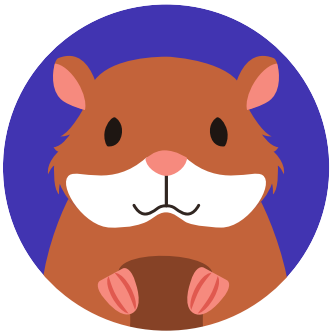
DEFIMOON PROJECT
AUDIT AND
DEVELOPMENT

CONTACTS
HTTPS://DEFIMOON.ORG
AUDIT@DEFIMOON.ORG
TELEGRAM
TWITTER

REPORT SMART CONTRACT AUDIT

HAMSTER.MONEY
PROJECT



FEBRUARY ,2022

AUDITOR
TEAM "DEFIMOON"

APPROVED BY
CYRILL MINYAEV, TEAM "DEFIMOON"

DEFIMOON
be secure

AUDIT OF PROJECT

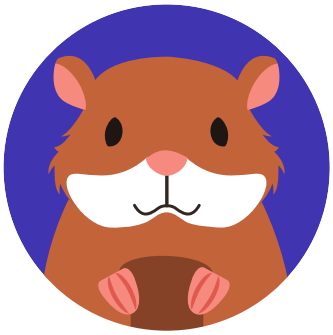
HTTPS://HAMSTER.MONEY/

HTTPS://GITHUB.COM/HAMSTER-MONEY/HAMSTER-CONTRACTS/TREE/MAIN/CONTRACTS

COMPARISON WITH

HTTPS://APP.BOMB.MONEY/

HTTPS://GITHUB.COM/BOMBMONEY



Hamster project

Operator contract — owner privileges

View functions

- `isOperator`
- `operator`

Privileged executables

- `transferOperator`

Features

```
    function isOperator() public view returns (bool) {
        return _msgSender() == _operator;
    }
```

Checks whether the invoker is the owner and has privileges

```
    function operator() public view returns (address) {
        return _operator;
    }
```

Queries curent owner of the contract

```
    function transferOperator(address newOperator_) public onlyOwner {
        _transferOperator(newOperator_);
    }


    function _transferOperator(address newOperator_) internal {
        require(newOperator_ != address(0), "operator: zero address given
for new operator");
        emit OperatorTransferred(address(0), newOperator_);
        _operator = newOperator_;
    }
```

No discrepancies with the original code were found

TaxOfficeV2 contract — taxation privileges

View functions

- `taxRate`

Executables

- `addLiquidityETHTaxFree`
- `addLiquidityTaxFree`
- `taxFreeTransferFrom`

Privileged executables

- `disableAutoCalculateTax`
- `enableAutoCalculateTax`
- `excludeAddressFromTax`
- `includeAddressInTax`
- `setBurnThreshold`
- `setTaxCollectorAddress`
- `setTaxExclusionForAddress`
- `setTaxRate`
- `setTaxTiersRate`
- `setTaxTiersTwap`
- `setTaxableHamsterOracle`
- `transferTaxOffice`

Features

```
    function taxRate() external view returns (uint256) {
        return ITaxable(hamster).taxRate();
    }
```

Fetches current tax rate

```
    function addLiquidityETHTaxFree(
        uint256 amtHamster,
        uint256 amtHamsterMin,
        uint256 amtFtmMin
    )
        external
        payable
        returns (
            uint256,
            uint256,
            uint256
        )
    {
```

```
        require(amtHamster != 0 && msg.value != 0, "amounts can't be 0");
        uint256 resultAmtHamster;
        uint256 resultAmtFtm;
        uint256 liquidity;
        _excludeAddressFromTax(msg.sender);
        IERC20(hamster).transferFrom(msg.sender, address(this),
amtHamster);
        _approveTokenIfNeeded(hamster, router);
        _includeAddressInTax(msg.sender);
        (resultAmtHamster, resultAmtFtm, liquidity) =
IUniswapV2Router(router).addLiquidityETH{value: msg.value}(
            hamster,
            amtHamster,
            amtHamsterMin,
            amtFtmMin,
            msg.sender,
            block.timestamp
        );
        if (amtHamster.sub(resultAmtHamster) > 0) {
            IERC20(hamster).transfer(msg.sender,
amtHamster.sub(resultAmtHamster));
        }
        return (resultAmtHamster, resultAmtFtm, liquidity);
    }


    function addLiquidityTaxFree(
        address token,
        uint256 amtHamster,
        uint256 amtToken,
        uint256 amtHamsterMin,
        uint256 amtTokenMin
    )
        external
        returns (
            uint256,
            uint256,
            uint256
        )
    {
        require(amtHamster != 0 && amtToken != 0, "amounts can't be 0");
        uint256 resultAmtHamster;
        uint256 resultAmtToken;
        uint256 liquidity;
        _excludeAddressFromTax(msg.sender);
        IERC20(hamster).transferFrom(msg.sender, address(this),
amtHamster);
        IERC20(token).transferFrom(msg.sender, address(this), amtToken);
        _approveTokenIfNeeded(hamster, router);
        _approveTokenIfNeeded(token, router);
        _includeAddressInTax(msg.sender);
        (resultAmtHamster, resultAmtToken, liquidity) =
```

```
IUniswapV2Router(router).addLiquidity(
            hamster,
            token,
            amtHamster,
            amtToken,
            amtHamsterMin,
            amtTokenMin,
            msg.sender,
            block.timestamp
        );
        if (amtHamster.sub(resultAmtHamster) > 0) {
            IERC20(hamster).transfer(msg.sender,
amtHamster.sub(resultAmtHamster));
        }
        if (amtToken.sub(resultAmtToken) > 0) {
            IERC20(token).transfer(msg.sender,
amtToken.sub(resultAmtToken));
        }
        return (resultAmtHamster, resultAmtToken, liquidity);
    }
```

Allows to provide liquidity without paying the tax

```
 function taxFreeTransferFrom(
        address _sender,
        address _recipient,
        uint256 _amt
    ) external {
        require(taxExclusionEnabled[msg.sender], "Address not approved for
tax free transfers");
        _excludeAddressFromTax(_sender);
        IERC20(hamster).transferFrom(_sender, _recipient, _amt);
        _includeAddressInTax(_sender);
    }
```

Allows to conduct tex-free transactions

```
    function disableAutoCalculateTax() external onlyOperator {
        ITaxable(hamster).disableAutoCalculateTax();
    }


    function enableAutoCalculateTax() external onlyOperator {
        ITaxable(hamster).enableAutoCalculateTax();
    }
```

Privileged
Toggles auto tax calculation

```
    function excludeAddressFromTax(address _address) external onlyOperator
returns (bool) {
        return _excludeAddressFromTax(_address);
    }


    function includeAddressInTax(address _address) external onlyOperator
returns (bool) {
        return _includeAddressInTax(_address);
    }
```

Privileged
Allows to exclude/include addresses from taxation

```
    function setBurnThreshold(uint256 _burnThreshold) external onlyOperator
{
        ITaxable(hamster).setBurnThreshold(_burnThreshold);
    }
```

Privileged
Specifies the burn threshold

```
    function setTaxCollectorAddress(address _taxCollectorAddress) external
onlyOperator {
        ITaxable(hamster).setTaxCollectorAddress(_taxCollectorAddress);
    }
```

Privileged
Specifies the address for tax collection

```
    function setTaxRate(uint256 _taxRate) external onlyOperator {
        ITaxable(hamster).setTaxRate(_taxRate);
    }
```

```
    function setTaxTiersRate(uint8 _index, uint256 _value) external
onlyOperator returns (bool) {
        return ITaxable(hamster).setTaxTiersRate(_index, _value);
    }


    function setTaxTiersTwap(uint8 _index, uint256 _value) external
onlyOperator returns (bool) {
        return ITaxable(hamster).setTaxTiersTwap(_index, _value);
    }
```

Privileged
Allows to modify taxation parameters

```
    function setTaxableHamsterOracle(address _hamsterOracle) external
onlyOperator {
        ITaxable(hamster).setHamsterOracle(_hamsterOracle);
    }
```

Privileged
Specifies the address of the hamster Oracle

```
        function transferTaxOffice(address _newTaxOffice) external
onlyOperator {
        ITaxable(hamster).setTaxOffice(_newTaxOffice);
    }
```

Privileged
Allows to transfer the TaxOffice privileges

No discrepancies with the original code were found

HShareRewardPool contract

View functions

- `pendingShare`
- `getGeneratedReward`
- `checkPoolDuplicate`

Executables

- `add`
- `deposit`
- `emergencyWithdraw`
- `governanceRecoverUnsupported`
- `set`
- `withdraw`
- `massUpdatePools`
- `updatePool`

Features

```
    function pendingShare(uint256 _pid, address _user) external view
returns (uint256) {
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][_user];
        uint256 accHSharePerShare = pool.accHSharePerShare;
        uint256 tokenSupply = pool.token.balanceOf(address(this));
        if (block.timestamp > pool.lastRewardTime && tokenSupply != 0) {
            uint256 _generatedReward =
getGeneratedReward(pool.lastRewardTime, block.timestamp);
            uint256 _hshareReward =
_generatedReward.mul(pool.allocPoint).div(totalAllocPoint);
            accHSharePerShare =
accHSharePerShare.add(_hshareReward.mul(1e18).div(tokenSupply));
        }
        return
user.amount.mul(accHSharePerShare).div(1e18).sub(user.rewardDebt);
    }
```

Returns pending pool share for the given user

```
    function getGeneratedReward(uint256 _fromTime, uint256 _toTime) public
view returns (uint256) {
        if (_fromTime >= _toTime) return 0;
        if (_toTime >= poolEndTime) {
            if (_fromTime >= poolEndTime) return 0;
            if (_fromTime <= poolStartTime) return
poolEndTime.sub(poolStartTime).mul(hSharePerSecond);
            return poolEndTime.sub(_fromTime).mul(hSharePerSecond);
        } else {
            if (_toTime <= poolStartTime) return 0;
            if (_fromTime <= poolStartTime) return
_toTime.sub(poolStartTime).mul(hSharePerSecond);
            return _toTime.sub(_fromTime).mul(hSharePerSecond);
        }
    }
```

Returns accumulated rewards over the time given from `_fromTime` to `_toTime` block

---

```
    function checkPoolDuplicate(IERC20 _token) internal view {
        uint256 length = poolInfo.length;
        for (uint256 pid = 0; pid < length; ++pid) {
            require(poolInfo[pid].token != _token, "HShareRewardPool:
existing pool?");
        }
    }
```

Checks for the pool's duplicate

```
    function add(
        uint256 _allocPoint,
        IERC20 _token,
        bool _withUpdate,
        uint256 _lastRewardTime
    ) external onlyOperator {
        checkPoolDuplicate(_token);
        if (_withUpdate) {
            massUpdatePools();
        }
        if (block.timestamp < poolStartTime) {
            if (_lastRewardTime == 0) {
                _lastRewardTime = poolStartTime;
            } else {
                if (_lastRewardTime < poolStartTime) {
                    _lastRewardTime = poolStartTime;
```

```
                }
            }
        } else {
            if (_lastRewardTime == 0 || _lastRewardTime < block.timestamp)
{
                _lastRewardTime = block.timestamp;
            }
        }
        bool _isStarted =
        (_lastRewardTime <= poolStartTime) ||
        (_lastRewardTime <= block.timestamp);
        poolInfo.push(PoolInfo({
            token : _token,
            allocPoint : _allocPoint,
            lastRewardTime : _lastRewardTime,
            accHSharePerShare : 0,
            isStarted : _isStarted
            }));
        if (_isStarted) {
            totalAllocPoint = totalAllocPoint.add(_allocPoint);
        }
    }
```

Privileged

Adds new LP to the pool

```
    function deposit(uint256 _pid, uint256 _amount) external {
        address _sender = msg.sender;
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][_sender];
        updatePool(_pid);
        if (user.amount > 0) {
            uint256 _pending =
user.amount.mul(pool.accHSharePerShare).div(1e18).sub(user.rewardDebt);
            if (_pending > 0) {
                safeHShareTransfer(_sender, _pending);
                emit RewardPaid(_sender, _pending);
            }
        }
        if (_amount > 0) {
            pool.token.safeTransferFrom(_sender, address(this), _amount);
            user.amount = user.amount.add(_amount);
        }
        user.rewardDebt =
user.amount.mul(pool.accHSharePerShare).div(1e18);
        emit Deposit(_sender, _pid, _amount);
    }
```

Deposits LP tokens

```
    function emergencyWithdraw(uint256 _pid) external {
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][msg.sender];
        uint256 _amount = user.amount;
        user.amount = 0;
        user.rewardDebt = 0;
        pool.token.safeTransfer(msg.sender, _amount);
        emit EmergencyWithdraw(msg.sender, _pid, _amount);
    }
```

Allows to withdraw funds in an emergency, rewards may be lost

---

```
    function governanceRecoverUnsupported(IERC20 _token, uint256 amount,
address to) external onlyOperator {
        if (block.timestamp < poolEndTime + 30 days) {
            require(_token != hshare, "hshare");
            uint256 length = poolInfo.length;
            for (uint256 pid = 0; pid < length; ++pid) {
                PoolInfo storage pool = poolInfo[pid];
                require(_token != pool.token, "pool.token");
            }
        }
        _token.safeTransfer(to, amount);
    }
```

Privileged
Allows to recover tokens, unsupported by the platform

```
    function set(uint256 _pid, uint256 _allocPoint) external onlyOperator {
        massUpdatePools();
        PoolInfo storage pool = poolInfo[_pid];
        if (pool.isStarted) {
            totalAllocPoint = totalAllocPoint.sub(pool.allocPoint).add(
                _allocPoint
            );
        }
        pool.allocPoint = _allocPoint;
    }
```

Privileged
Update the given pool's allocation point

```
    function withdraw(uint256 _pid, uint256 _amount) external {
        address _sender = msg.sender;
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][_sender];
        require(user.amount >= _amount, "withdraw: not good");
        updatePool(_pid);
        uint256 _pending =
user.amount.mul(pool.accHSharePerShare).div(1e18).sub(user.rewardDebt);
        if (_pending > 0) {
            safeHShareTransfer(_sender, _pending);
            emit RewardPaid(_sender, _pending);
        }
        if (_amount > 0) {
            user.amount = user.amount.sub(_amount);
            pool.token.safeTransfer(_sender, _amount);
        }
        user.rewardDebt = user.amount.mul(pool.accHSharePerShare).div(1e18);
        emit Withdraw(_sender, _pid, _amount);
    }
```

Allows to withdraw LP tokens

```
    function massUpdatePools() public {
        uint256 length = poolInfo.length;
        for (uint256 pid = 0; pid < length; ++pid) {
            updatePool(pid);
        }
    }
```

Updates reward variables for all pools
Be careful of gas spending — with a large `pid` value this could become very expensive

---

```
    function updatePool(uint256 _pid) public {
        PoolInfo storage pool = poolInfo[_pid];
        if (block.timestamp <= pool.lastRewardTime) {
            return;
```

```
        }
        uint256 tokenSupply = pool.token.balanceOf(address(this));
        if (tokenSupply == 0) {
            pool.lastRewardTime = block.timestamp;
            return;
        }
        if (!pool.isStarted) {
            pool.isStarted = true;
            totalAllocPoint = totalAllocPoint.add(pool.allocPoint);
        }
        if (totalAllocPoint > 0) {
            uint256 _generatedReward =
getGeneratedReward(pool.lastRewardTime, block.timestamp);
            uint256 _hshareReward =
_generatedReward.mul(pool.allocPoint).div(totalAllocPoint);
            pool.accHSharePerShare =
pool.accHSharePerShare.add(_hshareReward.mul(1e18).div(tokenSupply));
        }
        pool.lastRewardTime = block.timestamp;
    }
```

Updates reward variables of the given pool

No discrepancies with the original code were found

HamsterGenesisRewardPool contract

View functions

- `comissionTokensCount`
- `comissionToken`
- `pendingHAMSTER`
- `getGeneratedReward`

Executables

- `add`
- `deposit`
- `emergencyWithdraw`

- `withdraw`
- `updatePool`
- `safeHamsterTransfer`

Features

```
    function comissionTokensCount() external view returns (uint256) {
        return _comissionTokens.length();
    }
```

Returns the number of comission tokens in circulation

```
    function comissionToken(uint256 index) external view returns (address)
{
        return _comissionTokens.at(index);
    }
```

Queries the address (info) of the specific token

```
    function pendingHAMSTER(uint256 _pid, address _user) external view
returns (uint256) {
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][_user];
        uint256 accHamsterPerShare = pool.accHamsterPerShare;
        uint256 tokenSupply = pool.token.balanceOf(address(this));
        if (block.timestamp > pool.lastRewardTime && tokenSupply != 0) {
            uint256 _generatedReward =
getGeneratedReward(pool.lastRewardTime, block.timestamp);
            uint256 _hamsterReward =
_generatedReward.mul(pool.allocPoint).div(totalAllocPoint);
            accHamsterPerShare =
accHamsterPerShare.add(_hamsterReward.mul(1e18).div(tokenSupply));
        }
        return
user.amount.mul(accHamsterPerShare).div(1e18).sub(user.rewardDebt);
    }
```

Allows to query pending hamster tokens

```
    function getGeneratedReward(uint256 _fromTime, uint256 _toTime) public
view returns (uint256) {
        if (_fromTime >= _toTime) return 0;
        if (_toTime >= poolEndTime) {
            if (_fromTime >= poolEndTime) return 0;
            if (_fromTime <= poolStartTime) return
poolEndTime.sub(poolStartTime).mul(hamsterPerSecond);
            return poolEndTime.sub(_fromTime).mul(hamsterPerSecond);
        } else {
            if (_toTime <= poolStartTime) return 0;
            if (_fromTime <= poolStartTime) return
_toTime.sub(poolStartTime).mul(hamsterPerSecond);
            return _toTime.sub(_fromTime).mul(hamsterPerSecond);
        }
    }
```

Returns accumulated rewards over the time given from `_fromTime` to `_toTime` block

---

```
    function checkPoolDuplicate(IERC20 _token) internal view {
        uint256 length = poolInfo.length;
        for (uint256 pid = 0; pid < length; ++pid) {
            require(poolInfo[pid].token != _token, "HamsterGenesisPool:
existing pool?");
        }
    }
```

Checks for the pool's duplicate

```
  function add(
      uint256 _allocPoint,
      IERC20 _token,
      bool _withUpdate,
      uint256 _lastRewardTime,
      uint256 _maxDeposit
  ) external onlyOperator {
      checkPoolDuplicate(_token);
      if (_withUpdate) {
          massUpdatePools();
      }
      if (block.timestamp < poolStartTime) {
          if (_lastRewardTime == 0) {
```

```
                _lastRewardTime = poolStartTime;
            } else {
                if (_lastRewardTime < poolStartTime) {
                    _lastRewardTime = poolStartTime;
                }
            }
        } else {
            if (_lastRewardTime == 0 || _lastRewardTime < block.timestamp) {
                _lastRewardTime = block.timestamp;
            }
        }
        bool _isStarted =
        (_lastRewardTime <= poolStartTime) ||
        (_lastRewardTime <= block.timestamp);
        poolInfo.push(PoolInfo({
            token : _token,
            allocPoint : _allocPoint,
            lastRewardTime : _lastRewardTime,
            accHamsterPerShare : 0,
            isStarted : _isStarted,
            maxDeposit : _maxDeposit
        }));
        if (_isStarted) {
            totalAllocPoint = totalAllocPoint.add(_allocPoint);
        }
    }
```

Privileged
Adds new LP to the pool

```
    function deposit(uint256 _pid, uint256 _amount) external {
        address _sender = msg.sender;
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][_sender];
        updatePool(_pid);
        if (user.amount > 0) {
            uint256 _pending =
user.amount.mul(pool.accHamsterPerShare).div(1e18).sub(user.rewardDebt);
            if (_pending > 0) {
                safeHamsterTransfer(_sender, _pending);
                emit RewardPaid(_sender, _pending);
            }
        }
        uint256 possibleToDeposit = pool.maxDeposit.sub(user.amount);
```

```
        uint256 amount = _amount > possibleToDeposit ? possibleToDeposit :
_amount;
        if (amount > 0) {
            pool.token.safeTransferFrom(_sender, address(this), amount);
            if (_comissionTokens.contains(address(pool.token))) {
                uint256 hundred = 10000;
                user.amount =
user.amount.add(amount.mul(hundred.sub(comissionPercent)).div(hundred));
            } else {
                user.amount = user.amount.add(amount);
            }
        }
        user.rewardDebt =
user.amount.mul(pool.accHamsterPerShare).div(1e18);
        emit Deposit(_sender, _pid, amount);
    }
```

Deposits LP tokens

```
    function emergencyWithdraw(uint256 _pid) external {
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][msg.sender];
        uint256 _amount = user.amount;
        user.amount = 0;
        user.rewardDebt = 0;
        pool.token.safeTransfer(msg.sender, _amount);
        emit EmergencyWithdraw(msg.sender, _pid, _amount);
    }
```

Allows to withdraw funds in an emergency, rewards may be lost

---

```
    function withdraw(uint256 _pid, uint256 _amount) external {
        address _sender = msg.sender;
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][_sender];
        require(user.amount >= _amount, "withdraw: not good");
        updatePool(_pid);
        uint256 _pending =
user.amount.mul(pool.accHamsterPerShare).div(1e18).sub(user.rewardDebt);
        if (_pending > 0) {
            safeHamsterTransfer(_sender, _pending);
            emit RewardPaid(_sender, _pending);
        }
        if (_amount > 0) {
            user.amount = user.amount.sub(_amount);
```

```
            pool.token.safeTransfer(_sender, _amount);
        }
        user.rewardDebt =
user.amount.mul(pool.accHamsterPerShare).div(1e18);
        emit Withdraw(_sender, _pid, _amount);
    }

    function massUpdatePools() public {
        uint256 length = poolInfo.length;
        for (uint256 pid = 0; pid < length; ++pid) {
            updatePool(pid);
        }
    }
```

Allows to withdraw tokens

```
    function updatePool(uint256 _pid) public {
        PoolInfo storage pool = poolInfo[_pid];
        if (block.timestamp <= pool.lastRewardTime) {
            return;
        }
        uint256 tokenSupply = pool.token.balanceOf(address(this));
        if (tokenSupply == 0) {
            pool.lastRewardTime = block.timestamp;
            return;
        }
        if (!pool.isStarted) {
            pool.isStarted = true;
            totalAllocPoint = totalAllocPoint.add(pool.allocPoint);
        }
        if (totalAllocPoint > 0) {
            uint256 _generatedReward =
getGeneratedReward(pool.lastRewardTime, block.timestamp);
            uint256 _hamsterReward =
_generatedReward.mul(pool.allocPoint).div(totalAllocPoint);
            pool.accHamsterPerShare =
pool.accHamsterPerShare.add(_hamsterReward.mul(1e18).div(tokenSupply));
        }
        pool.lastRewardTime = block.timestamp;
    }
```

Allows to update specific pool

```
    function safeHamsterTransfer(address _to, uint256 _amount) internal {
        uint256 _hamsterBalance = hamster.balanceOf(address(this));
```

```
        if (_hamsterBalance > 0) {
            if (_amount > _hamsterBalance) {
                hamster.safeTransfer(_to, _hamsterBalance);
            } else {
                hamster.safeTransfer(_to, _amount);
            }
        }
    }
```

Allows to safely transfer hamster tokens in case if rounding error causes the pool to not have enough tokens

---

No major/impactful discrepancies with the original code were found — in the original code, values for pool emmision, total rewards and running time are hardcoded, while here they are provided at contract deployment to the `constructor` function

---

HamsterRewardPool contract

View functions

- `pendingHAMSTER`
- `getGeneratedReward`

Executables

- `add`
- `deposit`
- `emergencyWithdraw`
- `governanceRecoverUnsupported`
- `set`
- `withdraw`
- `massUpdatePools`
- `updatePool`
- `safeHamsterTransfer`

Features

```
    function pendingHAMSTER(uint256 _pid, address _user) external view
returns (uint256) {
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][_user];
        uint256 accHamsterPerShare = pool.accHamsterPerShare;
        uint256 tokenSupply = pool.token.balanceOf(address(this));
        if (block.timestamp > pool.lastRewardTime && tokenSupply != 0) {
            uint256 _generatedReward =
getGeneratedReward(pool.lastRewardTime, block.timestamp);
            uint256 _hamsterReward =
_generatedReward.mul(pool.allocPoint).div(totalAllocPoint);
            accHamsterPerShare =
accHamsterPerShare.add(_hamsterReward.mul(1e18).div(tokenSupply));
        }
        return
user.amount.mul(accHamsterPerShare).div(1e18).sub(user.rewardDebt);
    }
```

Allows to query pending hamster tokens

```
    function getGeneratedReward(uint256 _fromTime, uint256 _toTime) public
view returns (uint256) {
        for (uint8 epochId = 2; epochId >= 1; --epochId) {
            if (_toTime >= epochEndTimes[epochId - 1]) {
                if (_fromTime >= epochEndTimes[epochId - 1]) {
                    return
_toTime.sub(_fromTime).mul(epochHamsterPerSecond[epochId]);
                }
                uint256 _generatedReward =
_toTime.sub(epochEndTimes[epochId -
1]).mul(epochHamsterPerSecond[epochId]);
                if (epochId == 1) {
                    return
_generatedReward.add(epochEndTimes[0].sub(_fromTime).mul(epochHamsterPerSec
ond[0]));
                }
                for (epochId = epochId - 1; epochId >= 1; --epochId) {
                    if (_fromTime >= epochEndTimes[epochId - 1]) {
                        return _generatedReward
                            .add(epochEndTimes[epochId]
                            .sub(_fromTime)
                            .mul(epochHamsterPerSecond[epochId]));
                    }
                    _generatedReward = _generatedReward
                        .add(epochEndTimes[epochId]
                        .sub(epochEndTimes[epochId - 1])
                        .mul(epochHamsterPerSecond[epochId]));
```

```
                }
                return
_generatedReward.add(epochEndTimes[0].sub(_fromTime).mul(epochHamsterPerSec
ond[0]));
            }
        }
        return _toTime.sub(_fromTime).mul(epochHamsterPerSecond[0]);
    }
```

Returns accumulated rewards over the time given from `_fromTime` to `_toTime` block

---

```
    function add(
        uint256 _allocPoint,
        IERC20 _token,
        bool _withUpdate,
        uint256 _lastRewardTime
    ) external onlyOperator {
        checkPoolDuplicate(_token);
        if (_withUpdate) {
            massUpdatePools();
        }
        if (block.timestamp < poolStartTime) {
            if (_lastRewardTime == 0) {
                _lastRewardTime = poolStartTime;
            } else {
                if (_lastRewardTime < poolStartTime) {
                    _lastRewardTime = poolStartTime;
                }
            }
        } else {
            if (_lastRewardTime == 0 || _lastRewardTime < block.timestamp)
{
                _lastRewardTime = block.timestamp;
            }
        }
        bool _isStarted = (_lastRewardTime <= poolStartTime) ||
(_lastRewardTime <= block.timestamp);
        poolInfo.push(PoolInfo({
            token: _token,
            allocPoint: _allocPoint,
            lastRewardTime: _lastRewardTime,
            accHamsterPerShare: 0,
            isStarted: _isStarted
        }));
        if (_isStarted) {
            totalAllocPoint = totalAllocPoint.add(_allocPoint);
        }
    }
```

Privileged

Adds new tokens to the pool

```
    function deposit(uint256 _pid, uint256 _amount) external {
        address _sender = msg.sender;
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][_sender];
        updatePool(_pid);
        if (user.amount > 0) {
            uint256 _pending =
user.amount.mul(pool.accHamsterPerShare).div(1e18).sub(user.rewardDebt);
            if (_pending > 0) {
                safeHamsterTransfer(_sender, _pending);
                emit RewardPaid(_sender, _pending);
            }
        }
        if (_amount > 0) {
            pool.token.safeTransferFrom(_sender, address(this), _amount);
            user.amount = user.amount.add(_amount);
        }
        user.rewardDebt =
user.amount.mul(pool.accHamsterPerShare).div(1e18);
        emit Deposit(_sender, _pid, _amount);
    }
```

Deposits new LP tokens to the pool

```
    function emergencyWithdraw(uint256 _pid) external {
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][msg.sender];
        uint256 _amount = user.amount;
        user.amount = 0;
        user.rewardDebt = 0;
        pool.token.safeTransfer(msg.sender, _amount);
        emit EmergencyWithdraw(msg.sender, _pid, _amount);
    }
```

Allows to withdraw funds in an emergency, rewards may be lost

---

```
    function governanceRecoverUnsupported(
        IERC20 _token,
        uint256 amount,
        address to
    ) external onlyOperator {
        if (block.timestamp < epochEndTimes[1] + 30 days) {
            require(_token != hamster, "!hamster");
            uint256 length = poolInfo.length;
            for (uint256 pid = 0; pid < length; ++pid) {
                PoolInfo storage pool = poolInfo[pid];
                require(_token != pool.token, "!pool.token");
            }
        }
        _token.safeTransfer(to, amount);
    }
```

Privileged
Allows to recover tokens, unsupported by the platform

```
    function set(uint256 _pid, uint256 _allocPoint) external onlyOperator {
        massUpdatePools();
        PoolInfo storage pool = poolInfo[_pid];
        if (pool.isStarted) {
            totalAllocPoint =
totalAllocPoint.sub(pool.allocPoint).add(_allocPoint);
        }
        pool.allocPoint = _allocPoint;
    }
```

Privileged
Updates the given pool's hamster allocation point

```
    function withdraw(uint256 _pid, uint256 _amount) external {
        address _sender = msg.sender;
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][_sender];
        require(user.amount >= _amount, "withdraw: not good");
        updatePool(_pid);
        uint256 _pending =
user.amount.mul(pool.accHamsterPerShare).div(1e18).sub(user.rewardDebt);
        if (_pending > 0) {
            safeHamsterTransfer(_sender, _pending);
            emit RewardPaid(_sender, _pending);
```

```
        }
        if (_amount > 0) {
            user.amount = user.amount.sub(_amount);
            pool.token.safeTransfer(_sender, _amount);
        }
        user.rewardDebt =
user.amount.mul(pool.accHamsterPerShare).div(1e18);
        emit Withdraw(_sender, _pid, _amount);
    }
```

Allows to withdraw funds from the pool

```
    function massUpdatePools() public {
        uint256 length = poolInfo.length;
        for (uint256 pid = 0; pid < length; ++pid) {
            updatePool(pid);
        }
    }
```

Updates reward variables for all pools
Be careful of gas spending — with a large `pid` value this could become very expensive

---

```
    function updatePool(uint256 _pid) public {
        PoolInfo storage pool = poolInfo[_pid];
        if (block.timestamp <= pool.lastRewardTime) {
            return;
        }
        uint256 tokenSupply = pool.token.balanceOf(address(this));
        if (tokenSupply == 0) {
            pool.lastRewardTime = block.timestamp;
            return;
        }
        if (!pool.isStarted) {
            pool.isStarted = true;
            totalAllocPoint = totalAllocPoint.add(pool.allocPoint);
        }
        if (totalAllocPoint > 0) {
            uint256 _generatedReward =
getGeneratedReward(pool.lastRewardTime, block.timestamp);
            uint256 _hamsterReward =
_generatedReward.mul(pool.allocPoint).div(totalAllocPoint);
            pool.accHamsterPerShare =
pool.accHamsterPerShare.add(_hamsterReward.mul(1e18).div(tokenSupply));
        }
```

```
        pool.lastRewardTime = block.timestamp;
    }
```

Updates reward variables of the given pool

```
    function safeHamsterTransfer(address _to, uint256 _amount) internal {
        uint256 _hamsterBal = hamster.balanceOf(address(this));
        if (_hamsterBal > 0) {
            if (_amount > _hamsterBal) {
                hamster.safeTransfer(_to, _hamsterBal);
            } else {
                hamster.safeTransfer(_to, _amount);
            }
        }
    }
```

Allows to safely transfer hamster tokens in case if rounding error causes the pool to not have enough tokens

---

No discrepancies with the original code were found

HBond contract

Executables

- `burn`
- `burnFrom`
- `mint`

Features

```
    function burn(uint256 amount) public override {
        super.burn(amount);
    }
```

Takes the `amount` of tokens out of circulation

```
    function burnFrom(address account, uint256 amount) public override
onlyOperator {
        super.burnFrom(account, amount);
    }
```

Privileged
Allows to destroy the `amount` of tokens on the specified address

```
    function mint(address recipient_, uint256 amount_) public onlyOperator
returns (bool) {
        uint256 balanceBefore = balanceOf(recipient_);
        _mint(recipient_, amount_);
        uint256 balanceAfter = balanceOf(recipient_);
        return balanceAfter > balanceBefore;
    }
```

Privileged
Issues new tokens to the `recipient_` address

No discrepancies with the original code were found

HShare contract

View functions

- `unclaimedDevFund`
- `unclaimedTreasuryFund`

Executables

- `claimRewards`

- `distributeReward`
- `setDevFund`
- `setTreasuryFund`

Features

```
    function unclaimedDevFund() public view returns (uint256 _pending) {
        uint256 _now = block.timestamp;
        if (_now > endTime) _now = endTime;
        if (devFundLastClaimed >= _now) return 0;
        _pending = _now.sub(devFundLastClaimed).mul(devFundRewardRate);
    }
```

Queries the amount of unclaimed tokens in the developer fund

```
    function unclaimedTreasuryFund() public view returns (uint256 _pending)
{
        uint256 _now = block.timestamp;
        if (_now > endTime) _now = endTime;
        if (communityFundLastClaimed >= _now) return 0;
        _pending =
_now.sub(communityFundLastClaimed).mul(communityFundRewardRate);
    }
```

Queries the amount of unclaimed tokens in the treasury fund

```
    function claimRewards() external {
        uint256 _pending = unclaimedTreasuryFund();
        if (_pending > 0 && communityFund != address(0)) {
            _mint(communityFund, _pending);
            communityFundLastClaimed = block.timestamp;
        }
        _pending = unclaimedDevFund();
        if (_pending > 0 && devFund != address(0)) {
            _mint(devFund, _pending);
            devFundLastClaimed = block.timestamp;
        }
    }
```

Claims pending rewards to community and developer fund

```
    function distributeReward(address _farmingIncentiveFund, uint256
_farmingPoolAllocation) external onlyOperator {
        require(!rewardPoolDistributed, "only can distribute once");
        require(_farmingIncentiveFund != address(0),
"!_farmingIncentiveFund");
        rewardPoolDistributed = true;
        _mint(_farmingIncentiveFund, _farmingPoolAllocation);
    }
```

Privileged
Reward distribution to the pool
Note, this could only be done once

```
    function setDevFund(address _devFund) external {
        require(msg.sender == devFund, "!dev");
        require(_devFund != address(0), "zero");
        devFund = _devFund;
    }
```

Changes the address of the developer fund

```
    function setTreasuryFund(address _communityFund) external {
        require(msg.sender == devFund, "!dev");
        communityFund = _communityFund;
    }
```

Changes the address of the treasury fund

No major/impactful discrepancies with the original code were found — in the original code, values for fund allocations (`_vestingDuration`, `_communityFund`, `_communityFundAllocation`, `_devFund`, `_devFundAllocation`) are hardcoded, while here they are provided at contract deployment to the `constructor` function

Unless it is reqired for those values to be initialized dynamically, it is suggested to hardcode them for improved project transparency

---

Hamster contract

View functions

- `isAddressExcluded`
- `getTaxTiersTwapsCount`
- `getTaxTiersRatesCount`

Executables

- `disableAutoCalculateTax`
- `enableAutoCalculateTax`
- `distributeReward`
- `includeAddress`
- `excludeAddress`
- `mint`
- `setBurnThreshold`
- `setHamsterOracle`
- `setTaxCollectorAddress`
- `setTaxOffice`
- `setTaxRate`
- `setTaxTiersRate`
- `setTaxTiersTwap`
- `transferFrom`

Features

```
    function isAddressExcluded(address _address) external view returns
(bool) {
        return excludedAddresses[_address];
    }
```

Check whether the address is excluded from tax

```
    function getTaxTiersTwapsCount() public view returns (uint256 count) {
        return taxTiersTwaps.length;
    }


    function getTaxTiersRatesCount() public view returns (uint256 count) {
        return taxTiersRates.length;
    }
```

Queries taxation information

```
    function disableAutoCalculateTax() external onlyTaxOffice {
        autoCalculateTax = false;
    }


    function enableAutoCalculateTax() external onlyTaxOffice {
        autoCalculateTax = true;
    }
```

Privileged
Toggles automatic taxation calculation

```
    function distributeReward(
        address _genesisPool,
        uint256 _genesisPoolDistribution,
        address _hamsterPool,
        uint256 _hamsterPoolDistribution,
        address _airdropWallet,
        uint256 _airdropDistribution
    ) external onlyOperator {
        require(!rewardPoolDistributed, "only can distribute once");
        require(_genesisPool != address(0), "!_genesisPool");
        require(_hamsterPool != address(0), "!_hamsterPool");
        require(_airdropWallet != address(0), "!_airdropWallet");
        rewardPoolDistributed = true;
        _mint(_genesisPool, _genesisPoolDistribution);
        _mint(_hamsterPool, _hamsterPoolDistribution);
        _mint(_airdropWallet, _airdropDistribution);
    }
```

Privileged
Reward pool distribution — can only be done once

```
    function includeAddress(address _address) external
onlyOperatorOrTaxOffice returns (bool) {
        require(excludedAddresses[_address], "address can't be included");
        excludedAddresses[_address] = false;
        return true;
    }


    function excludeAddress(address _address) public
onlyOperatorOrTaxOffice returns (bool) {
        require(!excludedAddresses[_address], "address can't be excluded");
        excludedAddresses[_address] = true;
        return true;
    }
```

Privileged
Allows to include/exclude specified address from taxation

```
    function mint(address recipient_, uint256 amount_) external
onlyOperator returns (bool) {
        uint256 balanceBefore = balanceOf(recipient_);
        _mint(recipient_, amount_);
        uint256 balanceAfter = balanceOf(recipient_);
        return balanceAfter > balanceBefore;
    }
```

Privileged
Allows to mint hamster tokens to a recipient

```
    function setBurnThreshold(uint256 _burnThreshold) external
onlyTaxOffice returns (bool) {
        burnThreshold = _burnThreshold;
        return true;
    }
```

Privileged
Specifies the amount of tokens which could be burned

```
    function setHamsterOracle(address _hamsterOracle) external
onlyOperatorOrTaxOffice {
        require(_hamsterOracle != address(0), "oracle address cannot be 0
address");
        hamsterOracle = _hamsterOracle;
    }
```

Privileged
Specifies the address of the hamster oracle

```
    function setTaxCollectorAddress(address _taxCollectorAddress) external
onlyTaxOffice {
        require(_taxCollectorAddress != address(0), "tax collector address
must be non-zero address");
        taxCollectorAddress = _taxCollectorAddress;
    }
```

Privileged
Specifies the address which would recieve the tax

```
    function setTaxOffice(address _taxOffice) external
onlyOperatorOrTaxOffice {
        require(_taxOffice != address(0), "tax office address cannot be 0
address");
        emit TaxOfficeTransferred(taxOffice, _taxOffice);
        taxOffice = _taxOffice;
    }
```

Privileged
Specifies the address of the TaxOffice entity

It is recommended to perform the `emit` event after the `taxOffice` was actually transferred (best practices) — swap last two lines

---

```
    function setTaxRate(uint256 _taxRate) external onlyTaxOffice {
        require(!autoCalculateTax, "auto calculate tax cannot be enabled");
        require(_taxRate < 10000, "tax equal or bigger to 100%");
        taxRate = _taxRate;
```

```
    }


    function setTaxTiersRate(uint8 _index, uint256 _value) external
onlyTaxOffice returns (bool) {
        require(_index >= 0, "Index has to be higher than 0");
        require(_index < getTaxTiersRatesCount(), "Index has to lower than
count of tax tiers");
        taxTiersRates[_index] = _value;
        return true;
    }


    function setTaxTiersTwap(uint8 _index, uint256 _value) external
onlyTaxOffice returns (bool) {
        require(_index >= 0, "Index has to be higher than 0");
        require(_index < getTaxTiersTwapsCount(), "Index has to lower than
count of tax tiers");
        if (_index > 0) {
            require(_value > taxTiersTwaps[_index - 1]);
        }
        if (_index < getTaxTiersTwapsCount().sub(1)) {
            require(_value < taxTiersTwaps[_index + 1]);
        }
        taxTiersTwaps[_index] = _value;
        return true;
    }
```

Privileged
Allows the modification of the taxation parameters

```
    function transferFrom(
        address sender,
        address recipient,
        uint256 amount
    ) public override returns (bool) {
        uint256 currentTaxRate = 0;
        bool burnTax = false;
        if (autoCalculateTax) {
            uint256 currentHamsterPrice = _getHamsterPrice();
            currentTaxRate = _updateTaxRate(currentHamsterPrice);
            if (currentHamsterPrice < burnThreshold) {
                burnTax = true;
            }
        }
        if (currentTaxRate == 0 || excludedAddresses[sender]) {
            _transfer(sender, recipient, amount);
        } else {
```

```
            _transferWithTax(sender, recipient, amount, burnTax);
        }
        _approve(
            sender,
            _msgSender(),
            allowance(sender, _msgSender()).sub(
                amount,
                "ERC20: transfer amount exceeds allowance"
            )
        );
        return true;
    }
```

Allows to transfer tokens between accounts

No discrepancies with the original code were found

Oracle contract

View functions

- `consult`
- `twap`

Executables

- `update`

Features

```
    function consult(address _token, uint256 _amountIn) external view
returns (uint144 amountOut) {
        if (_token == token0) {
            amountOut = price0Average.mul(_amountIn).decode144();
        } else {
            require(_token == token1, "Oracle: INVALID_TOKEN");
            amountOut = price1Average.mul(_amountIn).decode144();
```

```
        }
    }
```

Queries the average price for the given token

Note that this will always return 0 before `update` has been called successfully for the first time

---

```
    function twap(address _token, uint256 _amountIn) external view returns
(uint144 _amountOut) {
        (
            uint256 price0Cumulative,
            uint256 price1Cumulative,
            uint32 blockTimestamp
        ) = UniswapV2OracleLibrary.currentCumulativePrices(address(pair));
        uint32 timeElapsed = blockTimestamp - blockTimestampLast;
        if (_token == token0) {
            _amountOut = FixedPoint.uq112x112(uint224((price0Cumulative -
price0CumulativeLast) / timeElapsed))
                .mul(_amountIn)
                .decode144();
        } else if (_token == token1) {
            _amountOut = FixedPoint.uq112x112(uint224((price1Cumulative -
price1CumulativeLast) / timeElapsed))
                .mul(_amountIn)
                .decode144();
        }
    }
```

Queries the time-weighted average price for the given token

```
    function update() external checkEpoch {
        (
            uint256 price0Cumulative,
            uint256 price1Cumulative,
            uint32 blockTimestamp
        ) = UniswapV2OracleLibrary.currentCumulativePrices(address(pair));
        uint32 timeElapsed = blockTimestamp - blockTimestampLast;
        if (timeElapsed == 0) {
            return;
        }
        price0Average = FixedPoint.uq112x112(uint224((price0Cumulative -
price0CumulativeLast) / timeElapsed));
        price1Average = FixedPoint.uq112x112(uint224((price1Cumulative -
```

```
price1CumulativeLast) / timeElapsed));
        price0CumulativeLast = price0Cumulative;
        price1CumulativeLast = price1Cumulative;
        blockTimestampLast = blockTimestamp;
        emit Updated(price0Cumulative, price1Cumulative);
    }
```

Used to update prices

Note that values for `timeElapsed`, `price0Average` and `price0Average` will overflow — unless this is desired, consider using the `unchecked` statement

---

No discrepancies with the original code were found

TaxOracle contract

View Functions

- `consult`
- `getHamsterBalance`
- `getWftmBalance`
- `getPrice`

Executables

- `setHamster`
- `setWftm`
- `setPair`

Features

```
    function consult(address _token, uint256 _amountIn) external view
returns (uint144 amountOut) {
        require(_token == address(hamster), "token needs to be hamster");
        uint256 hamsterBalance = hamster.balanceOf(pair);
        uint256 wftmBalance = wftm.balanceOf(pair);
```

```
        return uint144(hamsterBalance.mul(_amountIn).div(wftmBalance));
    }
```

Queries info about the hamster-WFTM (Wrapped Fantom) pair

```
    function getHamsterBalance() external view returns (uint256) {
            return hamster.balanceOf(pair);
    }
```

Queries the hamster balance of a pair

```
    function getWftmBalance() external view returns (uint256) {
            return wftm.balanceOf(pair);
    }
```

Queries the WFTM balance of a pair

```
    function getPrice() external view returns (uint256) {
        uint256 hamsterBalance = hamster.balanceOf(pair);
        uint256 wftmBalance = wftm.balanceOf(pair);
        return hamsterBalance.mul(1e18).div(wftmBalance);
    }
```

Calculates the price of a pair

```
    function setHamster(address _hamster) external onlyOwner returns (bool)
{
        require(_hamster != address(0), "hamster address cannot be 0");
        hamster = IERC20(_hamster);
        return true;
    }
```

Privileged
Allows to specify the hamster token address

```
function setWftm(address _wftm) external onlyOwner returns (bool) {
    require(_wftm != address(0), "wftm address cannot be 0");
    wftm = IERC20(_wftm);
    return true;
}
```

Privileged
Allows to specify the WFTM token address

```
function setPair(address _pair) external onlyOwner returns (bool) {
    require(_pair != address(0), "pair address cannot be 0");
    pair = _pair;
    return true;
}
```

Privileged
Allows to specify the token pair address

No discrepancies with the original code were found

Treasury contract

View functions

- `getBurnableHamsterLeft`
- `getHamsterUpdatedPrice`
- `getRedeemableBonds`
- `getReserve`
- `isInitialized`
- `getBondDiscountRate`
- `getBondPremiumRate`
- `getHamsterCirculatingSupply`
- `getHamsterPrice`
- `nextEpochPoint`

Executables

- `allocateSeigniorage`
- `buyBonds`
- `hamsterWheelSetLockUp`
- `initialize`
- `redeemBonds`
- `setBondDepletionFloorPercent`
- `setBootstrap`
- `setDiscountPercent`
- `setExtraFunds`
- `setHamsterPriceCeiling`
- `setMaxDebtRatioPercent`
- `setMaxDiscountRate`
- `setMaxExpansionTiersEntry`
- `setMaxPremiumRate`
- `setMaxSupplyContractionPercent`
- `setMaxSupplyExpansionPercents`
- `setMintingFactorForPayingDebt`
- `setPremiumPercent`
- `setPremiumThreshold`
- `setSupplyTiersEntry`

Features

```
    function getBurnableHamsterLeft() external view returns (uint256
_burnableHamsterLeft) {
        uint256 _hamsterPrice = getHamsterPrice();
        if (_hamsterPrice <= hamsterPriceOne) {
            uint256 _hamsterSupply = getHamsterCirculatingSupply();
            uint256 _bondMaxSupply =
_hamsterSupply.mul(maxDebtRatioPercent).div(10000);
            uint256 _bondSupply = IERC20(hamsterbond).totalSupply();
            if (_bondMaxSupply > _bondSupply) {
                uint256 _maxMintableBond = _bondMaxSupply.sub(_bondSupply);
                uint256 _maxBurnableHamster =
_maxMintableBond.mul(_hamsterPrice).div(1e18);
                _burnableHamsterLeft = Math.min(epochSupplyContractionLeft,
_maxBurnableHamster);
            }
        }
    }
```

Queries the ammount of burnable tokens left

```
    function getHamsterUpdatedPrice() external view returns (uint256
_hamsterPrice) {
        try IOracle(hamsterOracle).twap(hamster, 1e18) returns (uint144
price) {
            return uint256(price);
        } catch {
            revert("Treasury: failed to consult HAMSTER price from the
oracle");
        }
    }
```

Queries the hamster price from the Oracle

```
    function getRedeemableBonds() external view returns (uint256
_redeemableBonds) {
        uint256 _hamsterPrice = getHamsterPrice();
        if (_hamsterPrice > hamsterPriceCeiling) {
            uint256 _totalHamster =
IERC20(hamster).balanceOf(address(this));
            uint256 _rate = getBondPremiumRate();
            if (_rate > 0) {
                _redeemableBonds = _totalHamster.mul(1e18).div(_rate);
            }
        }
    }
```

Queries the amount of redeemable bonds in the application

```
    function getReserve() external view returns (uint256) {
        return seigniorageSaved;
    }
```

Queries the seigniorage amount available

```
    function isInitialized() external view returns (bool) {
        return initialized;
```

```
    }
```

Checks whether the treasury was initialized

```
    function getBondDiscountRate() public view returns (uint256 _rate) {
        uint256 _hamsterPrice = getHamsterPrice();
        if (_hamsterPrice <= hamsterPriceOne) {
            if (discountPercent == 0) {
                _rate = hamsterPriceOne;
            } else {
                uint256 _bondAmount =
hamsterPriceOne.mul(1e18).div(_hamsterPrice);
                uint256 _discountAmount =
_bondAmount.sub(hamsterPriceOne).mul(discountPercent).div(10000);
                _rate = hamsterPriceOne.add(_discountAmount);
                if (maxDiscountRate > 0 && _rate > maxDiscountRate) {
                    _rate = maxDiscountRate;
                }
            }
        }
    }


    function getBondPremiumRate() public view returns (uint256 _rate) {
        uint256 _hamsterPrice = getHamsterPrice();
        if (_hamsterPrice > hamsterPriceCeiling) {
            uint256 _hamsterPricePremiumThreshold =
hamsterPriceOne.mul(premiumThreshold).div(100);
            if (_hamsterPrice >= _hamsterPricePremiumThreshold) {
                uint256 _premiumAmount =
_hamsterPrice.sub(hamsterPriceOne).mul(premiumPercent).div(10000);
                _rate = hamsterPriceOne.add(_premiumAmount);
                if (maxPremiumRate > 0 && _rate > maxPremiumRate) {
                    _rate = maxPremiumRate;
                }
            } else {
                _rate = hamsterPriceOne;
            }
        }
    }


    function getHamsterCirculatingSupply() public view returns (uint256) {
        IERC20 hamsterErc20 = IERC20(hamster);
        uint256 totalSupply = hamsterErc20.totalSupply();
        uint256 balanceExcluded = 0;
        for (uint8 entryId = 0; entryId < excludedFromTotalSupply.length;
++entryId) {
```

```
            balanceExcluded =
balanceExcluded.add(hamsterErc20.balanceOf(excludedFromTotalSupply[entryId]
));
        }
        return totalSupply.sub(balanceExcluded);
    }


    function getHamsterPrice() public view returns (uint256 hamsterPrice) {
        try IOracle(hamsterOracle).consult(hamster, 1e18) returns (uint144
price) {
            return uint256(price);
        } catch {
            revert("Treasury: failed to consult HAMSTER price from the
oracle");
        }
    }
```

Queries budget information. about the token — bond discount rate, bond premium rate and current hamster token price

---

```
    function nextEpochPoint() public view returns (uint256) {
        return startTime.add(epoch.mul(PERIOD));
    }
```

Queries the time of the next epoch

```
    function allocateSeigniorage() external onlyOneBlock checkCondition
checkEpoch checkOperator {
        _updateHamsterPrice();
        previousEpochHamsterPrice = getHamsterPrice();
        uint256 hamsterSupply =
getHamsterCirculatingSupply().sub(seigniorageSaved);
        if (epoch < bootstrapEpochs) {

_sendToHamsterWheel(hamsterSupply.mul(bootstrapSupplyExpansionPercent).div(
10000));
        } else {
            if (previousEpochHamsterPrice > hamsterPriceCeiling) {
                uint256 bondSupply = IERC20(hamsterbond).totalSupply();
                uint256 _percentage =
previousEpochHamsterPrice.sub(hamsterPriceOne);
                uint256 _savedForBond;
                uint256 _savedForHamsterWheel;
```

```
                uint256 _mse =
_calculateMaxSupplyExpansionPercent(hamsterSupply).mul(1e14);
                if (_percentage > _mse) {
                    _percentage = _mse;
                }
                if (seigniorageSaved >=
bondSupply.mul(bondDepletionFloorPercent).div(10000)) {
                    _savedForHamsterWheel =
hamsterSupply.mul(_percentage).div(1e18);
                } else {
                    uint256 _seigniorage =
hamsterSupply.mul(_percentage).div(1e18);
                    _savedForHamsterWheel =
_seigniorage.mul(seigniorageExpansionFloorPercent).div(10000);
                    _savedForBond =
_seigniorage.sub(_savedForHamsterWheel);
                    if (mintingFactorForPayingDebt > 0) {
                        _savedForBond =
_savedForBond.mul(mintingFactorForPayingDebt).div(10000);
                    }
                }
                if (_savedForHamsterWheel > 0) {
                    _sendToHamsterWheel(_savedForHamsterWheel);
                }
                if (_savedForBond > 0) {
                    seigniorageSaved = seigniorageSaved.add(_savedForBond);
                    IBasisAsset(hamster).mint(address(this),
_savedForBond);
                    emit TreasuryFunded(now, _savedForBond, epoch);
                }
            }
        }
    }
```

Seigniorage logic implementation

```
    function buyBonds(
        uint256 _hamsterAmount,
        uint256 targetPrice
    ) external onlyOneBlock checkCondition checkOperator {
        require(_hamsterAmount > 0, "Treasury: cannot purchase bonds with
zero amount");
        uint256 hamsterPrice = getHamsterPrice();
        require(hamsterPrice == targetPrice, "Treasury: HAMSTER price
moved");
        require(
            hamsterPrice < hamsterPriceOne,
            "Treasury: hamsterPrice not eligible for bond purchase"
```

```
        );
        require(_hamsterAmount <= epochSupplyContractionLeft, "Treasury:
not enough bond left to purchase");
        uint256 _rate = getBondDiscountRate();
        require(_rate > 0, "Treasury: invalid bond rate");
        uint256 _bondAmount = _hamsterAmount.mul(_rate).div(1e18);
        uint256 hamsterSupply = getHamsterCirculatingSupply();
        uint256 newBondSupply =
IERC20(hamsterbond).totalSupply().add(_bondAmount);
        require(newBondSupply <=
hamsterSupply.mul(maxDebtRatioPercent).div(10000), "over max debt ratio");
        IBasisAsset(hamster).burnFrom(msg.sender, _hamsterAmount);
        IBasisAsset(hamsterbond).mint(msg.sender, _bondAmount);
        epochSupplyContractionLeft =
epochSupplyContractionLeft.sub(_hamsterAmount);
        _updateHamsterPrice();
        emit BoughtBonds(msg.sender, _hamsterAmount, _bondAmount, epoch);
    }
```

Allows to buy the bonds

```
    function hamsterWheelSetLockUp(
        uint256 _withdrawLockupEpochs,
        uint256 _rewardLockupEpochs
    ) external onlyOperator {
        IHamsterWheel(hamsterWheel).setLockUp(_withdrawLockupEpochs,
_rewardLockupEpochs);
    }
```

Privileged
Governs the hamster wheel lockup parameters

```
function initialize(
        address _hamster,
        address _hamsterbond,
        address _hamstershare,
        address _hamsterOracle,
        address _hamsterWheel,
        uint256 _startTime,
        address[] memory excludedFromTotalSupply_
    ) external notInitialized {
        hamster = _hamster;
        hamsterbond = _hamsterbond;
        hamstershare = _hamstershare;
```

```
        hamsterOracle = _hamsterOracle;
        hamsterWheel = _hamsterWheel;
        startTime = _startTime;
        hamsterPriceOne = 10**18;
        hamsterPriceCeiling = hamsterPriceOne.mul(101).div(100);
        supplyTiers = [
            0 ether,
            500000 ether,
            1000000 ether,
            1500000 ether,
            2000000 ether,
            5000000 ether,
            10000000 ether,
            20000000 ether,
            50000000 ether
        ];
        maxExpansionTiers = [
            450,
            400,
            350,
            300,
            250,
            200,
            150,
            125,
            100
        ];
        maxSupplyExpansionPercent = 400;
        bondDepletionFloorPercent = 10000;
        seigniorageExpansionFloorPercent = 3500;
        maxSupplyContractionPercent = 300;
        maxDebtRatioPercent = 3500;
        premiumThreshold = 110;
        premiumPercent = 7000;
        bootstrapEpochs = 28;
        bootstrapSupplyExpansionPercent = 450;
        seigniorageSaved = IERC20(hamster).balanceOf(address(this));
        initialized = true;
        operator = msg.sender;
        for (uint256 i = 0; i < excludedFromTotalSupply_.length; i++) {
            excludedFromTotalSupply.push(excludedFromTotalSupply_[i]);
            // HamsterGenesisPool && HamsterRewardPool
        }
        emit Initialized(msg.sender, block.number);
    }
```

Initialization of the treasury parameters

```
    function redeemBonds(
        uint256 _bondAmount,
        uint256 targetPrice
    ) external onlyOneBlock checkCondition checkOperator {
        require(_bondAmount > 0, "Treasury: cannot redeem bonds with zero
amount");
        uint256 hamsterPrice = getHamsterPrice();
        require(hamsterPrice == targetPrice, "Treasury: HAMSTER price
moved");
        require(
            hamsterPrice > hamsterPriceCeiling,
            "Treasury: hamsterPrice not eligible for bond purchase"
        );
        uint256 _rate = getBondPremiumRate();
        require(_rate > 0, "Treasury: invalid bond rate");
        uint256 _hamsterAmount = _bondAmount.mul(_rate).div(1e18);
        require(IERC20(hamster).balanceOf(address(this)) >= _hamsterAmount,
"Treasury: treasury has no more budget");
        seigniorageSaved = seigniorageSaved.sub(Math.min(seigniorageSaved,
_hamsterAmount));
        IBasisAsset(hamsterbond).burnFrom(msg.sender, _bondAmount);
        IERC20(hamster).safeTransfer(msg.sender, _hamsterAmount);
        _updateHamsterPrice();
        emit RedeemedBonds(msg.sender, _hamsterAmount, _bondAmount, epoch);
    }
```

Allows to redeem the bonds for hamster tokens

```
    function setBondDepletionFloorPercent(uint256
_bondDepletionFloorPercent) external onlyOperator {
        require(
            _bondDepletionFloorPercent >= 500 && _bondDepletionFloorPercent
<= 10000,
            "out of range"
        );
        bondDepletionFloorPercent = _bondDepletionFloorPercent;
    }


    function setBootstrap(uint256 _bootstrapEpochs, uint256
_bootstrapSupplyExpansionPercent) external onlyOperator {
        require(_bootstrapEpochs <= 120, "_bootstrapEpochs: out of range");
        require(
            _bootstrapSupplyExpansionPercent >= 100 &&
_bootstrapSupplyExpansionPercent <= 1000,
            "_bootstrapSupplyExpansionPercent: out of range"
        );
```

```
        bootstrapEpochs = _bootstrapEpochs;
        bootstrapSupplyExpansionPercent = _bootstrapSupplyExpansionPercent;
    }


    function setDiscountPercent(uint256 _discountPercent) external
onlyOperator {
        require(_discountPercent <= 20000, "_discountPercent is over
200%");
        discountPercent = _discountPercent;
    }


    function setExtraFunds(
        address _daoFund,
        uint256 _daoFundSharedPercent,
        address _devFund,
        uint256 _devFundSharedPercent
    ) external onlyOperator {
        require(_daoFund != address(0), "zero");
        require(_daoFundSharedPercent <= 3000, "out of range");
        require(_devFund != address(0), "zero");
        require(_devFundSharedPercent <= 1000, "out of range");
        daoFund = _daoFund;
        daoFundSharedPercent = _daoFundSharedPercent;
        devFund = _devFund;
        devFundSharedPercent = _devFundSharedPercent;
    }


    function setHamsterPriceCeiling(uint256 _hamsterPriceCeiling) external
onlyOperator {
        require(
            _hamsterPriceCeiling >= hamsterPriceOne && _hamsterPriceCeiling
<= hamsterPriceOne.mul(120).div(100),
            "out of range"
        );
        hamsterPriceCeiling = _hamsterPriceCeiling;
    }


    function setMaxDebtRatioPercent(uint256 _maxDebtRatioPercent) external
onlyOperator {
        require(_maxDebtRatioPercent >= 1000 && _maxDebtRatioPercent <=
10000, "out of range");
        maxDebtRatioPercent = _maxDebtRatioPercent;
    }


    function setMaxDiscountRate(uint256 _maxDiscountRate) external
onlyOperator {
        maxDiscountRate = _maxDiscountRate;
```

```
    }


    function setMaxExpansionTiersEntry(uint8 _index, uint256 _value)
external onlyOperator returns (bool) {
        require(_index >= 0, "Index has to be higher than 0");
        require(_index < 9, "Index has to be lower than count of tiers");
        require(_value >= 10 && _value <= 1000, "_value: out of range");
        maxExpansionTiers[_index] = _value;
        return true;
    }


    function setMaxPremiumRate(uint256 _maxPremiumRate) external
onlyOperator {
        maxPremiumRate = _maxPremiumRate;
    }


    function setMaxSupplyContractionPercent(uint256
_maxSupplyContractionPercent) external onlyOperator {
        require(
            _maxSupplyContractionPercent >= 100 &&
_maxSupplyContractionPercent <= 1500,
            "out of range"
        );
        maxSupplyContractionPercent = _maxSupplyContractionPercent;
    }


    function setMaxSupplyExpansionPercents(uint256
_maxSupplyExpansionPercent) external onlyOperator {
        require(
            _maxSupplyExpansionPercent >= 10 && _maxSupplyExpansionPercent
<= 1000,
            "_maxSupplyExpansionPercent: out of range"
        );
        maxSupplyExpansionPercent = _maxSupplyExpansionPercent;
    }


    function setMintingFactorForPayingDebt(uint256
_mintingFactorForPayingDebt) external onlyOperator {
        require(
            _mintingFactorForPayingDebt >= 10000 &&
_mintingFactorForPayingDebt <= 20000,
            "_mintingFactorForPayingDebt: out of range"
        );
        mintingFactorForPayingDebt = _mintingFactorForPayingDebt;
    }
```

```
    function setPremiumPercent(uint256 _premiumPercent) external
onlyOperator {
        require(_premiumPercent <= 20000, "_premiumPercent is over 200%");
        premiumPercent = _premiumPercent;
    }

    function setPremiumThreshold(uint256 _premiumThreshold) external
onlyOperator {
        require(_premiumThreshold >= hamsterPriceCeiling,
"_premiumThreshold exceeds hamsterPriceCeiling");
        require(_premiumThreshold <= 150, "_premiumThreshold is higher than
1.5");
        premiumThreshold = _premiumThreshold;
    }


    function setSupplyTiersEntry(uint8 _index, uint256 _value) external
onlyOperator returns (bool) {
        require(_index >= 0, "Index has to be higher than 0");
        require(_index < 9, "Index has to be lower than count of tiers");
        if (_index > 0) {
            require(_value > supplyTiers[_index - 1]);
        }
        if (_index < 8) {
            require(_value < supplyTiers[_index + 1]);
        }
        supplyTiers[_index] = _value;
        return true;
    }
```

Functions above allow the government and modification of various budgeting parameters, such as premiums, supply maintanence, debt management, etc.

No discrepancies with the original code were found

Note that the contract code size exceeds 24576 bytes (a limit introduced in Spurious Dragon) and therefore the deployment on mainnet will likely fail — consider refactoring the code and using libraries and inheritance to reduce the code size

---

Application security checklist

Test	Result
Compiler errors	Passed
Possible delays in data delivery	Passed

Test	Result
Timestamp dependence	Passed
Integer Overflow and Underflow	Passed
Race Conditions and Reentrancy	Passed
DoS with Revert	Passed
DoS with block gas limit	Passed
Methods execution permissions	Passed
Economy model of the contract	Passed
Private user data leaks	Passed
Malicious Events Log	Passed
Scoping and Declarations	Passed
Uninitialized storage pointers	Passed
Arithmetic accuracy	Passed
Design Logic	Passed
Impact of the exchange rate	Passed
Oracle Calls	Passed
Cross-function race conditions	Passed
Safe OpenZeppelin contracts and implementation usage	Passed [1]
Whitepaper-Website-Contract correlation	Not Checked
Front Running	Not Checked

[1] Contracts rely on the outdated version of OpenZeppelin contracts, even though it is not a direct risk or vulnerability thread, it is recommended to update the codebase to match current Solidity version and use the latest library contracts.

In case the update isn't feasible, make sure contracts point to the correct OpenZeppelin library to avoid compilation and deployment issues.

Project stats and evaluation

Contract Programming

Test	Result
Solidity version not specified	Passed
Solidity version too old	Failed [1]
Integer overflow/underflow	Passed
Function input parameters lack of check	Passed
Function input parameters check bypass	Passed
Function access control lacks management	Passed
Critical operation lacks event log	Passed
Human/contract checks bypass	Passed
Random number generation/use vulnerability	Passed
Fallback function misuse	Passed
Race condition	Passed
Logical vulnerability	Passed
Other programming issues	Passed

[1] Even though this doesn't explicitly pose a threat, the solidity version used in the codebase is 0.6.12, while the current version is 0.8.11 — if possible, it is recommended to update the codebase prior to deployment

Code Specification

Test	Result
Visibility not explicitly declared	Passed
Variable storage location not explicitly declared	Passed
Use keywords/functions to be deprecated	Passed
Other code specification issues	Passed

Gas Optimization

Test	Result
Assert () misuse	Passed
High consumption ‘for/while’ loop	Passed

Test	Result
High consumption ‘storage’ storage	Passed
“Out of Gas” Attack	Passed

Business Risk

Test	Result
“Short Address” Attack	Passed
“Double Spend” Attack	Passed

Overall Audit Result: Passed

Executive Summary

According to the standard audit assessment, customer's solidity smart contracts (Hamster project) are well-secured, however, it is recommended to perform an extensive audit assessment to bring a more assured conclusion.

We used various tools like Slither and Remix IDE. At the same time this findings are based on critical analysis conducted during static code review.

All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the Quick Stat section.

Issues	Number
Critical	0
High	0
Medium	0
Low	1

Code Quality

Hamster Finance project codebase consists of multiple smart contracts, forket from the Tomb Finance project. It has various inherited contracts such as IERC20 and

Ownable, as well as the utilization of well-known libraries, such as OpenZeppelin and SafeMath. Overall, the code is well-written and readable, however, the use of comments is encouraged. **No discrepancies, which could impact the project in any meaningful way, from the original code (TombFinance) were found**

Documentation

As mentioned above, it's recommended to write comments in the smart contract code, so that anyone can quickly understand the programming flow as well as complex code logic. The code was audited from the HamsterFinance GitHub repository, provided by the client.

Audit Findings

Critical

No critical severity vulnerabilities were found

High

No critical severity vulnerabilities were found

Medium

No critical severity vulnerabilities were found

Low

1

The size of the Treasury contract code exceeds the Spurious Dragon byte limit, which may cause the deployment on the mainnet to fail — it is suggested to refactor the code to reduce it's size

Conclusion

The codebase of the Hamster Project has overall passed the audit successfuly and can be considered as a "Well-Secured" application. No discrepancies from the original project's codebase (TombFinance) were found. There is, however, one low-level issue found, which could prevent the project from being successfuly deployed. Even though it cannot be considered as a vulnerability, it is strongly suggested to address this issue. Otherwise, the code can be considered to be production-ready.

Since possible test cases can be unlimited for such extensive project, we can provide no guarantees on the future outcomes and project operation. We have used all the latest static tools and manual analysis to cover the greatest possible amount of test cases. Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. Smart contracts high-level description of functionality and security was presented in the Application security checklist section of the report.

Audit report contains all found security vulnerabilities and other issues found in the reviewed code.

Security status of the reviewed codebase is "Well-Secured"

Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goal of our security audits is to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review

In manually reviewing the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis

Our audit techniques included manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site to get a high-level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code

dependencies, skim open issue tickets, and generally investigate details other than the implementation.

Documenting Results

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally, follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyze the feasibility of an attack in a live system to make a final decision.

Suggested Solutions

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.