PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For Polycat Paw

07 August 2021

paladinsec.co

info@paladinsec.co

Table of Contents

Disclaimer

Paladin Blockchain Security ("Paladin") has conducted an independent audit to verify the integrity of and highlight any vulnerabilities or errors, intentional or unintentional, that may be present in the codes that were provided for the scope of this audit. This audit report does not constitute agreement, acceptance or advocation for the Project that was audited, and users relying on this audit report should not consider this as having any merit for financial advice in any shape, form or nature. The contracts audited do not account for any economic developments that may be pursued by the Project in question, and that the veracity of the findings thus presented in this report relate solely to the proficiency, competence, aptitude and discretion of our independent auditors, who make no guarantees nor assurance that the contracts are completely free of exploits, bugs, vulnerabilities or deprecation of technologies. Further, this audit report shall not be disclosed nor transmitted to any persons or parties on any objective, goal or justification without due written assent, acquiescence or approval by Paladin.

All information provided in this report does not constitute financial or investment advice, nor should it be used to signal that any persons reading this report should invest their funds without sufficient individual due diligence regardless of the findings presented in this report. Information is provided 'as is', and Paladin is under no covenant to the completeness, accuracy or solidity of the contracts audited. In no event will Paladin or its partners, employees, agents or parties related to the provision of this audit report be liable to any parties for, or lack thereof, decisions and/or actions with regards to the information provided in this audit report.

Cryptocurrencies and any technologies by extension directly or indirectly related to cryptocurrencies are highly volatile and speculative by nature. All reasonable due diligence and safeguards may yet be insufficient, and users should exercise considerable caution when participating in any shape or form in this nascent industry.

The audit report has made all reasonable attempts to provide clear and articulate recommendations to the Project team with respect to the rectification, amendment and/or revision of any highlighted issues, vulnerabilities or exploits within the contracts provided. It is the sole responsibility of the Project team to sufficiently test and perform checks, ensuring that the contracts are functioning as intended, specifically that the functions therein contained within said contracts have the desired intended effects, functionalities and outcomes of the Project team.

1 Overview

This report has been prepared for Polycat Finance's Paw token and Masterchef. Paladin provides a user-centred examination of the smart contracts to look for vulnerabilities, logic errors or other issues from both an internal and external perspective.

At the time of assessment, the contracts were sent to Paladin via a GitHub repository and may differ from the ones deployed on the blockchain.

1.1 Summary

Project Name	Polycat Finance – Paw
URL	https://polycat.finance/
Platform	Polygon
Language	Solidity

1.2 Contracts Assessed

The final contracts were sent to us in a 7z archive file. After the project has deployed their contracts on the blockchain, we will verify if deployed contracts match the audited files and updated this report.

Name	Contract	Live Code Match
MasterChef	MasterChef.sol	
PawToken	PawToken.sol	
RewardLocker	RewardLocker.sol	
Archive	http://paladinsec.co/assets/audits/source/polycat-paw-round-3.7z SHA256 Checksum: 4f2b83453d2ba7ef420c51234dcd8fbbeeec9c90eae240c82a2cc96a8b29967f	

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	2	2	-	-
Medium	0	-	-	-
Low	5	4	-	1
Informational	13	9	-	4
Total	**20**	**15**	**0**	**5**

Classification of Issues

Severity	Description
High	Exploits, vulnerabilities or errors that will certainly or probabilistically lead towards loss of funds, control, or impairment of the contract and its functions. Issues under this classification are recommended to be fixed with utmost urgency.
Medium	Bugs or issues with that may be subject to exploit, though their impact is somewhat limited. Issues under this classification are recommended to be fixed as soon as possible.
Low	Effects are minimal in isolation and do not pose a significant danger to the project or its users. Issues under this classification are recommended to be fixed nonetheless.
Informational	Consistency, syntax or style best practices. Generally pose a negligible level of risk, if any.

1.3.1 MasterChef

ID	Severity	Summary	Status
01	HIGH	(Flash loan) Exploit can be carried out to drain Masterchef of all native tokens due to reward miscalculation when depositing to another wallet	RESOLVED
02	HIGH	Deposit takes the token amount twice from the user	RESOLVED
03	LOW	Allowing deposits to be sent to another wallet may make it difficult for users to verify that they are not being mislead into making a malicious transaction	RESOLVED
04	LOW	Withdraw does not transfer funds to the `to` address	RESOLVED
05	LOW	`massUpdatePools` should be called before a pool is actually added	RESOLVED
06	LOW	`transferPawOwnership` can be used by the `pawTransferOwner` to reclaim ownership of the token and mint and dump a large amount of tokens	ACKNOWLEDGED
07	LOW	`rewardLocker` does not have allowance to withdraw Paw tokens from the Masterchef by default	RESOLVED
08	INFORMATIONAL	UI DoS possible by creating thousands of vesting schedules for any user	RESOLVED
09	INFORMATIONAL	`msg.sender` is already an address and does not have to be cast to address again	RESOLVED
10	INFORMATIONAL	Pools use the contract balance to figure out the total deposits	RESOLVED
11	INFORMATIONAL	Lack of events for `add`, `set`, `harvest`, `resetLockerAllowance`, `setPawTransferOwnerAddress` and `transferPawOwnership`	RESOLVED

1.3.2 PawToken

ID	Severity	Summary	Status
12	INFORMATIONAL	`mint` function can be used to pre-mint large amounts of tokens before ownership is transferred to the Masterchef	ACKNOWLEDGED
13	INFORMATIONAL	Governance functionality is broken	ACKNOWLEDGED
14	INFORMATIONAL	`delegateBySig` can be frontrun and cause denial of service	ACKNOWLEDGED
15	INFORMATIONAL	Errors still reference the Goose EGG token	RESOLVED

1.3.3 RewardLocker

ID	Severity	Summary	Status
16	INFORMATIONAL	Setting the lock duration to a severely excessive value could revert locks	RESOLVED
17	INFORMATIONAL	The frontend might become slow if a user has many vesting schedules	ACKNOWLEDGED
18	INFORMATIONAL	Certain functions can revert due to the block gas limit	RESOLVED
19	INFORMATIONAL	Locking mechanism will misaccount with transfer tax tokens	RESOLVED
20	INFORMATIONAL	Lack of event for `updateMaxContractSize`	RESOLVED

2 Findings

2.1 MasterChef

The Masterchef is a fork of Goose Finance's Masterchef with a few customisations. A notable feature of forking this Masterchef is the removal of the migrator function from Pancakeswap, which of late has been used maliciously to steal users' tokens. Another notable feature is that is that they removed the `depositFee`. Finally, there's a unique feature where all rewards are actually sent to a vesting contract, where they will become withdrawable linearly over a duration set by Polycat.

We commend Polycat Finance on their decision to fork a relatively safer version of the Masterchef.

2.1.1 Privileged Roles

The following functions can be called by the owner of the Masterchef:

- `add`
- `set`
- `setDevAddress`
- `updateEmissionRate`
- `resetLockerAllowance`

The PawTransferOwner has the following privileges (this will be behind a longer timelock):

- `setPawTransferOwnerAddress`
- `transferPawOwnership`

2.1.2 Issues & Recommendations

Issue #01	(Flash loan) Exploit can be carried out to drain Masterchef of all native tokens due to reward miscalculation when depositing to another wallet
Severity	HIGH SEVERITY

Description

The deposit function has been modified to allow users to deposit to another wallet, this could for example be used to gift friends.

Line 169

```
function deposit(uint256 _pid, uint256 _amount, bool _shouldHarvest, address _to)
external nonReentrant {
```

Whenever a user receives a reward, a record is kept of their `lastPawPerShare`, which indicates how much rewards they have received per token staked. As time passes, the current accumulated `pawPerShare` increases and when a harvest is finally done, the difference between the two values is multiplied by the current stake of the user and is given as a reward to the user's locked balance.

Lines 207–222

```
// Update the rewards of caller, and harvests if needed
function _updateUserReward(uint256 _pid, address _to, bool _shouldHarvest)
internal {
    PoolInfo storage pool = poolInfo[_pid];
    UserInfo storage user = userInfo[_pid][msg.sender];
    updatePool(_pid);
    if (user.amount == 0) {
        user.lastPawPerShare = pool.accPawPerShare;
    }
    uint256 pending =
user.amount.mul(pool.accPawPerShare.sub(user.lastPawPerShare)).div(1e18).add(user
.unclaimed);
    user.unclaimed = _shouldHarvest ? 0 : pending;
    if (_shouldHarvest && pending > 0) {
         _lockReward(_to, pending);
    }
    user.lastPawPerShare = pool.accPawPerShare;
}
```

This behavior works since before every deposit and withdrawal, the reward is calculated and `lastPawPerShare` is updated to the current Paw per share using the `_updateUserReward` function mentioned above. In case this update before a deposit is not done, one could stake 1 token and after a long time deposit more tokens. When a harvest is called after the second deposit, the `lastPawPerShare` is still a very old value and the harvest is then done as if the user was holding the tokens since the first deposit. In this case, the user would receive significantly more rewards than appropriate.

Due to the `deposit` implementation, only the depositor's `lastPawPerShare` is incremented, not the `to_` recipient. This is because `_updateUserReward` uses `msg.sender` as the user account to update. Thus, the exact issue mentioned in the previous paragraph presents itself when you deposit with a secondary wallet. Using a flash loan, this issue can be exploited to cheaply drain and claim all tokens in the Masterchef.

Exploit scenario

1. Create two contracts `from` and `to`.
2. Deposit 1 native token with `to`.
3. Wait a long time.
4. Flash loan a high number of native tokens from the exchange pair.
5. Deposit all native tokens using the `from` account, but setting the `_to` variable to the `to` contract.
6. Harvest with `to`.
7. Withdraw tokens and send them to `from` for loan repayment.

The cost of this flashloan is between 0.2% and 0.3% of the value of loaned tokens (depending on the exchange used and its fees). This cost can be reduced to near zero as the exploiter could create for example 20 `to` contracts, and forward the tokens through all of them before repaying the loan.

Due to the negligible cost it is certain that this vulnerability will be exploited, resulting in the loss of all staked native tokens which will likely be sold off by the exploiter(s).

Recommendation Consider removing the `_to` behavior altogether. Although it might sound desirable to be able to send rewards to someone else, it makes the contract more complex as seen in this issue and may cause undesirable, unintended effects later on.

If this is impossible, adjusting the `_updateUserReward` with a from address parameter as follows could be considered:

Lines 208–220

```
function _updateUserReward(uint256 _pid, address _from, address _to, bool
_shouldHarvest) internal {
    PoolInfo storage pool = poolInfo[_pid];
    UserInfo storage user = userInfo[_pid][_from];
    updatePool(_pid);
    if (user.amount == 0) {
        user.lastPawPerShare = pool.accPawPerShare;
    }
    uint256 pending =
user.amount.mul(pool.accPawPerShare.sub(user.lastPawPerShare)).div(1e18).add(user
.unclaimed);
    user.unclaimed = _shouldHarvest ? 0 : pending;
    if (_shouldHarvest && pending > 0) {
        _lockReward(_to, pending);
    }
    user.lastPawPerShare = pool.accPawPerShare;
}
```

This should then be called before any user's `user.amount` adjustment. Note that this last solution is undesirable given a later issue.

Resolution RESOLVED

The client had initial plans to use the functionality for interesting features where other contracts could send their rewards to a user. However, after a few back-and-forth implementations which suffered from other issues, the client decided to opt for the safety of their native token by leaving this novel feature out.

The client and us believe that a lot of the interesting features that they were hoping to do are still possible to do with some clever contracts, even with this risk-bearing functionality removed.

Issue #02	Deposit takes the token amount twice from the user
Severity	HIGH SEVERITY

Location	Lines 178-183
	`uint256 beforeDeposit = pool.lpToken.balanceOf(address(this));`
	`pool.lpToken.safeTransferFrom(address(msg.sender), address(this), _amount);`
	`uint256 afterDeposit = pool.lpToken.balanceOf(address(this));`
	`_amount = afterDeposit.sub(beforeDeposit);`
	
	`pool.lpToken.safeTransferFrom(address(msg.sender), address(this), _amount);`

Description	A typo has been made in the PolyCat code that transfers in the tokens `lpTokens` twice from the user to the Masterchef. This will effectively cause the user to spend twice the amount of funds for a single deposit.
Recommendation	Consider removing the second `safeTransferFrom` occurrence.
Resolution	RESOLVED The client has explained that they had read through a few Paladin reports to align with our user-oriented requirements. Before the audit, they quickly added in some preliminary resolutions to reduce our workload and during this process they accidentally left in this double `transferFrom`. This code is now removed.

Issue #03	**Allowing deposits to be sent to another wallet may make it difficult for users to verify that they are not being mislead into making a malicious transaction**
Severity	LOW SEVERITY
Location	Line 169 `function deposit(uint256 _pid, uint256 _amount, bool _shouldHarvest, address _to) external nonReentrant {`
Description	As the PolyCat Paw Masterchef extends the `deposit` function with a `to` parameter, it allows deposited funds to be sent to a different wallet. A large part of the DeFi community has been trained to validate the contract they are interacting with using a database like the one of RugDoc's, but they might not be able to validate this `_to` parameter easily. This has a similar effect to the issue of allowing deposit fees up to 100%: if the frontend is ever compromised, user funds could be lost even if they check the contract they interact with. In our experience, third-party reviewers like RugDoc prefer this risk to not be present.
Recommendation	Consider removing the `_to` parameter and behavior if it is not necessary for any functionality. This also simplifies the previous severe issue.
Resolution	RESOLVED The ability to transfer funds and rewards to another address has been removed completely, simplifying the code.

Issue #04	Withdraw does not transfer funds to the `to` address
Severity	LOW SEVERITY

Location	Lines 181–193
	`// Withdraw LP tokens from MasterChef.`
	`function withdraw(uint256 _pid, uint256 _amount, bool _shouldHarvest, address _to) external nonReentrant {`
	`  PoolInfo storage pool = poolInfo[_pid];`
	`  UserInfo storage user = userInfo[_pid][msg.sender];`
	`  require(user.amount >= _amount, "withdraw: not good");`
	`  updatePool(_pid);`
	`  _updateUserReward(_pid, _to, _shouldHarvest);`
	`  if (_amount > 0) {`
	`  user.amount = user.amount.sub(_amount);`
	`  pool.lpToken.safeTransfer(address(msg.sender), _amount);`
	`  }`
	`  emit Withdraw(msg.sender, _pid, _amount, _shouldHarvest, _to);`
	`}`

Description	The withdraw function still transfers the tokens to the transaction sender, and not the `_to` address. Since this conflicts with the deposit behavior where the balance of the `_to` address is incremented, we believe this might not be desirable.
Recommendation	Consider removing the to parameter, as indicated in previous issues. If this parameter is important, consider whether this behavior is desired. If the `_to` parameter is considered sufficiently desirable but this behavior is wrong, consider changing the transfer to `pool.lpToken.safeTransfer(_to, _amount);`
Resolution	RESOLVED The ability to transfer funds to another address has been removed completely.

Issue #05	`massUpdatePools` should be called before a pool is actually added
Severity	LOW SEVERITY
Location	Lines 101–114

```
function add(uint256 _allocPoint, IERC20 _lpToken, bool
_massUpdatePools) external onlyOwner nonDuplicated(_lpToken) {
    uint256 lastRewardBlock = block.number > startBlock ?
block.number : startBlock;
    totalAllocPoint = totalAllocPoint.add(_allocPoint);
    poolExistence[_lpToken] = true;
    poolInfo.push(PoolInfo({
        lpToken: _lpToken,
        allocPoint: _allocPoint,
        lastRewardBlock: lastRewardBlock,
        accPawPerShare: 0
    }));
    if (_massUpdatePools) {
        massUpdatePools(); // This ensures that
massUpdatePools will not exceed gas limit
    }
}
```

Description

The reason why `massUpdatePools` is often called in the `add` function is because the addition of a new pool changes the reward distribution. By calling `massUpdatePools` before adjusting this distribution, the new distribution is not used for previous periods. If no update has been done in a long time, the other pools would unfairly receive slightly less rewards over the period between the addition and the harvest.

Recommendation

Consider moving the `massUpdatePools` section to the top of the `add` function:

Lines 101–114

```
function add(uint256 _allocPoint, IERC20 _lpToken, bool _massUpdatePools)
external onlyOwner nonDuplicated(_lpToken) {
    if (_massUpdatePools) {
        massUpdatePools(); // This ensures that massUpdatePools will not exceed
gas limit
    }
    uint256 lastRewardBlock = block.number > startBlock ? block.number :
startBlock;
    totalAllocPoint = totalAllocPoint.add(_allocPoint);
    poolExistence[_lpToken] = true;
    poolInfo.push(PoolInfo({
        lpToken: _lpToken,
        allocPoint: _allocPoint,
        lastRewardBlock: lastRewardBlock,
        accPawPerShare: 0
    }));
}
```

Resolution RESOLVED

Resolved using the recommended code.

Issue #06	**`transferPawOwnership` can be used by the `pawTransferOwner` to reclaim ownership of the token and mint and dump a large amount of tokens**
Severity	LOW SEVERITY

Location	Lines 310–313
	`function transferPawOwnership(address _newOwner) external {`
	`  require(msg.sender == pawTransferOwner);`
	`  paw.transferOwnership(_newOwner);`
	`}`

Description	Only the owner of the token is capable of minting and since the Masterchef will be the owner of said token, no one else can. However, this version of the Masterchef has a function `transferPawOwnership` to reclaim ownership of the Masterchef. After ownership of the Masterchef has been reclaimed, the new owner can potentially mint tokens and dump them. This issue is marked as low severity since the project has already added this function to a separate owner address and has indicated in the comments that it will be put behind a sufficiently long timelock.
Recommendation	This issue will be marked as resolved as soon as the ownership transfer to the timelock is confirmed.
Resolution	ACKNOWLEDGED Acknowledged pending confirmation.

Issue #07	`rewardLocker` does not have allowance to withdraw Paw tokens from the Masterchef by default
Severity	LOW SEVERITY
Description	By default, the `rewardLocker` cannot withdraw Paw tokens from the Masterchef as allowance has to be explicitly given through the `resetLockerAllowance` function. If this step is forgotten after transferring ownership to a timelock, this might cause unexpected issues.

Recommendation(s)	Consider making `resetLockerAllowance` `public` and calling it in the constructor.
	
	Note that this will require Paw to be no longer marked as `immutable`; an alternative can thus be to do an explicit approval in the constructor using the `_paw` variable. This latter option allows the Paw token to remain marked as `immutable`.
	Line 291-297
	`constructor(`
	`  PawToken _paw,`
	`  uint256 _startBlock,`
	`  IRewardLocker _rewardLocker,`
	`  address _devAddress,`
	`  address _pawTransferOwner`
	`) public {`
	`  paw = _paw;`
	`  startBlock = _startBlock;`
	`  rewardLocker = _rewardLocker;`
	`  devAddress = _devAddress;`
	`  pawTransferOwner = _pawTransferOwner;`
	`  IERC20(_paw).safeApprove(address(rewardLocker), uint256(0));`
	`  IERC20(_paw).safeIncreaseAllowance(address(rewardLocker), uint256(-1));`
	`}`

Resolution	RESOLVED The recommendation has been implemented.

Issue #08	UI DoS possible by creating thousands of vesting schedules for any user
Severity	INFORMATIONAL

Location	Lines 172–176
	`function deposit(uint256 _pid, uint256 _amount, bool _shouldHarvest, address _to) external nonReentrant {`
	`    PoolInfo storage pool = poolInfo[_pid];`
	`    UserInfo storage user = userInfo[_pid][_to];`
	`    updatePool(_pid);`
	`    _updateUserReward(_pid, _to, _shouldHarvest);`
	`}`

Description	Since a deposit can be made to anyone and they will receive a vesting entry for their rewards, a cheap attack to annoy a disliked user could be to repeatedly send tiny deposits to them, creating a new vesting schedule on every deposit. This would allow you to at a very low cost add thousands of vesting schedules to their UI and potentially deny them from finding their actual withdrawals.
Recommendation	Consider whether the transfer to functionality is really required and if it, consider whether this behavior is problematic.
Resolution	RESOLVED The `to` parameter has been removed from deposits and withdrawals.

Issue #09	`msg.sender` is already an address and does not have to be cast to address again
Severity	INFORMATIONAL
Description	Throughout the codebase, `msg.sender` is explicitly cast to an address like in the following code: `address(msg.sender)`. Since `msg.sender` is already an address, this code is redundant (although it does no harm either).
Recommendation	Consider replacing all occurrences of `address(msg.sender)` with `msg.sender`.
Resolution	RESOLVED

Issue #10	Pools use the contract balance to figure out the total deposits
Severity	INFORMATIONAL
Description	As with pretty much all Masterchefs, the total number of tokens in the Masterchef contract is used to determine the total number of deposits. This can cause dilution of rewards when people accidentally send tokens to the masterchef. More severely, because the native token is constantly minted, this will cause severe dilution on the native token pool.
Recommendation	Consider adding an `lpSupply` variable to the `PoolInfo` that keeps track of the total deposits. All occurrences that use the masterchef token balance should also be replaced with the pool `lpSupply`, for example: Line 155 `uint256 lpSupply = pool.lpSupply;`
Resolution	RESOLVED The recommended change has been implemented but using `totalDeposited` as the variable name. We believe `totalDeposited` is a cleaner variable name and commend the client for considering code readability.

Issue #11	**Lack of events for `add`, `set`, `harvest`, `resetLockerAllowance`, `setPawTransferOwnerAddress` and `transferPawOwnership`**
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation	Add events for `add`, `set`, `harvest`, `resetLockerAllowance`, `setPawTransferOwnerAddress` and `transferPawOwnership`.
Resolution	RESOLVED

2.2 PawToken

2.2.1 Token Overview

The PawToken is a near perfect copy of the Goose Egg token contract (https://bscscan.com/address/0xf952fc3ca7325cc27d15885d37117676d25bfda6), a simple token with governance.

Address	TBC
Token Supply	Unlimited
Decimal Places	18
Transfer Max Size	None
Transfer Min Size	None
Transfer Fees	None

2.2.2 Privileged Roles

The following functions can be called by the owner of the contract:

- `mint`

2.2.3 Issues & Recommendations

Issue #12	**`mint` function can be used to pre-mint large amounts of tokens before ownership is transferred to the Masterchef**
Severity	INFORMATIONAL

Location	Lines 10–13
	`function mint(address _to, uint256 _amount) external onlyOwner`
	`{`
	`  _mint(_to, _amount);`
	`  _moveDelegates(address(0), _delegates[_to], _amount);`
	`}`

Description	The `mint` function could be used to pre-mint tokens for legitimate uses including, but not limited to, the injection of initial liquidity, token presale, or airdrops; however, this function may also be used to pre-mint tokens for dumping. As token ownership is already transferred to the Masterchef, the main risk that remains is whether any tokens were pre-minted to the project owner prior to transferring ownership.
Recommendation	Consider being forthright if this `mint` function has been used by letting your community know how much was minted, where they are currently stored, if a vesting contract was used for token unlocking, and finally the purpose of the mints. Since Polycat is an established project, we have decided to reduce the severity of this issue from low to informational.
Resolution	ACKNOWLEDGED

Issue #13	Governance functionality is broken
Severity	INFORMATIONAL
Description	Although there is YAM related delegation code in the token contract which is usually used for governance and voting, the delegation code can be abused as the delegates are not moved during transfers and burns. This allows for double spending attacks on the voting mechanism. It should be noted that this mistake is present in pretty much every single farm out there including PancakeSwap and even sushiswap.
Recommendation	The broken delegation related code can be removed to reduce the size of the contract. If voting is ever desired, it can still be done through snapshot.org, used by many of the larger projects.
Resolution	ACKNOWLEDGED The client is actually already using snapshot.org so this will definitely not be a problem.

Issue #14	`delegateBySig` can be frontrun and cause denial of service
Severity	INFORMATIONAL
Location	Line 118 `require(nonce == nonces[signatory]++, "EGG::delegateBySig: invalid nonce");`
Description	Currently if `delegateBySig` is executed twice, the second execution will be reverted. It is thus in theory possible for a bot to pick up `delegateBySig` transactions in the mempool and execute them before a contract can. The issue with this is that the rest of said contract functionality would be lost as well. This could be a problem in case it would have been executed by a contract that would have rewarded you for your delegation for example.
Recommendation	Consider adding the desired message sender in the structhash and requiring this desired sender to be equal to `msg.sender`. This reduces the problem to having the message sender be able to frontrun you which is okay if it is a reviewed contract.
Resolution	ACKNOWLEDGED This issue is irrelevant since the governance functionality is unused anyway.

Issue #15	Errors still reference the Goose EGG token
Severity	INFORMATIONAL
Description	Currently a lot of the errors still contain the EGG token in their error message. An example is the following error: Line 117 `require(signatory != address(0), "EGG::delegateBySig: invalid signature");` Although we actually like this as it makes it even easier for third-party reviewers to review this contract, it might be a trademark issue.
Recommendation	Consider whether this is a trademark issue, if not, no change is recommended, otherwise we recommend changing Goose references to Polycat Paw ones.
Resolution	RESOLVED All references have been replaced with PAW.

2.3 RewardLocker

The RewardLocker is a near perfect copy of Kyber Network's KyberRewardLocker. It allows contracts like the Masterchef to lock in reward tokens for a user. These tokens will then vest linearly over a duration set by the owner. Although the owner can change this vesting duration to a potentially high duration, previous vestings remain unaffected by this.

2.3.1 Privileged Roles

The following functions can be called by the owner of the contract:

- `addRewardsContract`
- `removeRewardsContract`
- `setVestingDuration`
- `updateMaxContractSize`

The following function can be called by the onlyRewardsContract:

- `lockWithStartBlock`

2.3.2 Issues & Recommendations

Issue #16	Setting the lock duration to a severely excessive value could revert locks
Severity	INFORMATIONAL
Location	Lines 117–155

```
function lockWithStartBlock(
  IERC20 token,
  address account,
  uint256 quantity,
  uint256 startBlock
) public override payable onlyRewardsContract(token) {
  require(quantity > 0, '0 quantity');

  ...

  uint256 endBlock =
startBlock.add(vestingDurationPerToken[token]);

  ...

  // append new schedule
  schedules.data[schedulesLength] = VestingSchedule({
      startBlock: startBlock.toUint64(),
      endBlock: endBlock.toUint64(),
      quantity: quantity.toUint128(),
      vestedQuantity: 0
  });

  ...
}
```

Description

Currently there is no limit on the lock duration for a token, this means that the contract owner could set it to an exceptionally large value.

In case this value is near the maximum value of an uint64, the cast of the `endBlock` to uint64 at line 152 will overflow, reverting any `lockWithStartBlock` call and thus also most deposits and withdrawals in the masterchef.

Recommendation	Consider adding a realistic value to the maximum vesting duration, for example 2 years.
	
	`function setVestingDuration(IERC20 token, uint64 _vestingDuration) external onlyOwner {`
	`  require(_vestingDuration <= MAX_VESTING_DURATION);`
	`  vestingDurationPerToken[token] = _vestingDuration;`
	`  emit SetVestingDuration(token, _vestingDuration);`
	`}`
	
	Having an explicit limit in the code might also help with investor confidence, all though the code is already structured in a way that updates to this variable do not affect past vestings.

Resolution	RESOLVED An approximately 1 year maximum duration has been added.

Issue #17	The frontend might become slow if a user has many vesting schedules
Severity	INFORMATIONAL
Description	Currently there's no easy way for the frontend to retrieve unvested schedules, thus to fetch all unvested schedules with certainty, the whole mapping of user schedules has to be looked through. Since this mapping only increases over time, this might heavily reduce the performance of the frontend over time (in case no API backed by an index is used).
Recommendation	In case there is no off-chain indexing API, consider adding functions that can retrieve only the schedules that are still vesting. This could be done by adding a `vestingIndices` mapping and `vestingIndicesLength` value to the `vestingSchedules` array that only contains the active indices. When schedules are completely vested, they are removed from this array (this needs to be done with great care however). This would then be used as a helper function for the UI to only retrieve the schedules that are still vesting.
Resolution	ACKNOWLEDGED This is a copy from the Kyber source code and the client has indicated that keeping the logic like this compared to making modifications is safer for their use cases.

Issue #18	Certain functions can revert due to the block gas limit
Severity	INFORMATIONAL
Description	Quite a few functions will eventually likely revert since they loop through all vesting schedules. As noted in the previous issue, the size of this schedules mapping is only bound to increase over time. `vestCompletedSchedules` and `vestCompletedSchedulesForMultipleTokens` are the most notable function this will happen to.
Recommendation	Consider whether these functions are used for any important functionality and if so consider replacing them with pagination alternatives.
Resolution	RESOLVED The client will not use these functions within their UI.

Issue #19	Locking mechanism will misaccount with transfer tax tokens
Severity	INFORMATIONAL

Location	Lines 117–130
	`function lockWithStartBlock(`
	`  IERC20 token,`
	`  address account,`
	`  uint256 quantity,`
	`  uint256 startBlock`
	`) public override payable onlyRewardsContract(token) {`
	`  require(quantity > 0, '0 quantity');`
	
	`  if (token == IERC20(0)) {`
	`  require(msg.value == quantity, 'Invalid msg.value');`
	`  } else {`
	`  // transfer token from reward contract to lock contract`
	`  token.safeTransferFrom(msg.sender, address(this), quantity);`
	`  }`
	
	Line 140
	`lastSchedule.quantity = uint256(lastSchedule.quantity).add(quantity).toUint128();`

Description	In case a `rewardToken` is added with a transfer tax, the quantity added to the user's vesting amount is the amount sent and not the amount retrieved. This would result in the `RewardLocker` having insufficient tokens over time in case it is used for tokens with a transfer tax. This issue is marked as informational as it seems like the `RewardLocker` is only used for the `PawToken`, since other tokens can be added, we decided to still include it.
Recommendation	Consider using the current standard of handling deposits, which is based on how Uniswap handles transfer fees:

Lines 117–132

```
function lockWithStartBlock(
  IERC20 token,
  address account,
  uint256 quantity,
  uint256 startBlock
) public override payable onlyRewardsContract(token) {
  require(quantity > 0, '0 quantity');

  if (token == IERC20(0)) {
    require(msg.value == quantity, 'Invalid msg.value');
  } else {
    // transfer token from reward contract to lock contract
    uint256 balanceBefore = token.balanceOf(address(this));
    token.safeTransferFrom(msg.sender, address(this), quantity);
    quantity = token.balanceOf(address(this)).sub(balanceBefore);
  }
```

❗ Note that this is introduces an unlikely reentrancy vulnerability in case the token allows such reentrancy (for example ERC777 tokens could allow this). Should this resolution be chosen, `lockWithStartBlock` would have to include a `reentrancyLock` to avoid this.

Resolution

RESOLVED

The client has also added a reentrancy guard to prevent it from being exploited through for example ERC777 tokens.

Issue #20	Lack of event for `updateMaxContractSize`
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation	Add an event for `updateMaxContractSize`. ❗ Note that if too many contracts are added, view functions like `getRewardsContractsPerToken` may break due to hitting the gas limit.
Resolution	RESOLVED


PALADIN
BLOCKCHAIN SECURITY