking.sol#642) is not in mixedCase
Parameter
HLAWStaking.setFees(uint256,uint256,uint256,uint256,uint256,uint256)._instantRewardFee
(contracts/HLAWStaking.sol#643) is not in mixedCase
Parameter
HLAWStaking.setFees(uint256,uint256,uint256,uint256,uint256,uint256)._prizePoolFee
(contracts/HLAWStaking.sol#644) is not in mixedCase
Parameter HLAWStaking.setFees(uint256,uint256,uint256,uint256,uint256,uint256)._teamFee
(contracts/HLAWStaking.sol#645) is not in mixedCase
Parameter
HLAWStaking.setFees(uint256,uint256,uint256,uint256,uint256,uint256)._treasuryFee
(contracts/HLAWStaking.sol#646) is not in mixedCase
Parameter HLAWStaking.setUpdateSettings(uint256,uint256,uint256,bool)._updateInterval
(contracts/HLAWStaking.sol#676) is not in mixedCase
Parameter HLAWStaking.setUpdateSettings(uint256,uint256,uint256,bool)._updateRatio
(contracts/HLAWStaking.sol#677) is not in mixedCase
Parameter HLAWStaking.setUpdateSettings(uint256,uint256,uint256,bool)._updateThreshold
(contracts/HLAWStaking.sol#678) is not in mixedCase
Parameter HLAWStaking.setUpdateSettings(uint256,uint256,uint256,bool)._autoUpdate
(contracts/HLAWStaking.sol#679) is not in mixedCase
Parameter HLAWStaking.setSigner(address)._newSigner (contracts/HLAWStaking.sol#699) is
not in mixedCase
Parameter HLAWStaking.setMigration(bool)._migrationEnabled (contracts/
HLAWStaking.sol#705) is not in mixedCase
Parameter HLAWStaking.initializePool(IERC20,IERC20,address,uint256)._stakingToken
(contracts/HLAWStaking.sol#710) is not in mixedCase
Parameter HLAWStaking.initializePool(IERC20,IERC20,address,uint256)._dripToken
(contracts/HLAWStaking.sol#711) is not in mixedCase
Parameter HLAWStaking.initializePool(IERC20,IERC20,address,uint256)._rewardsPool
```

```
(contracts/HLAWStaking.sol#712) is not in mixedCase
Parameter HLAWStaking.initializePool(IERC20,IERC20,address,uint256)._allocPoint
(contracts/HLAWStaking.sol#713) is not in mixedCase
Parameter HLAWStaking.set(uint256,uint256,bool)._pid (contracts/HLAWStaking.sol#746) is
not in mixedCase
Parameter HLAWStaking.set(uint256,uint256,bool)._allocPoint (contracts/
HLAWStaking.sol#747) is not in mixedCase
Parameter HLAWStaking.set(uint256,uint256,bool)._active (contracts/HLAWStaking.sol#748)
is not in mixedCase
Parameter HLAWStaking.autoStake(uint256,address,uint256)._pid (contracts/
HLAWStaking.sol#766) is not in mixedCase
Parameter HLAWStaking.autoStake(uint256,address,uint256)._user (contracts/
HLAWStaking.sol#766) is not in mixedCase
Parameter HLAWStaking.autoStake(uint256,address,uint256)._amount (contracts/
HLAWStaking.sol#766) is not in mixedCase
Parameter HLAWStaking.distributeInstantRewardDividends(uint256)._amount (contracts/
HLAWStaking.sol#794) is not in mixedCase
Parameter HLAWStaking.distributeIncentiveDividends(uint256)._amount (contracts/
HLAWStaking.sol#817) is not in mixedCase
Parameter HLAWStaking.distributeBoost(uint256)._amount (contracts/HLAWStaking.sol#836)
is not in mixedCase
Parameter HLAWStaking.pendingRewards(uint256,address)._pid (contracts/
HLAWStaking.sol#870) is not in mixedCase
Parameter HLAWStaking.pendingRewards(uint256,address)._user (contracts/
HLAWStaking.sol#871) is not in mixedCase
Parameter HLAWStaking.pendingIncentives(address)._user (contracts/HLAWStaking.sol#899)
is not in mixedCase
Parameter HLAWStaking.pendingInstantRewards(address)._user (contracts/
HLAWStaking.sol#913) is not in mixedCase
Parameter HLAWStaking.getUserInfo(uint256,address)._pid (contracts/HLAWStaking.sol#932)
is not in mixedCase
Parameter HLAWStaking.getUserInfo(uint256,address)._account (contracts/
HLAWStaking.sol#933) is not in mixedCase
Parameter HLAWStaking.getPidInfo(uint256)._pid (contracts/HLAWStaking.sol#954) is not
in mixedCase
Parameter HLAWStaking.getIncentiveInfo(address)._user (contracts/HLAWStaking.sol#983)
is not in mixedCase
Parameter HLAWStaking.getInstantRewardInfo(address)._user (contracts/
HLAWStaking.sol#996) is not in mixedCase
Parameter HLAWStaking.safeRewardTransfer(uint256,address,uint256)._pid (contracts/
HLAWStaking.sol#1175) is not in mixedCase
```

```
Parameter HLAWStaking.safeRewardTransfer(uint256,address,uint256)._to (contracts/
HLAWStaking.sol#1176) is not in mixedCase
Parameter HLAWStaking.safeRewardTransfer(uint256,address,uint256)._amount (contracts/
HLAWStaking.sol#1177) is not in mixedCase
Parameter HLAWStaking.calcAccRewardsPerShare(uint256)._pid (contracts/
HLAWStaking.sol#1189) is not in mixedCase
Parameter HLAWStaking.getTimeDiff(uint256,uint256)._from (contracts/
HLAWStaking.sol#1289) is not in mixedCase
Parameter HLAWStaking.getTimeDiff(uint256,uint256)._to (contracts/HLAWStaking.sol#1290)
is not in mixedCase
Constant HLAWStaking.incToken (contracts/HLAWStaking.sol#341-342) is not in
UPPER_CASE_WITH_UNDERSCORES
Constant HLAWStaking.wplsToken (contracts/HLAWStaking.sol#343-344) is not in
UPPER_CASE_WITH_UNDERSCORES
Constant HLAWStaking.daiToken (contracts/HLAWStaking.sol#345-346) is not in
UPPER_CASE_WITH_UNDERSCORES
Constant HLAWStaking.daiWplsLpToken (contracts/HLAWStaking.sol#347-348) is not in
UPPER_CASE_WITH_UNDERSCORES
Constant HLAWStaking.pulseRouter (contracts/HLAWStaking.sol#349-350) is not in
UPPER_CASE_WITH_UNDERSCORES
Parameter HeartsLaw.init(address)._hlawExchange (contracts/HLAWToken.sol#17) is not in
mixedCase
Parameter HeartsLaw.withdrawTokens(address)._token (contracts/HLAWToken.sol#58) is not
in mixedCase
Parameter HLAWZapper.withdrawTokens(address,uint256)._tokenAddress (contracts/
HLAWZapper.sol#152) is not in mixedCase
Parameter HLAWZapper.withdrawTokens(address,uint256)._amount (contracts/
HLAWZapper.sol#153) is not in mixedCase
Parameter HLAWZapper.setCurr(address,bool)._curr (contracts/HLAWZapper.sol#163) is not
in mixedCase
Parameter HLAWZapper.setCurr(address,bool)._allowed (contracts/HLAWZapper.sol#163) is
not in mixedCase
Parameter HLAWZapper.setFeeAndTreasury(uint256,address)._newFee (contracts/
HLAWZapper.sol#175) is not in mixedCase
Parameter HLAWZapper.setFeeAndTreasury(uint256,address)._treasury (contracts/
HLAWZapper.sol#176) is not in mixedCase
Parameter HLAWZapper.zapTokens(address,uint256,uint256,uint256,uint256)._curr
(contracts/HLAWZapper.sol#185) is not in mixedCase
Parameter HLAWZapper.zapTokens(address,uint256,uint256,uint256,uint256)._currAmount
(contracts/HLAWZapper.sol#186) is not in mixedCase
Parameter HLAWZapper.zapTokens(address,uint256,uint256,uint256,uint256)._amountOutMin1
```

```
(contracts/HLAWZapper.sol#187) is not in mixedCase
Parameter HLAWZapper.zapTokens(address,uint256,uint256,uint256,uint256)._amountOutMin2
(contracts/HLAWZapper.sol#188) is not in mixedCase
Parameter HLAWZapper.zapTokens(address,uint256,uint256,uint256,uint256)._amountOutMin3
(contracts/HLAWZapper.sol#189) is not in mixedCase
Parameter HLAWZapper.zapPls(uint256)._amountOutMin (contracts/HLAWZapper.sol#255) is
not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-
solidity-naming-conventions
INFO:Detectors:
Redundant expression "success (contracts/HLAWZapper.sol#236)" inHLAWZapper (contracts/
HLAWZapper.sol#90-422)
Redundant expression "success (contracts/HLAWZapper.sol#267)" inHLAWZapper (contracts/
HLAWZapper.sol#90-422)
Redundant expression "success_scope_0 (contracts/HLAWZapper.sol#292)" inHLAWZapper
(contracts/HLAWZapper.sol#90-422)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-
statements
INFO:Detectors:
HLAWExchange.constructor(address) (contracts/HLAWExchange.sol#197-217) uses literals
with too many digits:
        - refStakeRequired = 10000000000000000000 (contracts/HLAWExchange.sol#205)
HLAWExchange.slitherConstructorVariables() (contracts/HLAWExchange.sol#129-638) uses
literals with too many digits:
        - refMax = 1000000e18 (contracts/HLAWExchange.sol#166)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-
digits
INFO:Detectors:
HLAWExchange.totalUsers (contracts/HLAWExchange.sol#167) should be constant
HLAWStaking.migrationAddress (contracts/HLAWStaking.sol#362) should be constant
HLAWZapper.dai (contracts/HLAWZapper.sol#93) should be constant
HLAWZapper.denominator (contracts/HLAWZapper.sol#96) should be constant
HLAWZapper.pulseRouter (contracts/HLAWZapper.sol#91-92) should be constant
HLAWZapper.wpls (contracts/HLAWZapper.sol#94) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-
variables-that-could-be-declared-constant
INFO:Detectors:
HLAWBoost.hlawStaking (contracts/HLAWBoost.sol#13) should be immutable
HLAWBoost.hlawToken (contracts/HLAWBoost.sol#12) should be immutable
HLAWInstantRewardPool.hlawToken (contracts/HLAWInstantRewardPool.sol#12) should be
immutable
```

```
HLAWPrizePool.hlawExchange (contracts/HLAWPrizePool.sol#39) should be immutable
HLAWPrizePool.hlawToken (contracts/HLAWPrizePool.sol#38) should be immutable
HLAWReferral.hlawToken (contracts/HLAWReferral.sol#27) should be immutable
HLAWRewardPool.hlawToken (contracts/HLAWRewardPool.sol#12) should be immutable
HLAWStaking.hlawExchange (contracts/HLAWStaking.sol#340) should be immutable
HLAWStaking.hlawToken (contracts/HLAWStaking.sol#339) should be immutable
HLAWStaking.incRewardPool (contracts/HLAWStaking.sol#354) should be immutable
HLAWStaking.instantRewardPool (contracts/HLAWStaking.sol#355) should be immutable
HLAWStaking.rewardPool (contracts/HLAWStaking.sol#352) should be immutable
HLAWStaking.startTime (contracts/HLAWStaking.sol#360) should be immutable
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-
variables-that-could-be-declared-immutable
INFO:Slither:. analyzed (44 contracts with 93 detectors), 288 result(s) found
```

0x Guard