



Smart contracts security assessment

Final report

[Tariff: Standard](#)

Pulse Farm

November 2023



0xguard.com



hello@0xguard.com

Contents

1. Introduction	3
2. Contracts checked	4
3. Procedure	4
4. Known vulnerabilities checked	4
5. Classification of issue severity	6
6. Issues	6
7. Conclusion	12
8. Disclaimer	13
9. Slither output	14

Introduction

The report has been prepared for **Pulse Farm**.

The project consists of:

- a project CowTip token. ERC20 mintable and burnable token, which may have a commission on selling;
- contracts for token distribution (automatic and manual);
- contracts for staking and farming;

The code is available at the GitHub [repository](#) and was audited after the commit [eee13d7a22a24a8ded1e88bef48895e0a59dde30](#).

The audit scope includes the following contracts:

yieldBarn.sol

cowTipFarm.sol

cowTip.sol

Report Update.

The contract's code was updated according to this report and rechecked after the commit [a72b13868c366a748209c13afb87a7ca2241e54b](#).

Name	Pulse Farm
Audit date	2023-11-25 - 2023-11-27
Language	Solidity
Platform	Pulse Chain

Contracts checked

Name	Address
------	---------

YieldBarn	
-----------	--

CowTip	
--------	--

CowTipFarm	
------------	--

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
-------	--------------

<u>Unencrypted Private Data On-Chain</u>	passed
--	--------

<u>Code With No Effects</u>	passed
-----------------------------	--------

<u>Message call with hardcoded gas amount</u>	passed
---	--------

<u>Typographical Error</u>	passed
----------------------------	--------

<u>DoS With Block Gas Limit</u>	passed
---------------------------------	--------

<u>Presence of unused variables</u>	passed
<u>Incorrect Inheritance Order</u>	passed
<u>Requirement Violation</u>	passed
<u>Weak Sources of Randomness from Chain Attributes</u>	passed
<u>Shadowing State Variables</u>	passed
<u>Incorrect Constructor Name</u>	passed
<u>Block values as a proxy for time</u>	passed
<u>Authorization through tx.origin</u>	passed
<u>DoS with Failed Call</u>	passed
<u>Delegatecall to Untrusted Callee</u>	passed
<u>Use of Deprecated Solidity Functions</u>	passed
<u>Assert Violation</u>	passed
<u>State Variable Default Visibility</u>	passed
<u>Reentrancy</u>	passed
<u>Unprotected SELFDESTRUCT Instruction</u>	passed
<u>Unprotected Ether Withdrawal</u>	passed
<u>Unchecked Call Return Value</u>	passed
<u>Floating Pragma</u>	passed
<u>Outdated Compiler Version</u>	passed
<u>Integer Overflow and Underflow</u>	passed
<u>Function Default Visibility</u>	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. Unable to endStake() without rewards (YieldBarn)

Status: Fixed

The `stopReward()` function allows the administrator to withdraw all rewards from the contract. After executing this function, users will not be able to close their stakes, since there will not be enough rewards in the contract, and the `require` statement on L174 will block all withdrawals.

Recommendation: We recommend:

- 1) adding the `emergencyEndStake()` function, which will allow to close stakes without taking into account rewards (ans simply resetting the rewards to zero).
- 2) consider the possibility of paying the available balance of LP tokens, instead of the `require` statements on L174, L211.

2. The owner of the contract may set unlimited minting rate (CowTipFarm)

Status: Fixed

The smart contract currently allows the contract owner to set the minting rate. This functionality, while providing flexibility, has been identified to lack sufficient safeguards, potentially enabling the contract owner to set an unlimited minting rate. This lack of constraints poses significant risks, as it grants the contract owner the ability to inflate the token supply without bounds.

Potential risks are:

1. Risk of Hyperinflation: Unrestricted minting could lead to a rapid increase in token supply, causing hyperinflation and devaluing the token.
2. Tokenomics Instability: Such unlimited minting undermines the token's economic model, leading to instability and loss of investor confidence.
3. Security and Trust Concerns: The potential for misuse raises security concerns and could diminish trust among users and investors in the integrity of the contract.
4. Market Manipulation: The ability to arbitrarily inflate the token supply could be exploited for market manipulation, adversely affecting the ecosystem's fairness.

Recommendation: To mitigate these risks, it is strongly recommended to implement a hard cap or a maximum limit on the minting rate that can be set by the contract owner.

Medium severity issues

1. Tokens for referral program are not minted (CowTipFarm)

Status: Fixed

The smart contract mints and distributes tokens to users as part of its functionality. However, the contract not only distributes the minted tokens but also erroneously allocates additional tokens for the referral program, which were not minted.

```
function updatePool(uint256 _pid) public {  
    PoolInfo storage pool = poolInfo[_pid];  
    ...  
}
```

```

    if (totalAllocPoint > 0) {
        uint256 multiplier = getMultiplier(
            pool.lastRewardTime,
            block.timestamp
        );
        uint256 _generatedReward = multiplier.mul(sharesPerSecond);
        uint256 _cowTipReward = _generatedReward
            .mul(pool.allocPoint)
            .div(totalAllocPoint);
        uint256 lpPercent = 10000 - devPercent - feePercent;
        cowTip.mint(
            devAddress,
            _cowTipReward.mul(devPercent).div(10000)
        );
        cowTip.mint(
            feeAddress,
            _cowTipReward.mul(feePercent).div(10000)
        );
        cowTip.mint(
            address(this),
            _cowTipReward.mul(lpPercent).div(10000)
        );

        pool.accTokensPerShare = pool.accTokensPerShare.add(
            _cowTipReward.mul(1e18).div(tokenSupply).mul(lpPercent).div(
                10000
            )
        );
    }
    pool.lastRewardTime = block.timestamp;
}

```

This leads to a situation when some users who claim last won't get their tokens as there won't be enough for the distribution.

```

function _deposit(
    uint256 _pid,
    uint256 _amount,
    address _referrer,

```



```

        address _staker
    ) private {
        ...
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][_staker];
        updatePool(_pid);
        if (user.amount > 0) {
            uint256 _pending = user
                .amount
                .mul(pool.accTokensPerShare)
                .div(1e18)
                .sub(user.rewardDebt);
            if (_pending > 0) {
                uint256 referralAmount = ((_pending) * referralRate) / 10000;
                if (referralAmount > 0) {
                    referralEarned[_referrer] =
                        referralEarned[_referrer] +
                        referralAmount;
                    safeCowTipTransfer(_referrer, referralAmount); //@audit not minted
                }
                safeCowTipTransfer(_staker, _pending);
                emit RewardPaid(_staker, _pending);
            }
        }
        ...
    }

```

Recommendation: Mint tokens for the referral program as well.

Low severity issues

1. Constructor parameters validation (YieldBarn)

Status: Fixed

We recommend adding non-zero address validation for the `_COWTIP`, `_LP` constructor parameters.

2. Gas optimization (YieldBarn)

Status: Fixed

Visibility of the functions `stopReward()`, `deposit()` can be declared as `'external'` to save gas.

3. Gas optimization (CowTip)

Status: Fixed

Visibility of the functions `setTaxRate()`, `setTaxManager()`, `setCommunityFund()`, `isAddressExcluded()`, `setLP()`, `setSwap()`, `excludeAddress()`, `includeAddress()` can be declared as `'external'` to save gas.

4. The function `deposit()` always require passing non-zero referrer address parameter (CowTipFarm)

Status: Fixed

The function `deposit()` always require passing a non-zero referrer address parameter even if the referrer has already been set.

```
function deposit(
    uint256 _pid,
    uint256 _amount,
    address _referrer
) external nonReentrant {
    address staker = _msgSender();
    _deposit(_pid, _amount, _referrer, staker);
}

function _deposit(
    uint256 _pid,
    uint256 _amount,
    address _referrer,
    address _staker
) private {
    require(
        _referrer != address(0) &&
        _referrer != _staker &&
        _referrer != address(this),
        "CowTipFarm: Invalid referrer"
    );
}
```

Recommendation: Check first if a referrer for the staker is already set. If so, allow passing zero address as a referrer.

5. Function `massUpdatePools` may exceed block gas limit (CowTipFarm)

Status: Fixed

The function `massUpdatePools` iterates through each pool in the contract's array of pools to perform necessary updates. If the contract accumulates a very large number of pools, the `massUpdatePools()` function, when called from within other functions like `set()` or `setEmissionRate()`, could require more gas than the block gas limit allows. This would lead to failed transactions and could render these functions unusable.

Recommendation: Monitor gas usage before adding new pools.

Update: The recommendation was taken into account by the dev-team.

6. Wrong require message text (CowTipFarm)

Status: Partially fixed

The function `setDevPercent()` has an erroneous require message:

```
function setDevPercent(uint256 _devPercent) external onlyOwner {
    require(
        _devPercent <= 1000,
        "CowTipFarm: Zero address not allowed" //@audit wrong message
    );
    devPercent = _devPercent;
}
```

Update: In the updated code, the message in the `setFeePercent()` function has been corrected. But at the same time, functions `setDevAddress()` and `setDevPercent()` still have the wrong messages.

Conclusion

Pulse Farm YieldBarn, CowTip, CowTipFarm contracts were audited. 2 high, 1 medium, 6 low severity issues were found.

2 high, 1 medium, 5 low severity issues have been fixed in the update.

We strongly recommend writing unit tests to have extensive coverage of the codebase minimize the possibility of bugs and ensure that everything works as expected.

The contracts are dependent on the owner's account. Users interacting with the contracts have to trust the owner.

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 OxGuard 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 OxGuard 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.

OxGuard retains exclusive publishing rights for the results of this audit on its website and social networks.

Slither output

INFO:Detectors:

CommunityFund.sendCustomTransaction(address,uint256,string,bytes) (contracts/CommunityFund.sol#44-68) sends eth to arbitrary user

Dangerous calls:

- (success,returnData) = target.call{value: value}(callData) (contracts/CommunityFund.sol#60-62)

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to-arbitrary-destinations>

INFO:Detectors:

PreSale.getRandomPlayer() (contracts/presale.sol#187-192) uses a weak PRNG: "index = random % playerIndex.length() (contracts/presale.sol#190)"

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#weak-PRNG>

INFO:Detectors:

Distributor.distribute(address,address[],uint256) (contracts/distributor.sol#51-69) ignores return value by erc20token.transfer(_receivers[i],toSend) (contracts/distributor.sol#64)

Distributor.automatedDistribution() (contracts/distributor.sol#71-85) ignores return value by cowTip.transfer(team[i],toSendCowTip) (contracts/distributor.sol#81)

Distributor.singleSendTokens(address,address,uint256) (contracts/distributor.sol#87-94) ignores return value by token.transfer(to,amount) (contracts/distributor.sol#93)

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer>

INFO:Detectors:

CowTipFarm.pendingShare(uint256,address) (contracts/cowTipFarm.sol#277-303) performs a multiplication on the result of a division:

- _cowTipReward =

_generatedReward.mul(pool.allocPoint).div(totalAllocPoint).mul(1pPercent).div(10000) (contracts/cowTipFarm.sol#292-296)

CowTipFarm.pendingShare(uint256,address) (contracts/cowTipFarm.sol#277-303) performs a multiplication on the result of a division:

- _cowTipReward =

_generatedReward.mul(pool.allocPoint).div(totalAllocPoint).mul(1pPercent).div(10000) (contracts/cowTipFarm.sol#292-296)

- accTokensPerShare =

accTokensPerShare.add(_cowTipReward.mul(1e18).div(tokenSupply)) (contracts/cowTipFarm.sol#297-299)

CowTipFarm.updatePool(uint256) (contracts/cowTipFarm.sol#314-359) performs a

multiplication on the result of a division:

```
- _cowTipReward = _generatedReward.mul(pool.allocPoint).div(totalAllocPoint)
(contracts/cowTipFarm.sol#335-337)
- cowTip.mint(devAddress,_cowTipReward.mul(devPercent).div(10000)) (contracts/
cowTipFarm.sol#339-342)
```

CowTipFarm.updatePool(uint256) (contracts/cowTipFarm.sol#314-359) performs a multiplication on the result of a division:

```
- _cowTipReward = _generatedReward.mul(pool.allocPoint).div(totalAllocPoint)
(contracts/cowTipFarm.sol#335-337)
- cowTip.mint(feeAddress,_cowTipReward.mul(feePercent).div(10000)) (contracts/
cowTipFarm.sol#343-346)
```

CowTipFarm.updatePool(uint256) (contracts/cowTipFarm.sol#314-359) performs a multiplication on the result of a division:

```
- _cowTipReward = _generatedReward.mul(pool.allocPoint).div(totalAllocPoint)
(contracts/cowTipFarm.sol#335-337)
- cowTip.mint(address(this),_cowTipReward.mul(1pPercent).div(10000)) (contracts/
cowTipFarm.sol#347-350)
```

CowTipFarm.updatePool(uint256) (contracts/cowTipFarm.sol#314-359) performs a multiplication on the result of a division:

```
- pool.accTokensPerShare = pool.accTokensPerShare.add(_cowTipReward.mul(1e18).di
v(tokenSupply).mul(1pPercent).div(10000)) (contracts/cowTipFarm.sol#352-356)
```

CowTipFarm.getPoolView(uint256) (contracts/cowTipFarm.sol#591-615) performs a multiplication on the result of a division:

```
- rewardsPerSecond =
pool.allocPoint.mul(sharesPerSecond).div(totalAllocPoint).mul(1pPercent).div(10000)
(contracts/cowTipFarm.sol#595-600)
```

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply>

INFO:Detectors:

Reentrancy in YieldBarn._deposit(address,uint256,uint256) (contracts/yieldBarn.sol#108-157):

External calls:

```
- COWTIP.safeTransferFrom(_msgSender(),address(this),_amount) (contracts/
yieldBarn.sol#123)
```

State variables written after the call(s):

```
- totalCrops += _crop (contracts/yieldBarn.sol#134)
```

YieldBarn.totalCrops (contracts/yieldBarn.sol#35) can be used in cross function reentrancies:

```
- YieldBarn.pendingLPs(uint256) (contracts/yieldBarn.sol#324-347)
- YieldBarn.totalCrops (contracts/yieldBarn.sol#35)
- YieldBarn.updateAccLpPerCrop() (contracts/yieldBarn.sol#254-268)
```

```

- user.staked += _amount (contracts/yieldBarn.sol#152)
YieldBarn.users (contracts/yieldBarn.sol#21) can be used in cross function
reentrancies:
- YieldBarn.getUserActiveStakes(address) (contracts/yieldBarn.sol#375-400)
- YieldBarn.getUserPendingLPAllStakes(address) (contracts/
yieldBarn.sol#313-322)
- YieldBarn.getUserStats(address) (contracts/yieldBarn.sol#349-373)
- YieldBarn.pendingLPs(uint256) (contracts/yieldBarn.sol#324-347)
- user.lastDepositTime = block.timestamp (contracts/yieldBarn.sol#153)
YieldBarn.users (contracts/yieldBarn.sol#21) can be used in cross function
reentrancies:
- YieldBarn.getUserActiveStakes(address) (contracts/yieldBarn.sol#375-400)
- YieldBarn.getUserPendingLPAllStakes(address) (contracts/
yieldBarn.sol#313-322)
- YieldBarn.getUserStats(address) (contracts/yieldBarn.sol#349-373)
- YieldBarn.pendingLPs(uint256) (contracts/yieldBarn.sol#324-347)
- user.totalCrop += _crop (contracts/yieldBarn.sol#154)
YieldBarn.users (contracts/yieldBarn.sol#21) can be used in cross function
reentrancies:
- YieldBarn.getUserActiveStakes(address) (contracts/yieldBarn.sol#375-400)
- YieldBarn.getUserPendingLPAllStakes(address) (contracts/
yieldBarn.sol#313-322)
- YieldBarn.getUserStats(address) (contracts/yieldBarn.sol#349-373)
- YieldBarn.pendingLPs(uint256) (contracts/yieldBarn.sol#324-347)
Reentrancy in
CowTipFarm.add(uint256,IERC20,bool,uint256,uint16,uint16,IMasterChef,uint256)
(contracts/cowTipFarm.sol#161-237):
    External calls:
    - massUpdatePools() (contracts/cowTipFarm.sol#202)
      - cowTip.mint(devAddress,_cowTipReward.mul(devPercent).div(10000))
        (contracts/cowTipFarm.sol#339-342)
      - cowTip.mint(feeAddress,_cowTipReward.mul(feePercent).div(10000))
        (contracts/cowTipFarm.sol#343-346)
      - cowTip.mint(address(this),_cowTipReward.mul(lpPercent).div(10000))
        (contracts/cowTipFarm.sol#347-350)
    State variables written after the call(s):
    - poolInfo.push(PoolInfo(_token,_allocPoint,_lastRewardTime,0,_isStarted,_deposi
tFeeBP,_withdrawFeeBP,_externalFarm,0,_externalPid)) (contracts/cowTipFarm.sol#218-231)
    CowTipFarm.poolInfo (contracts/cowTipFarm.sol#98) can be used in cross function
reentrancies:
    - CowTipFarm.add(uint256,IERC20,bool,uint256,uint16,uint16,IMasterChef,uint256)

```


(contracts/cowTipFarm.sol#161-237)

- CowTipFarm.getAllPoolViews() (contracts/cowTipFarm.sol#617-623)
- CowTipFarm.getExternalReward(uint256) (contracts/cowTipFarm.sol#529-537)
- CowTipFarm.getPoolView(uint256) (contracts/cowTipFarm.sol#591-615)
- CowTipFarm.getUserView(uint256,address) (contracts/cowTipFarm.sol#625-641)
- CowTipFarm.getUserViews(address) (contracts/cowTipFarm.sol#643-651)
- CowTipFarm.massUpdatePools() (contracts/cowTipFarm.sol#306-311)
- CowTipFarm.pendingShare(uint256,address) (contracts/cowTipFarm.sol#277-303)
- CowTipFarm.poolInfo (contracts/cowTipFarm.sol#98)
- CowTipFarm.poolLength() (contracts/cowTipFarm.sol#156-158)
- CowTipFarm.set(uint256,uint256,uint16,uint16) (contracts/

cowTipFarm.sol#240-260)

- CowTipFarm.updatePool(uint256) (contracts/cowTipFarm.sol#314-359)
- totalAllocPoint = totalAllocPoint.add(_allocPoint) (contracts/

cowTipFarm.sol#233)

CowTipFarm.totalAllocPoint (contracts/cowTipFarm.sol#105) can be used in cross function reentrancies:

- CowTipFarm.add(uint256,IERC20,bool,uint256,uint16,uint16,IMasterChef,uint256)

(contracts/cowTipFarm.sol#161-237)

- CowTipFarm.getPoolView(uint256) (contracts/cowTipFarm.sol#591-615)
- CowTipFarm.pendingShare(uint256,address) (contracts/cowTipFarm.sol#277-303)
- CowTipFarm.set(uint256,uint256,uint16,uint16) (contracts/

cowTipFarm.sol#240-260)

- CowTipFarm.totalAllocPoint (contracts/cowTipFarm.sol#105)
- CowTipFarm.updatePool(uint256) (contracts/cowTipFarm.sol#314-359)

Reentrancy in PresaleDistributor.claim() (contracts/presaleDistributor.sol#90-117):

External calls:

- cowtip.safeTransfer(user,pendingCowTipAmount) (contracts/

presaleDistributor.sol#114)

State variables written after the call(s):

- users[user].cowtipClaimed += pendingCowTipAmount (contracts/

presaleDistributor.sol#115)

PresaleDistributor.users (contracts/presaleDistributor.sol#42) can be used in cross function reentrancies:

- PresaleDistributor.getTotalValues() (contracts/presaleDistributor.sol#62-76)
- PresaleDistributor.pendingCowTip(address) (contracts/

presaleDistributor.sol#124-134)

- PresaleDistributor.rewardsPerSecondCowTip(address) (contracts/

presaleDistributor.sol#119-122)

- PresaleDistributor.updateAllUsers() (contracts/

presaleDistributor.sol#188-198)

- PresaleDistributor.updateSingleUser(address) (contracts/presaleDistributor.sol#163-167)
- PresaleDistributor.updateSingleUserMan(address,uint256) (contracts/presaleDistributor.sol#169-175)
- PresaleDistributor.updateUsers(uint256,uint256) (contracts/presaleDistributor.sol#177-186)
- PresaleDistributor.users (contracts/presaleDistributor.sol#42)
- users[user].lastClaimTime = block.timestamp (contracts/presaleDistributor.sol#116)

PresaleDistributor.users (contracts/presaleDistributor.sol#42) can be used in cross function reentrancies:

- PresaleDistributor.getTotalValues() (contracts/presaleDistributor.sol#62-76)
- PresaleDistributor.pendingCowTip(address) (contracts/presaleDistributor.sol#124-134)
- PresaleDistributor.rewardsPerSecondCowTip(address) (contracts/presaleDistributor.sol#119-122)
- PresaleDistributor.updateAllUsers() (contracts/presaleDistributor.sol#188-198)
- PresaleDistributor.updateSingleUser(address) (contracts/presaleDistributor.sol#163-167)
- PresaleDistributor.updateSingleUserMan(address,uint256) (contracts/presaleDistributor.sol#169-175)
- PresaleDistributor.updateUsers(uint256,uint256) (contracts/presaleDistributor.sol#177-186)
- PresaleDistributor.users (contracts/presaleDistributor.sol#42)

Reentrancy in CowTipFarm.emergencyWithdraw(uint256) (contracts/cowTipFarm.sol#495-515):

External calls:

- pool.externalFarm.withdraw(pool.externalPid,_amount) (contracts/cowTipFarm.sol#504)
- pool.token.safeTransfer(feeAddress,withdrawFee) (contracts/cowTipFarm.sol#508)
- pool.token.safeTransfer(_msgSender(),_amount.sub(withdrawFee)) (contracts/cowTipFarm.sol#509)
- pool.token.safeTransfer(_msgSender(),_amount) (contracts/cowTipFarm.sol#511)

State variables written after the call(s):

- pool.lpBalance -= _amount (contracts/cowTipFarm.sol#513)

CowTipFarm.poolInfo (contracts/cowTipFarm.sol#98) can be used in cross function reentrancies:

- CowTipFarm.add(uint256,IERC20,bool,uint256,uint16,uint16,IMasterChef,uint256) (contracts/cowTipFarm.sol#161-237)
- CowTipFarm.getAllPoolViews() (contracts/cowTipFarm.sol#617-623)

- CowTipFarm.getExternalReward(uint256) (contracts/cowTipFarm.sol#529-537)
- CowTipFarm.getPoolView(uint256) (contracts/cowTipFarm.sol#591-615)
- CowTipFarm.getUserView(uint256,address) (contracts/cowTipFarm.sol#625-641)
- CowTipFarm.getUserViews(address) (contracts/cowTipFarm.sol#643-651)
- CowTipFarm.massUpdatePools() (contracts/cowTipFarm.sol#306-311)
- CowTipFarm.pendingShare(uint256,address) (contracts/cowTipFarm.sol#277-303)
- CowTipFarm.poolInfo (contracts/cowTipFarm.sol#98)
- CowTipFarm.poolLength() (contracts/cowTipFarm.sol#156-158)
- CowTipFarm.set(uint256,uint256,uint16,uint16) (contracts/cowTipFarm.sol#240-260)
- CowTipFarm.updatePool(uint256) (contracts/cowTipFarm.sol#314-359)

Reentrancy in CowTipFarm.removeExternalFarm(uint256) (contracts/cowTipFarm.sol#541-547):

External calls:

- pool.externalFarm.emergencyWithdraw(pool.externalPid) (contracts/cowTipFarm.sol#545)

State variables written after the call(s):

- pool.externalFarm = IMasterChef(address(0)) (contracts/cowTipFarm.sol#546)

CowTipFarm.poolInfo (contracts/cowTipFarm.sol#98) can be used in cross function reentrancies:

- CowTipFarm.add(uint256,IERC20,bool,uint256,uint16,uint16,IMasterChef,uint256) (contracts/cowTipFarm.sol#161-237)
- CowTipFarm.getAllPoolViews() (contracts/cowTipFarm.sol#617-623)
- CowTipFarm.getExternalReward(uint256) (contracts/cowTipFarm.sol#529-537)
- CowTipFarm.getPoolView(uint256) (contracts/cowTipFarm.sol#591-615)
- CowTipFarm.getUserView(uint256,address) (contracts/cowTipFarm.sol#625-641)
- CowTipFarm.getUserViews(address) (contracts/cowTipFarm.sol#643-651)
- CowTipFarm.massUpdatePools() (contracts/cowTipFarm.sol#306-311)
- CowTipFarm.pendingShare(uint256,address) (contracts/cowTipFarm.sol#277-303)
- CowTipFarm.poolInfo (contracts/cowTipFarm.sol#98)
- CowTipFarm.poolLength() (contracts/cowTipFarm.sol#156-158)
- CowTipFarm.set(uint256,uint256,uint16,uint16) (contracts/cowTipFarm.sol#240-260)
- CowTipFarm.updatePool(uint256) (contracts/cowTipFarm.sol#314-359)

Reentrancy in CowTipFarm.set(uint256,uint256,uint16,uint16) (contracts/cowTipFarm.sol#240-260):

External calls:

- massUpdatePools() (contracts/cowTipFarm.sol#250)
 - cowTip.mint(devAddress,_cowTipReward.mul(devPercent).div(10000)) (contracts/cowTipFarm.sol#339-342)
 - cowTip.mint(feeAddress,_cowTipReward.mul(feePercent).div(10000))

```

(contracts/cowTipFarm.sol#343-346)
    - cowTip.mint(address(this),_cowTipReward.mul(1pPercent).div(10000))
(contracts/cowTipFarm.sol#347-350)
    State variables written after the call(s):
    - pool.allocPoint = _allocPoint (contracts/cowTipFarm.sol#257)
    CowTipFarm.poolInfo (contracts/cowTipFarm.sol#98) can be used in cross function
reentrancies:
    - CowTipFarm.add(uint256,IERC20,bool,uint256,uint16,uint16,IMasterChef,uint256)
(contracts/cowTipFarm.sol#161-237)
    - CowTipFarm.getAllPoolViews() (contracts/cowTipFarm.sol#617-623)
    - CowTipFarm.getExternalReward(uint256) (contracts/cowTipFarm.sol#529-537)
    - CowTipFarm.getPoolView(uint256) (contracts/cowTipFarm.sol#591-615)
    - CowTipFarm.getUserView(uint256,address) (contracts/cowTipFarm.sol#625-641)
    - CowTipFarm.getUserViews(address) (contracts/cowTipFarm.sol#643-651)
    - CowTipFarm.massUpdatePools() (contracts/cowTipFarm.sol#306-311)
    - CowTipFarm.pendingShare(uint256,address) (contracts/cowTipFarm.sol#277-303)
    - CowTipFarm.poolInfo (contracts/cowTipFarm.sol#98)
    - CowTipFarm.poolLength() (contracts/cowTipFarm.sol#156-158)
    - CowTipFarm.set(uint256,uint256,uint16,uint16) (contracts/
cowTipFarm.sol#240-260)
    - CowTipFarm.updatePool(uint256) (contracts/cowTipFarm.sol#314-359)
    - poolInfo[_pid].depositFeeBP = _depositFeeBP (contracts/cowTipFarm.sol#258)
    CowTipFarm.poolInfo (contracts/cowTipFarm.sol#98) can be used in cross function
reentrancies:
    - CowTipFarm.add(uint256,IERC20,bool,uint256,uint16,uint16,IMasterChef,uint256)
(contracts/cowTipFarm.sol#161-237)
    - CowTipFarm.getAllPoolViews() (contracts/cowTipFarm.sol#617-623)
    - CowTipFarm.getExternalReward(uint256) (contracts/cowTipFarm.sol#529-537)
    - CowTipFarm.getPoolView(uint256) (contracts/cowTipFarm.sol#591-615)
    - CowTipFarm.getUserView(uint256,address) (contracts/cowTipFarm.sol#625-641)
    - CowTipFarm.getUserViews(address) (contracts/cowTipFarm.sol#643-651)
    - CowTipFarm.massUpdatePools() (contracts/cowTipFarm.sol#306-311)
    - CowTipFarm.pendingShare(uint256,address) (contracts/cowTipFarm.sol#277-303)
    - CowTipFarm.poolInfo (contracts/cowTipFarm.sol#98)
    - CowTipFarm.poolLength() (contracts/cowTipFarm.sol#156-158)
    - CowTipFarm.set(uint256,uint256,uint16,uint16) (contracts/
cowTipFarm.sol#240-260)
    - CowTipFarm.updatePool(uint256) (contracts/cowTipFarm.sol#314-359)
    - poolInfo[_pid].withdrawFeeBP = _withdrawFeeBP (contracts/cowTipFarm.sol#259)
    CowTipFarm.poolInfo (contracts/cowTipFarm.sol#98) can be used in cross function
reentrancies:

```

- CowTipFarm.add(uint256,IERC20,bool,uint256,uint16,uint16,IMasterChef,uint256) (contracts/cowTipFarm.sol#161-237)
- CowTipFarm.getAllPoolViews() (contracts/cowTipFarm.sol#617-623)
- CowTipFarm.getExternalReward(uint256) (contracts/cowTipFarm.sol#529-537)
- CowTipFarm.getPoolView(uint256) (contracts/cowTipFarm.sol#591-615)
- CowTipFarm.getUserView(uint256,address) (contracts/cowTipFarm.sol#625-641)
- CowTipFarm.getUserViews(address) (contracts/cowTipFarm.sol#643-651)
- CowTipFarm.massUpdatePools() (contracts/cowTipFarm.sol#306-311)
- CowTipFarm.pendingShare(uint256,address) (contracts/cowTipFarm.sol#277-303)
- CowTipFarm.poolInfo (contracts/cowTipFarm.sol#98)
- CowTipFarm.poolLength() (contracts/cowTipFarm.sol#156-158)
- CowTipFarm.set(uint256,uint256,uint16,uint16) (contracts/cowTipFarm.sol#240-260)
- CowTipFarm.updatePool(uint256) (contracts/cowTipFarm.sol#314-359)
- totalAllocPoint = totalAllocPoint.sub(pool.allocPoint).add(_allocPoint) (contracts/cowTipFarm.sol#253-255)

CowTipFarm.totalAllocPoint (contracts/cowTipFarm.sol#105) can be used in cross function reentrancies:

- CowTipFarm.add(uint256,IERC20,bool,uint256,uint16,uint16,IMasterChef,uint256) (contracts/cowTipFarm.sol#161-237)
- CowTipFarm.getPoolView(uint256) (contracts/cowTipFarm.sol#591-615)
- CowTipFarm.pendingShare(uint256,address) (contracts/cowTipFarm.sol#277-303)
- CowTipFarm.set(uint256,uint256,uint16,uint16) (contracts/cowTipFarm.sol#240-260)
- CowTipFarm.totalAllocPoint (contracts/cowTipFarm.sol#105)
- CowTipFarm.updatePool(uint256) (contracts/cowTipFarm.sol#314-359)

Reentrancy in PresaleDistributor.stake() (contracts/presaleDistributor.sol#78-88):

External calls:

- cowtip.approve(address(yieldBarn),amount) (contracts/presaleDistributor.sol#84)
- yieldBarn.depositFor(user,amount,stakingType) (contracts/presaleDistributor.sol#85)

State variables written after the call(s):

- users[user].cowtipClaimed = users[user].cowtipBought (contracts/presaleDistributor.sol#86)

PresaleDistributor.users (contracts/presaleDistributor.sol#42) can be used in cross function reentrancies:

- PresaleDistributor.getTotalValues() (contracts/presaleDistributor.sol#62-76)
- PresaleDistributor.pendingCowTip(address) (contracts/presaleDistributor.sol#124-134)
- PresaleDistributor.rewardsPerSecondCowTip(address) (contracts/presaleDistributor.sol#119-122)

- PresaleDistributor.updateAllUsers() (contracts/presaleDistributor.sol#188-198)
- PresaleDistributor.updateSingleUser(address) (contracts/presaleDistributor.sol#163-167)
- PresaleDistributor.updateSingleUserMan(address,uint256) (contracts/presaleDistributor.sol#169-175)
- PresaleDistributor.updateUsers(uint256,uint256) (contracts/presaleDistributor.sol#177-186)
- PresaleDistributor.users (contracts/presaleDistributor.sol#42)
- users[user].cowtipBought = 0 (contracts/presaleDistributor.sol#87)

PresaleDistributor.users (contracts/presaleDistributor.sol#42) can be used in cross function reentrancies:

- PresaleDistributor.getTotalValues() (contracts/presaleDistributor.sol#62-76)
- PresaleDistributor.pendingCowTip(address) (contracts/presaleDistributor.sol#124-134)
- PresaleDistributor.rewardsPerSecondCowTip(address) (contracts/presaleDistributor.sol#119-122)
- PresaleDistributor.updateAllUsers() (contracts/presaleDistributor.sol#188-198)
- PresaleDistributor.updateSingleUser(address) (contracts/presaleDistributor.sol#163-167)
- PresaleDistributor.updateSingleUserMan(address,uint256) (contracts/presaleDistributor.sol#169-175)
- PresaleDistributor.updateUsers(uint256,uint256) (contracts/presaleDistributor.sol#177-186)
- PresaleDistributor.users (contracts/presaleDistributor.sol#42)

Reentrancy in YieldBarn.stopReward() (contracts/yieldBarn.sol#93-98):

External calls:

- LP.safeTransfer(_msgSender(),tokenReward) (contracts/yieldBarn.sol#96)

State variables written after the call(s):

- endTime = block.timestamp (contracts/yieldBarn.sol#97)

YieldBarn.endTime (contracts/yieldBarn.sol#31) can be used in cross function reentrancies:

- YieldBarn._injectRewardsWithTime(uint256,uint256) (contracts/yieldBarn.sol#292-311)
- YieldBarn.constructor(IERC20,IERC20) (contracts/yieldBarn.sol#83-91)
- YieldBarn.endTime (contracts/yieldBarn.sol#31)
- YieldBarn.getMultiplier(uint256,uint256) (contracts/yieldBarn.sol#270-281)
- YieldBarn.stopReward() (contracts/yieldBarn.sol#93-98)

Reentrancy in CowTipFarm.updateEmissionRate(uint256) (contracts/cowTipFarm.sol#586-589):

External calls:

- massUpdatePools() (contracts/cowTipFarm.sol#587)
 - cowTip.mint(devAddress,_cowTipReward.mul(devPercent).div(10000)) (contracts/cowTipFarm.sol#339-342)
 - cowTip.mint(feeAddress,_cowTipReward.mul(feePercent).div(10000)) (contracts/cowTipFarm.sol#343-346)
 - cowTip.mint(address(this),_cowTipReward.mul(1pPercent).div(10000)) (contracts/cowTipFarm.sol#347-350)

State variables written after the call(s):

- sharesPerSecond = _sharesPerSecond (contracts/cowTipFarm.sol#588)

CowTipFarm.sharesPerSecond (contracts/cowTipFarm.sol#113) can be used in cross function reentrancies:

-

CowTipFarm.constructor(ICOWTIP,address,address,uint256,uint256,uint256,uint256) (contracts/cowTipFarm.sol#127-154)

- CowTipFarm.getPoolView(uint256) (contracts/cowTipFarm.sol#591-615)
- CowTipFarm.pendingShare(uint256,address) (contracts/cowTipFarm.sol#277-303)
- CowTipFarm.sharesPerSecond (contracts/cowTipFarm.sol#113)
- CowTipFarm.updateEmissionRate(uint256) (contracts/cowTipFarm.sol#586-589)
- CowTipFarm.updatePool(uint256) (contracts/cowTipFarm.sol#314-359)

Reentrancy in CowTipFarm.updatePool(uint256) (contracts/cowTipFarm.sol#314-359):

External calls:

- cowTip.mint(devAddress,_cowTipReward.mul(devPercent).div(10000)) (contracts/cowTipFarm.sol#339-342)
- cowTip.mint(feeAddress,_cowTipReward.mul(feePercent).div(10000)) (contracts/cowTipFarm.sol#343-346)
- cowTip.mint(address(this),_cowTipReward.mul(1pPercent).div(10000)) (contracts/cowTipFarm.sol#347-350)

State variables written after the call(s):

- pool.accTokensPerShare = pool.accTokensPerShare.add(_cowTipReward.mul(1e18).div(tokenSupply).mul(1pPercent).div(10000)) (contracts/cowTipFarm.sol#352-356)

CowTipFarm.poolInfo (contracts/cowTipFarm.sol#98) can be used in cross function reentrancies:

- CowTipFarm.add(uint256,IERC20,bool,uint256,uint16,uint16,IMasterChef,uint256) (contracts/cowTipFarm.sol#161-237)
- CowTipFarm.getAllPoolViews() (contracts/cowTipFarm.sol#617-623)
- CowTipFarm.getExternalReward(uint256) (contracts/cowTipFarm.sol#529-537)
- CowTipFarm.getPoolView(uint256) (contracts/cowTipFarm.sol#591-615)
- CowTipFarm.getUserView(uint256,address) (contracts/cowTipFarm.sol#625-641)
- CowTipFarm.getUserViews(address) (contracts/cowTipFarm.sol#643-651)
- CowTipFarm.massUpdatePools() (contracts/cowTipFarm.sol#306-311)

- CowTipFarm.pendingShare(uint256,address) (contracts/cowTipFarm.sol#277-303)
- CowTipFarm.poolInfo (contracts/cowTipFarm.sol#98)
- CowTipFarm.poolLength() (contracts/cowTipFarm.sol#156-158)
- CowTipFarm.set(uint256,uint256,uint16,uint16) (contracts/cowTipFarm.sol#240-260)
- CowTipFarm.updatePool(uint256) (contracts/cowTipFarm.sol#314-359)
- pool.lastRewardTime = block.timestamp (contracts/cowTipFarm.sol#358)

CowTipFarm.poolInfo (contracts/cowTipFarm.sol#98) can be used in cross function reentrancies:

- CowTipFarm.add(uint256,IERC20,bool,uint256,uint16,uint16,IMasterChef,uint256) (contracts/cowTipFarm.sol#161-237)
- CowTipFarm.getAllPoolViews() (contracts/cowTipFarm.sol#617-623)
- CowTipFarm.getExternalReward(uint256) (contracts/cowTipFarm.sol#529-537)
- CowTipFarm.getPoolView(uint256) (contracts/cowTipFarm.sol#591-615)
- CowTipFarm.getUserView(uint256,address) (contracts/cowTipFarm.sol#625-641)
- CowTipFarm.getUserViews(address) (contracts/cowTipFarm.sol#643-651)
- CowTipFarm.massUpdatePools() (contracts/cowTipFarm.sol#306-311)
- CowTipFarm.pendingShare(uint256,address) (contracts/cowTipFarm.sol#277-303)
- CowTipFarm.poolInfo (contracts/cowTipFarm.sol#98)
- CowTipFarm.poolLength() (contracts/cowTipFarm.sol#156-158)
- CowTipFarm.set(uint256,uint256,uint16,uint16) (contracts/cowTipFarm.sol#240-260)
- CowTipFarm.updatePool(uint256) (contracts/cowTipFarm.sol#314-359)

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1>

INFO:Detectors:

YieldBarn._deposit(address,uint256,uint256)._duration (contracts/yieldBarn.sol#111) is a local variable never initialized

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables>

INFO:Detectors:

CowTip._swapCOWTIP(uint256) (contracts/cowTip.sol#69-85) ignores return value by this.approve(address(ROUTER),type()(uint256).max) (contracts/cowTip.sol#76)

CowTip._swapCOWTIP(uint256) (contracts/cowTip.sol#69-85) ignores return value by ROUTER.swapExactTokensForTokens(taxAmount,0,path,communityFund,deadline) (contracts/cowTip.sol#78-84)

CowTipFarm.add(uint256,IERC20,bool,uint256,uint16,uint16,IMasterChef,uint256) (contracts/cowTipFarm.sol#161-237) ignores return value by (masterChefLpToken) = _externalFarm.poolInfo(_externalPid) (contracts/cowTipFarm.sol#186)

CowTipFarm.add(uint256,IERC20,bool,uint256,uint16,uint16,IMasterChef,uint256) (contracts/cowTipFarm.sol#161-237) ignores return value by

1pTokens.add(address(_token)) (contracts/cowTipFarm.sol#236)
 CowTipFarm._deposit(uint256,uint256,address,address) (contracts/cowTipFarm.sol#383-444)
 ignores return value by pool.token.approve(_externalFarm,toDeposit) (contracts/
 cowTipFarm.sol#440)
 PreSale.setRegisterPublic(address,bool) (contracts/presale.sol#67-77) ignores return
 value by playerIndex.add(_addr) (contracts/presale.sol#73)
 PreSale.setRegisterPublic(address,bool) (contracts/presale.sol#67-77) ignores return
 value by playerIndex.remove(_addr) (contracts/presale.sol#75)
 PreSale.addRegisterMultiplePublic(address[]) (contracts/presale.sol#79-87) ignores
 return value by playerIndex.add(_addrs[i]) (contracts/presale.sol#85)
 PresaleDistributor.stake() (contracts/presaleDistributor.sol#78-88) ignores return
 value by cowtip.approve(address(yieldBarn),amount) (contracts/
 presaleDistributor.sol#84)
 YieldBarn._deposit(address,uint256,uint256) (contracts/yieldBarn.sol#108-157) ignores
 return value by user.activeStakes.add(stakeID) (contracts/yieldBarn.sol#151)
 YieldBarn._endStake(uint256) (contracts/yieldBarn.sol#160-199) ignores return value by
 user.activeStakes.remove(_stakeID) (contracts/yieldBarn.sol#188)
 YieldBarn._endStake(uint256) (contracts/yieldBarn.sol#160-199) ignores return value by
 user.endedStakes.add(_stakeID) (contracts/yieldBarn.sol#189)
 Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return>
 INFO:Detectors:
 Distributor.setCaller(address) (contracts/distributor.sol#25-27) should emit an event
 for:
 - caller = _caller (contracts/distributor.sol#26)
 Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-access-control>
 INFO:Detectors:
 StakingVault.setStakingRoundDuration(uint256) (contracts/StakingVault.sol#88-92) should
 emit an event for:
 - stakingRoundDuration = _stakingRoundDuration (contracts/StakingVault.sol#91)
 StakingVault.setDistributionPercentage(uint256) (contracts/StakingVault.sol#94-98)
 should emit an event for:
 - distributionPercentage = _distributionPercentage (contracts/
 StakingVault.sol#97)
 CowTipFarm.add(uint256,IERC20,bool,uint256,uint16,uint16,IMasterChef,uint256)
 (contracts/cowTipFarm.sol#161-237) should emit an event for:
 - totalAllocPoint = totalAllocPoint.add(_allocPoint) (contracts/
 cowTipFarm.sol#233)
 CowTipFarm.set(uint256,uint256,uint16,uint16) (contracts/cowTipFarm.sol#240-260) should
 emit an event for:
 - totalAllocPoint = totalAllocPoint.sub(pool.allocPoint).add(_allocPoint)

(contracts/cowTipFarm.sol#253-255)

CowTipFarm.setFeePercent(uint256) (contracts/cowTipFarm.sol#557-563) should emit an event for:

- feePercent = _feePercent (contracts/cowTipFarm.sol#562)

CowTipFarm.setDevPercent(uint256) (contracts/cowTipFarm.sol#573-579) should emit an event for:

- devPercent = _devPercent (contracts/cowTipFarm.sol#578)

CowTipFarm.setReferralRate(uint256) (contracts/cowTipFarm.sol#581-584) should emit an event for:

- referralRate = _referralRate (contracts/cowTipFarm.sol#583)

CowTipFarm.updateEmissionRate(uint256) (contracts/cowTipFarm.sol#586-589) should emit an event for:

- sharesPerSecond = _sharesPerSecond (contracts/cowTipFarm.sol#588)

Distributor.setArrayLimit(uint256) (contracts/distributor.sol#38-41) should emit an event for:

- arrayLimit = _newLimit (contracts/distributor.sol#40)

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic>

INFO:Detectors:

CommunityFund.sendCustomTransaction(address,uint256,string,bytes).target (contracts/CommunityFund.sol#45) lacks a zero-check on :

- (success,returnData) = target.call{value: value}(callData) (contracts/CommunityFund.sol#60-62)

CowTip.constructor(address)._communityFund (contracts/cowTip.sol#44) lacks a zero-check on :

- communityFund = _communityFund (contracts/cowTip.sol#52)

CowTip.setTaxManager(address)._taxManager (contracts/cowTip.sol#61) lacks a zero-check on :

- taxManager = _taxManager (contracts/cowTip.sol#62)

CowTip.setCommunityFund(address)._communityFund (contracts/cowTip.sol#65) lacks a zero-check on :

- communityFund = _communityFund (contracts/cowTip.sol#66)

Distributor.setCaller(address)._caller (contracts/distributor.sol#25) lacks a zero-check on :

- caller = _caller (contracts/distributor.sol#26)

Distributor.setTokens(address)._COWTIP (contracts/distributor.sol#47) lacks a zero-check on :

- COWTIP = _COWTIP (contracts/distributor.sol#48)

Distributor.sendCustomTransaction(address,uint256,string,bytes).target (contracts/distributor.sol#105) lacks a zero-check on :

- (success,returnData) = target.call{value: value}(callData) (contracts/distributor.sol#120-122)

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation>

INFO:Detectors:

CowTipFarm.updatePool(uint256) (contracts/cowTipFarm.sol#314-359) has external calls inside a loop: cowTip.mint(devAddress,_cowTipReward.mul(devPercent).div(10000)) (contracts/cowTipFarm.sol#339-342)

CowTipFarm.updatePool(uint256) (contracts/cowTipFarm.sol#314-359) has external calls inside a loop: cowTip.mint(feeAddress,_cowTipReward.mul(feePercent).div(10000)) (contracts/cowTipFarm.sol#343-346)

CowTipFarm.updatePool(uint256) (contracts/cowTipFarm.sol#314-359) has external calls inside a loop: cowTip.mint(address(this),_cowTipReward.mul(lpPercent).div(10000)) (contracts/cowTipFarm.sol#347-350)

CowTipFarm.getUserView(uint256,address) (contracts/cowTipFarm.sol#625-641) has external calls inside a loop: lpBalance = pool.token.balanceOf(account) (contracts/cowTipFarm.sol#632)

CowTipFarm.getUserView(uint256,address) (contracts/cowTipFarm.sol#625-641) has external calls inside a loop: UserView(pid,user.amount,unclaimedRewards,lpBalance,pool.token.allowance(account,address(this))) (contracts/cowTipFarm.sol#633-640)

Distributor.distribute(address,address[],uint256) (contracts/distributor.sol#51-69) has external calls inside a loop: erc20token.transfer(_receivers[i],toSend) (contracts/distributor.sol#64)

Distributor.automatedDistribution() (contracts/distributor.sol#71-85) has external calls inside a loop: cowTip.transfer(team[i],toSendCowTip) (contracts/distributor.sol#81)

PresaleDistributor.getTotalValues() (contracts/presaleDistributor.sol#62-76) has external calls inside a loop: currentUser = presale.userIndex(i) (contracts/presaleDistributor.sol#72)

PresaleDistributor.updateUsers(uint256,uint256) (contracts/presaleDistributor.sol#177-186) has external calls inside a loop: currentUser = presale.userIndex(i) (contracts/presaleDistributor.sol#179)

PresaleDistributor.updateUsers(uint256,uint256) (contracts/presaleDistributor.sol#177-186) has external calls inside a loop: userData = presale.users(currentUser) (contracts/presaleDistributor.sol#180-182)

PresaleDistributor.updateAllUsers() (contracts/presaleDistributor.sol#188-198) has external calls inside a loop: currentUser = presale.userIndex(i) (contracts/presaleDistributor.sol#191)

PresaleDistributor.updateAllUsers() (contracts/presaleDistributor.sol#188-198) has external calls inside a loop: userData = presale.users(currentUser) (contracts/presaleDistributor.sol#192-194)

YieldBarn._endStake(uint256) (contracts/yieldBarn.sol#160-199) has external calls inside a loop: require(bool,string)(LP.balanceOf(address(this)) >=

pendingRewards,Insufficient LP balance) (contracts/yieldBarn.sol#174-177)

YieldBarn._claimRewards(uint256) (contracts/yieldBarn.sol#201-223) has external calls inside a loop: require(bool,string)(LP.balanceOf(address(this)) >=

pendingRewards,Insufficient LP balance) (contracts/yieldBarn.sol#211-214)

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop>

INFO:Detectors:

Reentrancy in CowTipFarm._deposit(uint256,uint256,address,address) (contracts/cowTipFarm.sol#383-444):

External calls:

- updatePool(_pid) (contracts/cowTipFarm.sol#404)
 - cowTip.mint(devAddress,_cowTipReward.mul(devPercent).div(10000)) (contracts/cowTipFarm.sol#339-342)
 - cowTip.mint(feeAddress,_cowTipReward.mul(feePercent).div(10000)) (contracts/cowTipFarm.sol#343-346)
 - cowTip.mint(address(this),_cowTipReward.mul(lpPercent).div(10000)) (contracts/cowTipFarm.sol#347-350)

State variables written after the call(s):

- referralEarned[_referrer] = referralEarned[_referrer] + referralAmount (contracts/cowTipFarm.sol#414-416)

Reentrancy in YieldBarn._deposit(address,uint256,uint256) (contracts/yieldBarn.sol#108-157):

External calls:

- COWTIP.safeTransferFrom(_msgSender(),address(this),_amount) (contracts/yieldBarn.sol#123)

State variables written after the call(s):

- infiniteStakesCount ++ (contracts/yieldBarn.sol#128)
- stakes[stakeID] = newStake (contracts/yieldBarn.sol#150)
- totalCOWTIPStaked += _amount (contracts/yieldBarn.sol#155)
- totalStakesCount ++ (contracts/yieldBarn.sol#156)

Reentrancy in CowTipFarm._withdraw(uint256,uint256) (contracts/cowTipFarm.sol#446-492):

External calls:

- updatePool(_pid) (contracts/cowTipFarm.sol#454)
 - cowTip.mint(devAddress,_cowTipReward.mul(devPercent).div(10000)) (contracts/cowTipFarm.sol#339-342)
 - cowTip.mint(feeAddress,_cowTipReward.mul(feePercent).div(10000)) (contracts/cowTipFarm.sol#343-346)
 - cowTip.mint(address(this),_cowTipReward.mul(lpPercent).div(10000)) (contracts/cowTipFarm.sol#347-350)

State variables written after the call(s):

- referralEarned[referrer] = referralEarned[referrer] + referralAmount

(contracts/cowTipFarm.sol#465-467)

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2>

INFO:Detectors:

Reentrancy in Distributor.automatedDistribution() (contracts/distributor.sol#71-85):

External calls:

- cowTip.transfer(team[i],toSendCowTip) (contracts/distributor.sol#81)

Event emitted after the call(s):

- Multisended(totalCowTip,COWTIP) (contracts/distributor.sol#84)

Reentrancy in Distributor.distribute(address,address[],uint256) (contracts/distributor.sol#51-69):

External calls:

- erc20token.transfer(_receivers[i],toSend) (contracts/distributor.sol#64)

Event emitted after the call(s):

- Multisended(total,token) (contracts/distributor.sol#68)

Reentrancy in CommunityFund.sendCustomTransaction(address,uint256,string,bytes) (contracts/CommunityFund.sol#44-68):

External calls:

- (success,returnData) = target.call{value: value}(callData) (contracts/CommunityFund.sol#60-62)

Event emitted after the call(s):

- ExecuteTransaction(txHash,target,value,signature,data) (contracts/CommunityFund.sol#65)

Reentrancy in Distributor.sendCustomTransaction(address,uint256,string,bytes) (contracts/distributor.sol#104-128):

External calls:

- (success,returnData) = target.call{value: value}(callData) (contracts/distributor.sol#120-122)

Event emitted after the call(s):

- ExecuteTransaction(txHash,target,value,signature,data) (contracts/distributor.sol#125)

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3>

INFO:Detectors:

StakingVault.checkRound() (contracts/StakingVault.sol#50-54) uses timestamp for comparisons

Dangerous comparisons:

- block.timestamp > lastStakingRound + stakingRoundDuration (contracts/StakingVault.sol#51)

CowTipFarm.constructor(ICOWTIP,address,address,uint256,uint256,uint256,uint256) (contracts/cowTipFarm.sol#127-154) uses timestamp for comparisons

Dangerous comparisons:

- require(bool,string)(block.timestamp < _poolStartTime,CowTipFarm: late)

(contracts/cowTipFarm.sol#136)

CowTipFarm.add(uint256,IERC20,bool,uint256,uint16,uint16,IMasterChef,uint256)

(contracts/cowTipFarm.sol#161-237) uses timestamp for comparisons

Dangerous comparisons:

- block.timestamp < poolStartTime (contracts/cowTipFarm.sol#205)
- _lastRewardTime < poolStartTime (contracts/cowTipFarm.sol#207)
- _lastRewardTime < block.timestamp (contracts/cowTipFarm.sol#212)
- _isStarted = (block.timestamp >= poolStartTime) && (block.timestamp >=

_lastRewardTime) (contracts/cowTipFarm.sol#216-217)

CowTipFarm.getMultiplier(uint256,uint256) (contracts/cowTipFarm.sol#263-274) uses timestamp for comparisons

Dangerous comparisons:

- _from >= _to (contracts/cowTipFarm.sol#267)
- _to <= poolStartTime (contracts/cowTipFarm.sol#268)

CowTipFarm.pendingShare(uint256,address) (contracts/cowTipFarm.sol#277-303) uses timestamp for comparisons

Dangerous comparisons:

- block.timestamp > pool.lastRewardTime && tokenSupply != 0 (contracts/cowTipFarm.sol#285)

CowTipFarm.massUpdatePools() (contracts/cowTipFarm.sol#306-311) uses timestamp for comparisons

Dangerous comparisons:

- pid < length (contracts/cowTipFarm.sol#308)

CowTipFarm.updatePool(uint256) (contracts/cowTipFarm.sol#314-359) uses timestamp for comparisons

Dangerous comparisons:

- block.timestamp <= pool.lastRewardTime (contracts/cowTipFarm.sol#316)

CowTipFarm.getPoolView(uint256) (contracts/cowTipFarm.sol#591-615) uses timestamp for comparisons

Dangerous comparisons:

- require(bool,string)(pid < poolInfo.length,CowTipFarm: pid out of range)

(contracts/cowTipFarm.sol#592)

CowTipFarm.getAllPoolViews() (contracts/cowTipFarm.sol#617-623) uses timestamp for comparisons

Dangerous comparisons:

- i < poolInfo.length (contracts/cowTipFarm.sol#619)

CowTipFarm.getUserViews(address) (contracts/cowTipFarm.sol#643-651) uses timestamp for comparisons

Dangerous comparisons:

```

- i < poolInfo.length (contracts/cowTipFarm.sol#647)
PreSale.drawWinner() (contracts/presale.sol#117-136) uses timestamp for comparisons
    Dangerous comparisons:
    - require(bool,string)(block.timestamp < presaleStartTime,Cannot draw winner
after presale starts) (contracts/presale.sol#119)
PreSale.Buy() (contracts/presale.sol#152-184) uses timestamp for comparisons
    Dangerous comparisons:
    - require(bool,string)(block.timestamp > presaleStartTime,Not started yet!)
(contracts/presale.sol#158)
    - block.timestamp >= presaleStartTime && block.timestamp < publicStartTime
(contracts/presale.sol#161)
    - block.timestamp >= publicStartTime && block.timestamp < fcfsStartTime
(contracts/presale.sol#164)
    - require(bool,string)(block.timestamp >= fcfsStartTime,Sale not yet open to
public) (contracts/presale.sol#168)
PresaleDistributor.claim() (contracts/presaleDistributor.sol#90-117) uses timestamp for
comparisons
    Dangerous comparisons:
    - (pendingCowTipAmount + cowtipClaimed) > cowtipBought (contracts/
presaleDistributor.sol#106)
    - require(bool,string)((pendingCowTipAmount) > 0,Nothing to claim!) (contracts/
presaleDistributor.sol#109-112)
PresaleDistributor.pendingCowTip(address) (contracts/presaleDistributor.sol#124-134)
uses timestamp for comparisons
    Dangerous comparisons:
    - block.timestamp > lastRewardTime (contracts/presaleDistributor.sol#127)
PresaleDistributor.getMultiplier(uint256,uint256) (contracts/
presaleDistributor.sol#136-147) uses timestamp for comparisons
    Dangerous comparisons:
    - _to <= endTime (contracts/presaleDistributor.sol#140)
YieldBarn._endStake(uint256) (contracts/yieldBarn.sol#160-199) uses timestamp for
comparisons
    Dangerous comparisons:
    - require(bool,string)(_msgSender() == stake.owner,Not the owner) (contracts/
yieldBarn.sol#163)
    - require(bool,string)
(users[_msgSender()].activeStakes.contains(_stakeID),Invalid stake number) (contracts/
yieldBarn.sol#164-167)
    - require(bool,string)(block.timestamp >= stake.stakeEndTime,Staking duration
has not ended) (contracts/yieldBarn.sol#168-171)
    - require(bool,string)(LP.balanceOf(address(this)) >=

```

pendingRewards,Insufficient LP balance) (contracts/yieldBarn.sol#174-177)

YieldBarn._claimRewards(uint256) (contracts/yieldBarn.sol#201-223) uses timestamp for comparisons

Dangerous comparisons:

- require(bool,string)(_msgSender() == stake.owner,Not the owner) (contracts/ yieldBarn.sol#204)

- require(bool,string)

(users[_msgSender()].activeStakes.contains(_stakeID),Invalid stake number) (contracts/ yieldBarn.sol#205-208)

- require(bool,string)(LP.balanceOf(address(this)) >=

pendingRewards,Insufficient LP balance) (contracts/yieldBarn.sol#211-214)

YieldBarn.endAllStakes() (contracts/yieldBarn.sol#233-242) uses timestamp for comparisons

Dangerous comparisons:

- block.timestamp >= stakes[stakeID].stakeEndTime (contracts/yieldBarn.sol#238)

YieldBarn.updateAccLpPerCrop() (contracts/yieldBarn.sol#254-268) uses timestamp for comparisons

Dangerous comparisons:

- block.timestamp <= lastRewardTime (contracts/yieldBarn.sol#255)

YieldBarn.getMultiplier(uint256,uint256) (contracts/yieldBarn.sol#270-281) uses timestamp for comparisons

Dangerous comparisons:

- _to <= endTime (contracts/yieldBarn.sol#274)

- _from >= endTime (contracts/yieldBarn.sol#276)

YieldBarn._injectRewardsWithTime(uint256,uint256) (contracts/yieldBarn.sol#292-311) uses timestamp for comparisons

Dangerous comparisons:

- remainingBal > 0 (contracts/yieldBarn.sol#301)

- block.timestamp >= startTime (contracts/yieldBarn.sol#306)

- endTime > block.timestamp (contracts/yieldBarn.sol#297-299)

YieldBarn.pendingLPs(uint256) (contracts/yieldBarn.sol#324-347) uses timestamp for comparisons

Dangerous comparisons:

- block.timestamp > lastRewardTime && _totalCrops != 0 (contracts/ yieldBarn.sol#336)

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp>

INFO:Detectors:

PreSale.setRegisterPublic(address,bool) (contracts/presale.sol#67-77) compares to a boolean constant:

-require(bool,string)(user.preRegistered == false,cant public register, pre

registered wallets) (contracts/presale.sol#70)

PreSale.addRegisterMultiplePublic(address[]) (contracts/presale.sol#79-87) compares to a boolean constant:

-require(bool,string)(user.preRegistered == false,cant public register, pre registered wallets) (contracts/presale.sol#83)

PreSale.setRegisterPre(address,bool) (contracts/presale.sol#89-94) compares to a boolean constant:

-require(bool,string)(user.publicRegistered == false,cant pre register, public registered wallets) (contracts/presale.sol#92)

PreSale.addRegisterMultiplePre(address[]) (contracts/presale.sol#96-103) compares to a boolean constant:

-require(bool,string)(user.publicRegistered == false,Cannot pre register, public registered wallets) (contracts/presale.sol#100)

PreSale.drawWinner() (contracts/presale.sol#117-136) compares to a boolean constant:

-require(bool,string)(winnerDrawn == false,Only one winner) (contracts/presale.sol#118)

PreSale.Buy() (contracts/presale.sol#152-184) compares to a boolean constant:

-require(bool,string)(end == false,presale is stopped) (contracts/presale.sol#157)

PreSale.Buy() (contracts/presale.sol#152-184) compares to a boolean constant:

-require(bool,string)(user.once == false,Only one time allowed) (contracts/presale.sol#159)

PreSale.Buy() (contracts/presale.sol#152-184) compares to a boolean constant:

-user.preRegistered == false && user.publicRegistered == false (contracts/presale.sol#170)

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality>

INFO:Detectors:

CowTipFarm.updatePool(uint256) (contracts/cowTipFarm.sol#314-359) has costly operations inside a loop:

- totalAllocPoint = totalAllocPoint.add(pool.allocPoint) (contracts/cowTipFarm.sol#321)

YieldBarn.updateAccLpPerCrop() (contracts/yieldBarn.sol#254-268) has costly operations inside a loop:

- lastRewardTime = block.timestamp (contracts/yieldBarn.sol#259)

YieldBarn.updateAccLpPerCrop() (contracts/yieldBarn.sol#254-268) has costly operations inside a loop:

- accLpPerCrop = accLpPerCrop.add(tokenReward.mul(1e18).div(totalCrops)) (contracts/yieldBarn.sol#264-266)

YieldBarn.updateAccLpPerCrop() (contracts/yieldBarn.sol#254-268) has costly operations inside a loop:

- lastRewardTime = block.timestamp (contracts/yieldBarn.sol#267)
 YieldBarn._endStake(uint256) (contracts/yieldBarn.sol#160-199) has costly operations inside a loop:

- totalCrops -= crop (contracts/yieldBarn.sol#181)
 YieldBarn._endStake(uint256) (contracts/yieldBarn.sol#160-199) has costly operations inside a loop:

- totalCOWTIPStaked -= amount (contracts/yieldBarn.sol#195)
 YieldBarn._endStake(uint256) (contracts/yieldBarn.sol#160-199) has costly operations inside a loop:

- totalLPCLaimed += pendingRewards (contracts/yieldBarn.sol#196)
 YieldBarn._claimRewards(uint256) (contracts/yieldBarn.sol#201-223) has costly operations inside a loop:

- totalLPCLaimed += pendingRewards (contracts/yieldBarn.sol#221)
 Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#costly-operations-inside-a-loop>

INFO:Detectors:

PreSale.publicStartTime (contracts/presale.sol#40) is set pre-construction with a non-constant function or state variable:

- 1800 + presaleStartTime

PreSale.fcfsStartTime (contracts/presale.sol#41) is set pre-construction with a non-constant function or state variable:

- 3600 + publicStartTime

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#function-initializing-state>

INFO:Detectors:

solc-0.8.20 is not recommended for deployment

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity>

INFO:Detectors:

Pragma version0.8.19 (contracts/CommunityFund.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.8.18.

Pragma version0.8.19 (contracts/StakingVault.sol#4) necessitates a version too recent to be trusted. Consider deploying with 0.8.18.

Pragma version0.8.19 (contracts/cowTip.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.8.18.

Pragma version0.8.19 (contracts/cowTipFarm.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.8.18.

Pragma version0.8.19 (contracts/distributor.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.8.18.

Pragma version0.8.19 (contracts/presale.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.8.18.

Pragma version0.8.19 (contracts/presaleDistributor.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.8.18.

Pragma version0.8.19 (contracts/yieldBarn.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.8.18.

solc-0.8.19 is not recommended for deployment

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity>

INFO:Detectors:

Low level call in CommunityFund.sendCustomTransaction(address,uint256,string,bytes) (contracts/CommunityFund.sol#44-68):

- (success,returnData) = target.call{value: value}(callData) (contracts/CommunityFund.sol#60-62)

Low level call in Distributor.sendCustomTransaction(address,uint256,string,bytes) (contracts/distributor.sol#104-128):

- (success,returnData) = target.call{value: value}(callData) (contracts/distributor.sol#120-122)

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls>

INFO:Detectors:

CowTip (contracts/cowTip.sol#18-179) should inherit from ICOWTIP (contracts/cowTipFarm.sol#35-37)

YieldBarn (contracts/yieldBarn.sol#13-401) should inherit from IStaking (contracts/StakingVault.sol#15-17)

YieldBarn (contracts/yieldBarn.sol#13-401) should inherit from IyieldBarn (contracts/presaleDistributor.sol#27-29)

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#missing-inheritance>

INFO:Detectors:

Loop condition `i < poolInfo.length` (contracts/cowTipFarm.sol#647) should use cached array length instead of referencing `length` member of the storage array.

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#cache-array-length>

INFO:Detectors:

Distributor.cowTipAmount (contracts/distributor.sol#11) should be constant

PreSale.cowtipPrice (contracts/presale.sol#42) should be constant

PresaleDistributor.runtime (contracts/presaleDistributor.sol#40) should be constant

PresaleDistributor.stakeType (contracts/presaleDistributor.sol#36) should be constant

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant>

INFO:Detectors:

CowTipFarm.cowTip (contracts/cowTipFarm.sol#92) should be immutable

```
CowTipFarm.poolStartTime (contracts/cowTipFarm.sol#108) should be immutable
PresaleDistributor.cowtip (contracts/presaleDistributor.sol#37) should be immutable
PresaleDistributor.endTime (contracts/presaleDistributor.sol#39) should be immutable
PresaleDistributor.presale (contracts/presaleDistributor.sol#34) should be immutable
PresaleDistributor.startTime (contracts/presaleDistributor.sol#38) should be immutable
PresaleDistributor.yieldBarn (contracts/presaleDistributor.sol#35) should be immutable
StakingVault.staking (contracts/StakingVault.sol#28) should be immutable
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-immutable
```

INFO:Detectors:

```
The function CowTip._swapCOWTIP(uint256) (contracts/cowTip.sol#69-85) reads allowance =
this.allowance(address(this),address(ROUTER)) (contracts/cowTip.sol#74) with `this`
which adds an extra STATICCALL.
```

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#public-variable-read-in-external-context>

INFO:Slither:. analyzed (61 contracts with 86 detectors), 142 result(s) found

