# Goat Finance Contracts Security Review

Version 1.0

feb 10, 2024

Conducted by:
**MaslarovK and Solthodox**, Independent Security Researchers

# Table of Contents

# 1 About Solthodox

Solthodox is a smart contract developer and independent security researcher experienced in Solidity smart contract development and transitioning to security. With +1 year of experience in the development side, he has been joining security contests in the last few months. He also serves as a smart contract developer at Unlockd Finance, where he has been involved in building defi yield farming strategies to maximze the APY of it's users.

# 2 About MaslarovK

MaslarovK is an independent security researcher from Bulgaria with 3 years of experience in Web2 development. His curiosity and love for decentralisation and transparency made him transition to Web3. He has secured various protocols through public contests and private audits.

# 3 Disclaimer

Audits are a time, resource, and expertise bound effort where trained experts evaluate smart contracts using a combination of automated and manual techniques to identify as many vulnerabilities as possible. Audits can show the presence of vulnerabilities **but not their absence**.

# 4 Risk classification

| Severity | Impact: High | Impact: Medium | Impact: Low |
|---|---|---|---|
| Likelihood: High | Critical | High | Medium |
| Likelihood: Medium | High | Medium | Low |
| Likelihood: Low | Medium | Low | Low |

## 4.1 Impact

- **High** - leads to a significant loss of assets in the protocol or significantly harms a group of users.
- **Medium** - only a small amount of funds can be lost or a functionality of the protocol is affected.
- **Low** - any kind of unexpected behaviour that's not so critical.

## 4.2 Likelihood

- **High** - direct attack vector; the cost is relatively low to the amount of funds that can be lost.
- **Medium** - only conditionally incentivized attack vector, but still relatively likely.
- **Low** - too many or too unlikely assumptions; provides little or no incentive.

### 4.3 Actions required by severity level

- **Critical** - client **must** fix the issue.
- **High** - client **must** fix the issue.
- **Medium** - client **should** fix the issue.
- **Low** - client **could** fix the issue.

# 5 Executive summary

**Overview**

| | |
|---|---|
| Project | Goat Finance Contracts |
| Repository | https://github.com/goatfi/contracts/tree/main |
| Commit hash | 097fc762503cb61afdb80a06e8c6338e2275ec95 |
| Reslution | 724d610f4f5d7bb9abf0b965a66cbf0ec809953b |
| Documentation | https://docs.goat.fi/ |
| Methods | Manual review & testing |

**Scope**

| |
|---|
| src/infra/GoatFeeBatch.sol |
| src/infra/GoatRewardPool.sol |

**Issues Found**

| | |
|---|---|
| Critical risk | 0 |
| High risk | 0 |
| Medium risk | 2 |
| Low risk | 2 |
| Informational | 7 |

# 6 Findings

## 6.1 Medium risk

### 6.1.1 New rewards pool lacks approval

**Severity:** *Medium risk*

**Context:** GoatFeeBatch.sol#L160

**Description:** When the reward pool is initialized through the constructor, an approve is made.

```
constructor(
        address _native,
        address _rewardPool,
        address _treasury,
        uint256 _treasuryFee
    ) Ownable(msg.sender) {
        native = IERC20(_native);
        treasury = _treasury;
        rewardPool = _rewardPool;
        treasuryFee = _treasuryFee;
        native.forceApprove(rewardPool, type(uint).max);
        duration = 7 days;
    }
```

But when it is done through the `setRewardPool` function, there is no approve:

```
function setRewardPool(address _rewardPool) external onlyOwner {
        rewardPool = _rewardPool;
        emit SetRewardPool(_rewardPool);
 }
```

This would result in the DoS of the harvest logic since the reward pool performs a `transferFrom` everytime and without any allowance it will always revert.

**Recommendation:** Perform an approval when the reward pool is changed:

```
function setRewardPool(address _rewardPool) external onlyOwner {
        rewardPool = _rewardPool;
        native.forceApprove(rewardPool, type(uint).max);
        emit SetRewardPool(_rewardPool);
 }
```

**Resolution:** Resolved

### 6.1.2 No zero address check can result in permanent loss of funds

**Severity:** *Medium risk*

**Context:** GoatFeeBatch.sol#L167

**Description:** In the `setTreasury` function there is no zero address check. If the _treasury is wrongly set to the address(0), the tokens meant to go to the treasury would be burned forever, since anyone can harvest and when transferring the fees to the treasury it won't revert in mainnet. Mainnet WETH does not revert on `address(0)` transfers.

```
function setTreasury(address _treasury) external onlyOwner {
        treasury = _treasury;
        emit SetTreasury(_treasury);
    }
```

**Recommendation:** Implement the following check:

```
function setTreasury(address _treasury) external onlyOwner {
        if (_treasury == address(0)) revert InvalidZeroAddress();
        treasury = _treasury;
        emit SetTreasury(_treasury);
    }
```

**Resolution:** Resolved

## 6.2 Low risk

### 6.2.1 Ether cant be rescued

**Severity:** *Low risk*

**Context:** GoatFeeBatch.sol#L209

**Description:** The `rescueTokens` function intends to rescue any unsupported tokens locked in the contract, but can only handle ERC20 tokens.

```
    function rescueTokens(address _token, address _recipient) external onlyOwner {
        if(_token == address(native)) revert WithdrawingRewardToken();

       // uses ERC20 interface for the interactions
        uint256 amount = IERC20(_token).balanceOf(address(this));
        IERC20(_token).safeTransfer(_recipient, amount);
        emit RescueTokens(_token, _recipient);
    }
```

Therefore, if some unwrapped native assets are sent to the contract the contract will receive them, but they will be stuck in the contract.

```
// the contract can receive ETH
 receive() external payable {}
```

**Recommendation:** Use some specific `_token` address to refer to the unwrapped assets so the contract can handle that scenario too:

```
    function rescueTokens(address _token, address _recipient) external onlyOwner {
        // use address(0) for example
        if(_token == address(0)){
            (bool success, ) = _recipient.call{value : address(this).balance}("")
            if(!success) = revert FailedToSendEther();
        }
        if(_token == address(native)) revert WithdrawingRewardToken();

        uint256 amount = IERC20(_token).balanceOf(address(this));
        IERC20(_token).safeTransfer(_recipient, amount);
        emit RescueTokens(_token, _recipient);
    }
```

**Resolution:** Resolved

### 6.2.2 Rewards may be lost forever if there is not enough balance

**Severity:** *Low risk*

**Context:** GoatRewardPool.sol#L361)

**Description:** In the `_getReward` function, if the `rewardEarned > 0`, the `earned` for the user is set to 0 and then the rewards are distributed, as it should be:

```
function _getReward() private {
        uint256 rewardLength = rewards.length;
        for (uint i; i < rewardLength;) {
            address reward = rewards[i];
            uint256 rewardEarned = _earned(msg.sender, reward);
            if (rewardEarned > 0) {
                _getRewardInfo(reward).earned[msg.sender] = 0;
                _rewardTransfer(reward, msg.sender, rewardEarned);
                emit RewardPaid(msg.sender, reward, rewardEarned);
            }
            unchecked { ++i; }
        }
    }
```

but there is a problem in the `_rewardTransfer` function. If the reward balance is less than the earned amount, the amount is set to the balance, which means only what is present in the contract will be distributed. Given the state change before the transfer, this will result in lost funds forever.

```
function _rewardTransfer(address _reward, address _recipient, uint256 _amount)
    private {
        uint256 rewardBal = IERC20(_reward).balanceOf(address(this));
        if (_amount > rewardBal) _amount = rewardBal;
        if (_amount > 0) IERC20(_reward).safeTransfer(_recipient, _amount);
    }
```

**Recommendation:** Reduce the `_getRewardInfo(reward).earned[msg.sender]`by the actual amount that was transferred to him, instead of setting it to zero. This way, even there was not enought rewards for the user to claim at that time, if more rewards are sent to the contract later the user will be able to claim the rest of the rewards owed to him.

```
// the function returns the actual amount transferred
 function _rewardTransfer(address _reward, address _recipient, uint256 _amount)
     private returns(uint256){
        uint256 rewardBal = IERC20(_reward).balanceOf(address(this));
        if (_amount > rewardBal) _amount = rewardBal;
        if (_amount > 0) IERC20(_reward).safeTransfer(_recipient, _amount);
        return amount;
    }
```

```
function _getReward() private {
        uint256 rewardLength = rewards.length;
        for (uint i; i < rewardLength;) {
            address reward = rewards[i];
            uint256 rewardEarned = _earned(msg.sender, reward);
            if (rewardEarned > 0) {
                // catch the return value
```

```
                uint256 transferred = _rewardTransfer(reward, msg.sender,
                    rewardEarned);
                // substract instead of setting to 0
                _getRewardInfo(reward).earned[msg.sender] -= transferred;
                emit RewardPaid(msg.sender, reward, rewardEarned);
            }
            unchecked { ++i; }
        }
    }
```

**Resolution:** Resolved

## 6.3 Informational

### 6.3.1 Consider renaming the `DIVISOR` to `FEE_DENOMINATOR`

**Severity:** *Informational*

**Context:** GoatFeeBatch.sol#L31

**Description:** Consider renaming the `DIVIDOR` to a more standard and intuitive `FEE_DENOMINATOR` (e.g. in Curve.fi)

**Recommendation:**
Consider calling it `FEE_DENOMINATOR`

**Resolution:** Resolved

### 6.3.2 Suboptimal name `native`

**Severity:** *Informational*

**Context:** GoatFeeBatch.sol#L16

**Description:** Generally speaking, `native` is understood as the native assets of a chain, not a ERC20 wrapper token in this case.

**Recommendation:**
Consider renaming it to a more intuitive `wrappedNative`.

**Resolution:** Resolved

### 6.3.3 Unconventional input validation

**Severity:** *Informational*

**Context:** GoatFeeBatch.sol#L190

**Description:** The `setTreasuryFee` function modifies the `treasuryFee` to a fixed value if the parameter `_treasuryFee` is invalid. This unconventional approach could result in undesired behaviour.

```
    function setTreasuryFee(uint256 _treasuryFee) external onlyOwner {
        // if invalid set it to MAX_TREASURY_FEE
        if (_treasuryFee > MAX_TREASURY_FEE) _treasuryFee = MAX_TREASURY_FEE;
        treasuryFee = _treasuryFee;
```

```
        emit SetTreasuryFee(_treasuryFee);
    }
```

**Recommendation:**
Consider reverting when the parameter `_treasuryFee` is invalid.

```
    function setTreasuryFee(uint256 _treasuryFee) external onlyOwner {
        // if invalid set it to MAX_TREASURY_FEE
        if (_treasuryFee > MAX_TREASURY_FEE) revert InvalidTreasuryFee(_treasuryFee)
            ;
        treasuryFee = _treasuryFee;
        emit SetTreasuryFee(_treasuryFee);
    }
```

**Resolution:** Resolved

#### 6.3.4 Typo in `harvesterMax`

**Severity:** *Informational*

**Context:** GoatFeeBatch.sol#L39

**Description:** The comment in the code says this variable represents the minimum operating has level for the harvester, and its treated as such in the flow of the contract, but the variable name is `harvesterMax`.

```
    /// @notice Minimum operating gas level on the harvester
    uint256 public harvesterMax;
```

**Recommendation:**
Consider naming it `minHarvesterGas` or `minHarvesterOperatingGas`.

**Resolution:** Resolved

#### 6.3.5 Inconsistent representation of `uint256`

**Severity:** *Informational*

**Context:** GoatFeeBatch.sol#L107

**Description:** While in the contract the type is represented as **`uint256`** it is represented as **`uint`** in one line:

```
    native.forceApprove(rewardPool, type(uint).max);
```

**Recommendation:**
Consider representing it as **`uint256`** for more consistency and readability.

**Resolution:** Resolved

### 6.3.6 Consider using a math library for earnings calculations

**Severity:** *Informational*

**Context:** GoatRewardPool.sol#L378

**Description:** By using a math library precision of the reward calculations in `GoatRewardPool`can be improved.

**Recommendation:**
Consider using OpenZeppelin's Math library with 30 decimals of precision instead of 18 to avoid "phantom overflow" and improve precision;

```
import {Math} from "openzeppelin/contracts/utils/math/Math.sol";
//...
function _rewardPerToken(address _reward) private view returns (uint256
    rewardPerToken) {
      RewardInfo storage rewardData = _getRewardInfo(_reward);
      if (totalSupply() == 0) {
          rewardPerToken = rewardData.rewardPerTokenStored;
      } else {
          rewardPerToken = rewardData.rewardPerTokenStored + Math.mulDiv(
              (_lastTimeRewardApplicable(rewardData.periodFinish) - rewardData.
                  lastUpdateTime),
              rewardData.rate* 1e30,
              totalSupply()
          );
      }
}
//...
function _earned(address _user, address _reward) private view returns (uint256
    earnedAmount) {
      RewardInfo storage rewardData = _getRewardInfo(_reward);
      earnedAmount = rewardData.earned[_user] + Math.mulDiv(
          balanceOf(_user) ,
          (_rewardPerToken(_reward) - rewardData.userRewardPerTokenPaid[_user]),
          1e30
      );
}
```

**Resolution:** Resolved

### 6.3.7 Unnecessary substraction in `harvest`

**Severity:** *Informational*

**Context:** GoatFeeBatch.sol#L378

**Description:** In the `harvest` function in `GoatFeeBatch` the `Harvest` event is emitted, where the total harvested amount is logged. To calculate so it substracts the native balace to the `totalFee` value. This is unnecessary because the native balance at that point will always be zero.

```
   function harvest() external {
       uint256 totalFees = native.balanceOf(address(this));
```

```
        if (sendHarvesterGas) _sendHarvesterGas();
        _distributeTreasuryFee();
        // _notifyRewardPool will take the remaining native balance
        _notifyRewardPool();
        // native.balanceOf(address(this)) will always be 0 at this point
        emit Harvest(totalFees - native.balanceOf(address(this)), block.timestamp);
    }
```

**Recommendation:**
Remove the unnecessary substraction

```
    function harvest() external {
        uint256 totalFees = native.balanceOf(address(this));

        if (sendHarvesterGas) _sendHarvesterGas();
        _distributeTreasuryFee();
        _notifyRewardPool();

        emit Harvest(totalFees, block.timestamp);
    }
```

**Resolution:** Resolved