PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For Dragon Crypto Gaming

26 October 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 Dragon Crypto Gaming: Aurum on the Avalanche network. 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.

1.1 Summary

Project Name	Dragon Crypto Gaming: Aurum
URL	aurum.dragoncrypto.io
Platform	Avalanche
Language	Solidity

1.2 Contracts Assessed

Name	Contract	Live Code Match
DCAU	0x100Cc3a819Dd3e8573fD2E46D1E66ee866068f30	MATCH
MasterChef	0x0bE0d3D0C3A122B7F57b6119766880a83F95aE9f	MATCH
DragonNestSupporter	0x253f2BF92AA76e0E448A77ED98197f44EeC96a9F	MATCH
VaultChef	0xdEAE788795Bbc8126E7532EB94B7e6b9E5960cfF	MATCH
BaseStrategy	Dependency of StrategyMasterChef and StrategyMasterChefLP	MATCH
BaseStrategyLP	Dependency of StrategyMasterChef and StrategyMasterChefLP	MATCH
BaseStrategyLPSingle	Dependency of StrategyMasterChef and StrategyMasterChefLP	MATCH
StrategyMasterChefLP		PENDING
StrategyMasterChef		PENDING
Operators	Dependency of StrategyMasterChef and StrategyMasterChefLP	MATCH

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	5	5	-	-
Medium	9	9	-	-
Low	11	8	2	1
Informational	26	25	1	-
Total	**51**	**47**	**3**	**1**

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 DCAU

ID	Severity	Summary	Status
01	MEDIUM	Supply cap can be exceeded by minting a large amount of tokens when it is nearly reached	RESOLVED
02	LOW	`mint` function can be used to pre-mint large amounts of tokens before ownership is transferred to the Masterchef	PARTIAL
03	INFO	`_initialMint` and `_limitAmount` are not constant	RESOLVED

1.3.2 MasterChef

ID	Severity	Summary	Status
04	MEDIUM	Uncapped rewards per second can be abused by governance to mint and potentially sell a large amount of tokens	RESOLVED
05	MEDIUM	Nest withdrawals can revert when there are many pools	RESOLVED
06	MEDIUM	The updatePool function insufficiently accounts for the maximum supply cap which might temporarily lead to deposits and withdrawals reverting	RESOLVED
07	LOW	Lack of constructor validation	RESOLVED
08	LOW	pendingDcauOfDragonNest reverts when nestSupportersLength is set to zero	RESOLVED
09	INFO	Within updatePool, the logic that sets the emissions ending variable uses an outdated variable	RESOLVED
10	INFO	Gas optimization: updatePool supply mechanism can be made slightly more efficient	RESOLVED
11	INFO	Inconsistency: The deposit function validates that the pool actually exists while none of the other functions validate this	RESOLVED
12	INFO	Pending function is not consistent with update function as it does not incorporate the supply cap	RESOLVED
13	INFO	Rounding vulnerability to tokens with a very large supply can cause large supply tokens to receive zero emissions	RESOLVED
14	INFO	Lack of events for depositMarketFee	RESOLVED
15	INFO	_updatePoolDragonNest reverts when nestSupportersLength is 0	RESOLVED

1.3.3 DragonNestSupporter

ID	Severity	Summary	Status
16	LOW	_whitelistAllowance, _tokenIds and _currentSaleId are private	RESOLVED
17	INFO	Lack of events for activateSale, addWhitelist	RESOLVED
18	INFO	PUBLICSALETIMESTAMP can be made immutable	RESOLVED
19	INFO	devWallet is unnecessarily marked as payable	RESOLVED
20	INFO	addWhiteList requirement does not check current balance	RESOLVED
21	INFO	devWallet() and itemCost() can be made external	RESOLVED
22	INFO	_itemCost is initially set to zero	RESOLVED

1.3.4 VaultChef

ID	Severity	Summary	Status
23	MEDIUM	Strategy can be drained through withdrawAll in case the underlying withdraw function is vulnerable to reentrancy	RESOLVED
24	MEDIUM	Phishing: Malicious strategies could be added as a pool	RESOLVED
25	LOW	No token validation is done on strategy addition, potentially causing resetAllowances to irreversibly fail	RESOLVED
26	LOW	VaultChef does not work with reflective tokens	RESOLVED
27	LOW	Withdrawals could become irreversibly impossible due to overflow reversion for tokens with a very high supply	RESOLVED
28	INFO	resetAllowances and resetSingleAllowance should emit events	RESOLVED
29	INFO	EmergencyWithdraw event is unused	RESOLVED
30	INFO	Events do not contain important variables	PARTIAL
31	INFO	Gas optimization: Inconsistent usage of local scoped variables	RESOLVED

1.3.5 BaseStrategy

ID	Severity	Summary	Status
32	HIGH	If panic() and unpause() is ever called in sequence by the governance, an exploiter or governance can abuse the deposit function to steal half of all funds (all funds if repeated many times).	RESOLVED
33	HIGH	sharesRemoved on withdrawal is based upon wantLockedTotal after the underlying withdrawal occurs which causes vault value to decrease in case there are withdrawal fees or transfer taxes	RESOLVED
34	MEDIUM	Router can be swapped to siphon transfer fees to any EOA	RESOLVED
35	MEDIUM	Contract might be incompatible with strategies where the earned token is equal to the staked token	RESOLVED
36	LOW	Contract contains many variables which are exclusively used in the derivative protocols	PARTIAL
37	LOW	No non-zero validation before swaps can result in the transaction failing	RESOLVED
38	INFO	Lack of events for panic, unpanic, resetAllowances and setGov	RESOLVED
39	INFO	The panic function does not revoke approvals	RESOLVED
40	INFO	Unnecessary addition in _safeSwap timestamp	RESOLVED
41	INFO	Calling the funds staked in the underlying staking contract shares is a misnomer	RESOLVED
42	INFO	Slippage factor misses the intended point of slippage guards in uniswap protocols	RESOLVED
43	INFO	withdrawFeeFactor is initially set outside of the permitted withdrawal fee range	RESOLVED
44	INFO	Contract is likely incompatible with deflationary tokens	RESOLVED

1.3.6 BaseStrategyLP

ID	Severity	Summary	Status
45	HIGH	convertDustToEarned is completely broken as it tries to assign uninitialized memory	RESOLVED

1.3.7 BaseStrategyLPSingle

ID	Severity	Summary	Status
46	INFO	Unnecessary addition in _safeSwap timestamp	RESOLVED

1.3.8 StrategyMasterChefLP

ID	Severity	Summary	Status
47	MEDIUM	Lack of sanity checks in the constructor for `_wantAddress` and `_earnedAddress`	RESOLVED
48	LOW	Contract contains variables which are left uninitialized which could cause certain functionality to remain unusable	RESOLVED

1.3.9 StrategyMasterChef

ID	Severity	Summary	Status
49	HIGH	The `earn` function tries to assign to an uninitialized array which causes the earn function to revert	RESOLVED
50	HIGH	The `earn` function only swaps half or the earned amount to want tokens	RESOLVED

1.3.10 Operators

ID	Severity	Summary	Status
51	LOW	operators mapping can not be iterated over	ACKNOWLEDGED

2 Findings

2.1 DCAU

The DragonCryptoAurum token is a simple ERC-20 token which will be used as the main reward token for the Masterchef. It allows for DragonCryptoAurum tokens to be minted when the `mint` function is called by the owner of the contract, which at the time of deployment would be the The Dragon's Lair team. Users should therefore carefully inspect that the ownership of this contract has been transferred to the Masterchef. The token has a maximum supply of 155,000 tokens.

Of the emissions one-fifteenth are minted as a bonus to the governance. One-third of it is given to the DEVADDRESS wallet while the remainder is given to the GAMEADDRESS. Both of these addresses cannot be changed and can be easily inspected by third-parties.

2.1.2 Privileged Roles

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

- `mint`
- `renounceOwnership`
- `transferOwnership`

2.1.3 Issues & Recommendations

Issue #01	**Supply cap can be exceeded by minting a large amount of tokens when it is nearly reached**
Severity	MEDIUM SEVERITY

Location	Lines 16-18
	`function mint(address _to, uint256 _amount) public onlyOwner {`
	`    require(totalSupply() <= _limitAmount, "Dragon: Mint reached to limit");`
	`    _mint(_to, _amount);`

Description	The token contract has a maximum supply of 155,000 tokens. However, this maximum is only enforced by checking that the current supply is not larger than 155,000 tokens whenever new tokens are minted. However, this could be a problem when the maximum supply is nearly reached, for example, the current supply is 154,999 and a large amount of tokens is minted. In the example, if 2 tokens are minted, this passes without a problem since the current supply is still less than 155,000 tokens. However, the supply after the mint is in fact 155,001 tokens. This could be a high risk since it would allow the governance to mint an exceptionally large amount of tokens while the supply cap is not reached.
Recommendation	Consider also checking that the supply including the amount to mint is smaller or equal to the supply cap. `require(totalSupply() + _amount <= _limitAmount, "Dragon: Mint reached to limit");`
Resolution	RESOLVED The recommended update has been implemented to the limit requirement.

Issue #02	`mint` function can be used to pre-mint large amounts of tokens before ownership is transferred to the Masterchef
Severity	LOW SEVERITY
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 and dump tokens when the token contract has been deployed but before ownership is set to the Masterchef contract. This risk is prevalent amongst less-reputable projects, and any pre-mints can be prominently seen on the Blockchain.
Recommendation	Consider being forthright if this `mint` function is to be 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.
Resolution	PARTIALLY RESOLVED The client has stated in their docs that 50,000 tokens are to be used for SWAP and 5,000 for liquidity. They have indicated that part of their deployment process is immediately transferring ownership to the Masterchef. Users can therefore easily verify this themselves and this issue will be updated to fully resolved once the client indicates to us that this has been completed within the deployed contracts.

Issue #03	`_initialMint` and `_limitAmount` are not constant
Severity	INFORMATIONAL
Description	Variables that are never modified can be indicated as such with the constant keyword. This is considered best practice since it makes the code more accessible for third-party reviewers and saves gas.
Recommendation	Consider making the above variables explicitly constant.
Resolution	RESOLVED

2.2 MasterChef

The Masterchef is a contract forked from the Goose Masterchef. One of the key improvements the DCAU Masterchef has over the Goose Masterchef is that the deposit fees has been limited to a maximum of 4.01%. In addition, they have also implemented other recommended safety features.

The contract has been extended to have the ability to stake Dragon Nests (Dragon Nest pools) which will receive a portion of deposit fees, and these fees are distributed to all Dragon Nests from all regular pools.

Finally, this contract gives the NFT marketplace the ability to deposit its market fees (in native token only) into the Dragon Nest pool which will then be distributed amongst Nest stakers as well. It should be noted that the NFT marketplace was not within the scope of this audit and we are thus unable to provide any guarantees regarding the behavior of this contract.

2.2.1 Privileged Roles

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

- `add`
- `set`
- `setStartTime`
- `setDcauPerSecond`
- `setEmissionEndTime`
- `transferOwnership`
- `renounceOwnership`
- `depositMarketFee (NFT_MARKET)`

2.2.2 Issues & Recommendations

Issue #04	**Uncapped rewards per second can be abused by governance to mint and potentially sell a large amount of tokens**
Severity	MEDIUM SEVERITY
Description	When setting the rewards per second, there should be a cap on the amount to prevent abuse. If there is no such cap, governance could increase the emission rate to an absurd amount and have a single pool which only they could harvest to potentially mint a bunch of tokens to their private wallet. ❗ Note that `massUpdatePools` should also be called before updating this reward rate.
Recommendation	Cap the amount of rewards per second to reasonable value. Such a value for example could be 10 tokens. `require(_dcauPerSecond <= MAX_EMISSION_RATE, "Too high");` ❗ Note that this check should be enforced in the constructor and `setDcauPerSecond`.
Resolution	RESOLVED The emission rate is now capped at 1 token per second.

Issue #05	Nest withdrawals can revert when there are many pools
Severity	MEDIUM SEVERITY
Location	Line 428
Description	Withdrawing a nest requires paying out the fees from every pool, as the number of pools grows (and never shrinks), over time withdrawals will become impossible, trapping the nests in the contract.
Recommendation	Consider a pool limit cap of about 50, careful validation should be done to verify this pool limit does not cause withdrawals to run out of gas. This cap should be enforced within the add function. The client should furthermore carefully validate that 50 pools is in fact sufficiently small to still withdraw with the specific pool tokens they are using and network they are deploying on. `require(poolInfo.length <= 50, "Too many pools");`
Resolution	RESOLVED A maximum of 20 pools has been implemented. We expect this to be a safe maximum under most pool tokens. It should be noted that while this maximum is not reached, theoretically speaking the governance could add in a bad token that would lock in the nests. This is not noted as an explicit issue as we expect the value of nests to be close to zero anyways if the governance would ever turn as malicious as this to lock in nests without any profit.

Issue #06	**The `updatePool` function insufficiently accounts for the maximum supply cap which might temporarily lead to deposits and withdrawals reverting**
Severity	MEDIUM SEVERITY

Location	Lines 229-236
	`if (dcauTotalSupply > DCAU_MAX_SUPPLY) { /// >=`
	`dcauReward = 0;`
	`gameDevDcauReward = 0;`
	`} else if ((dcauTotalSupply + dcauReward) > DCAU_MAX_SUPPLY) {`
	`uint256 dcauSupplyRemaining = DCAU_MAX_SUPPLY - dcauTotalSupply;`
	`dcauReward = (dcauSupplyRemaining * 15) / 16;`
	`gameDevDcauReward = dcauSupplyRemaining - dcauReward;`
	`}`

Description	As the token does not allow minting more than the maximum supply, the client has included functionality that automatically adjusts the remaining rewards up to the limit that can still be minted. However, although an attempt was made, this logic still takes insufficient care for the fact that not the `dcauReward` is minted, but 16/15th of the `dcauReward` is minted. This is because 1/15th of the reward is minted as a bonus split amongst the `DEVADDRESS` and `GAMEADDRESS` wallets. Although the function body correctly adjusts the remainder if it detects that insufficient tokens can be minted, the detection of this state does not account for the fact that extra tokens will be minted to these two addresses. While this state is present, which we expect to only be temporarily, this would cause minting and therefore deposits and withdrawals to revert.
Recommendation	Consider adjusting the aforementioned clause to properly include the adjusted `dcauReward`. `} else if ((dcauTotalSupply + dcauReward * 16 / 15) > DCAU_MAX_SUPPLY) {`
Resolution	RESOLVED The function has been implemented in a slightly more elegant and optimal way by using (`dcauTotalSupply + dcauReward + gameDevDcauReward`) in the `else if` statement.

Issue #07	Lack of constructor validation
Severity	LOW SEVERITY
Location	Line 95 `constructor(`
Description	The constructor does not validate any of the input variables. This could lead to misconfiguration of the contract which could potentially prevent deposits from occurring.
Recommendation	Consider adding non-zero requirements for the `_feeAddress`, `_devAddress` and `_gameAddress`. Consider adding the limit to `_dcauPerSecond` here as well. Finally, consider validating that `startBlock` is set to a future block number.
Resolution	RESOLVED

Issue #08	`pendingDcauOfDragonNest` reverts when `nestSupportersLength` is set to zero
Severity	LOW SEVERITY
Location	Line 454 `uint256 accDepFeePerShare = poolDragonNest.accDepFeePerShare + _pendingDepFee / nestSupportersLength;`
Description	When `nestSupportersLength` is 0, a division by 0 occurs, causing the update to revert without an explicit error message.
Recommendation	Consider changing logic to account for `nestSupportersLength` being 0 and either just returning `poolDragonNest.accDepFeePerShare` or reverting with an error message.
Resolution	RESOLVED

Issue #09	**Within `updatePool`, the logic that sets the emissions ending variable uses an outdated variable**
Severity	INFORMATIONAL
Location	Line 251 `if (dcauTotalSupply >= DCAU_MAX_SUPPLY && emissionEndTime == type(uint256).max) emissionEndTime = block.timestamp;`
Description	At the end of the `updatePool` function, there is a check to see whether the `DCAU_MAX_SUPPLY` is exceeded. For this check however, the function uses an outdated `dcauTotalSupply` variable which has already likely increased during the execution of the function.
Recommendation	Consider whether this is desired behavior, and if not, consider using the current supply of the token.
Resolution	RESOLVED The cached `dcauTotalSupply` is now refreshed before this statement is executed.

Issue #10	**Gas optimization: updatePool supply mechanism can be made slightly more efficient**
Severity	INFORMATIONAL
Location	Line 229 `if (dcauTotalSupply > DCAU_MAX_SUPPLY) {`
Description	The `updatePool` function only assumes no tokens need to be minted if the `DCAU_MAX_SUPPLY` has been reached. It however also cannot mint any tokens if the supply is equal to the maximum. Not only would making such a change likely save some gas, it also better conveys the business logic behind this if clause.
Recommendation	Consider using >= instead of > in the above code section.
Resolution	RESOLVED

Issue #11	**Inconsistency: The `deposit` function validates that the pool actually exists while none of the other functions validate this**
Severity	INFORMATIONAL
Description	While the `deposit` function has a requirement that the pool id must exist, this requirement is not present on other functions like `withdrawal`, `emergencyWithdrawal`, `updatePool`, etc.
Recommendation	Consider including the pool id validation within all relevant functions.
Resolution	RESOLVED Pool id validation has now been added to the vast majority of functions.

Issue #12	**Pending function is not consistent with update function as it does not incorporate the supply cap**
Severity	INFORMATIONAL
Location	Line 451 `function pendingDcau(uint256 _pid, address _user) external view returns (uint256)`
Description	While the `update` function accounts for the supply cap by decreasing the rewards accordingly when it is nearly reached, this is not done within the pending rewards function which could mislead users that are looking at their pending rewards on the frontend.
Recommendation	Consider including the supply cap logic in the view function. In case this is cumbersome it is fine to leave this out as well and accept that the pending rewards might very temporarily be misleading on the frontend.
Resolution	RESOLVED

Issue #13	**Rounding vulnerability to tokens with a very large supply can cause large supply tokens to receive zero emissions**
Severity	INFORMATIONAL
Description	Within `updatePool`, deposit, withdraw and the pending rewards function, accDcauPerShare is based upon the lpSupply variable. `pool.accDcauPerShare = pool.accDcauPerShare.add(dcauReward.mul(1e12).div(pool.lpSupply)` However, if this `lpSupply` becomes a severely large value this will cause precision errors due to rounding. This is famously seen when pools decide to add meme-tokens which usually have huge supplies and no decimals.
Recommendation	Consider increasing precision to 1e18 across the entire contract. It should be noted that even a precision of 1e18 has been considered small when tokens like PolyDoge were added to Masterchefs of our client. In case the client thinks it is realistic that such tokens will be added, we recommend testing which precision variable is most appropriate to support them without potentially reverting due to overflows.
Resolution	RESOLVED The client has indicated that they have no intention to support such tokens and would rather keep the chances of overflow small.

Issue #14	Lack of events for `depositMarketFee`
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation	Add events for the above functions.
Resolution	RESOLVED An event has been added to the above function. The client has also removed the update which would cause this function to revert in case there were zero supporters. The next issue can be consulted to understand this case better, which was previously present.

Issue #15	`_updatePoolDragonNest` reverts when `nestSupportersLength` is 0
Severity	INFORMATIONAL
Location	Line 385
Description	When `nestSupportersLength` is 0, a division by 0 occurs, causing the update to revert without an explicit error message.
Recommendation	Consider adding a `require` with an explicit error message. `require(nestSupportersLength > 0, "Must have supporters");`
Resolution	RESOLVED The recommended requirement has been implemented causing a proper error message to occur whenever this code is ran with zero nest supporters.

2.3 DragonNestSupporter

The DragonNestSupporter contract is an NFT token contract which allows users to purchase dragon nest NFTs. Up to 25 dragon nests can be purchased for the item cost which is configurable by the owner. Up to two dragon nests can be owned by any individual wallet. Once the owner marks that the sale has started, the whitelisted addresses can purchase either one or two dragon nests depending on their allowance. Once the public sale timestamp (configurable during deployment) is reached, anyone else can still purchase the remaining dragon nests.

2.3.1 Privileged Roles

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

- `addWhiteList`
- `mintItem`
- `setItemCost`
- `transferOwnership`
- `renounceOwnership`

2.3.2 Issues & Recommendations

Issue #16	`_whitelistAllowance`, `_tokenIds` and `_currentSaleId` are `private`
Severity	LOW SEVERITY
Description	Important variables that third-parties might want to inspect should be marked as `public` so that these third-parties can easily inspect them through the explorer, web3 and derivative contracts.
Recommendation	Consider marking the above variables as public. ❗ Since `_tokenIds` and `_currentSaleIds` are counters, they in fact need a public view function.
Resolution	RESOLVED It should be noted that `_tokenIds` and `_currentSaleIds` have simply been made public which causes these variables to expose a struct including their value.

Issue #17	Lack of events for `activateSale` and `addWhitelist`
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation	Add events for the above functions. ❗ Don't forget to make `activateSale` only callable once
Resolution	RESOLVED

Issue #18	**PUBLICSALETIMESTAMP can be made `immutable`**
Severity	INFORMATIONAL
Description	Variables that are only set in the constructor but never modified can be indicated as such with the `immutable` keyword. This is considered best practice since it makes the code more accessible for third-party reviewers and saves gas.
Recommendation	Consider making the above variables explicitly `immutable`.
Resolution	RESOLVED

Issue #19	**devWallet is unnecessarily marked as payable**
Severity	INFORMATIONAL
Location	Line 22 `address payable private immutable _devWallet;`
Description	The `devWallet` is unnecessarily marked as payable as no native chain token transfers occur to it.
Recommendation	Consider removing the `payable` modifier.
Resolution	RESOLVED

Issue #20	**addWhiteList requirement does not check current balance**
Severity	INFORMATIONAL
Location	Line 50 `require(_whitelistAllowance[whitelistAddress] < 2, "already has 2 in allowance");`
Description	The `addWhiteList` function does not know whether the user already has dragons in their balance.
Recommendation	Consider whether it is desired to give a wallet a greater allowance than what they can actually purchase. If not, consider updating the requirement to the following code. `require(_whitelistAllowance[whitelistAddress] + balanceOf(whitelistAddress) < 2, "already has 2 in allowance");`
Resolution	RESOLVED

Issue #21	**devWallet() and itemCost() can be made external**
Severity	INFORMATIONAL
Description	Functions that are not used within the contract but only externally can be marked as such with the external keyword. Apart from being a best practice when the function is not used within the contract, this can lead to a lower gas usage in certain cases.
Recommendation	Consider marking the above variables as external.
Resolution	RESOLVED The relevant variables have been marked as public and these functions have been removed given that they would be redundant.

Issue #22	`_itemCost` is initially set to zero
Severity	INFORMATIONAL
Location	Line 21 `uint256 private _itemCost;`
Description	The cost per dragon is initially set to zero, if the deployer forgets to update this with the set function, users will be able to purchase dragon nests for free.
Recommendation	Consider adding an initial item cost or not allowing sale activation while it is set to zero.
Resolution	RESOLVED The client has updated the code as to not allow the sale to start until the item cost is set to a value greater than zero.

2.4 VaultChef

The VaultChef contract is the main entrypoint for users to interact with the DCAU vault system. It manages all strategies and user shares and takes care of the deposits and withdrawals of users into strategies.

2.4.1 Issues & Recommendations

Issue #23	**Strategy can be drained through `withdrawAll` in case the underlying withdraw function is vulnerable to reentrancy**
Severity	MEDIUM SEVERITY
Description	Although the project correctly contains a `nonReentrant` modifier on the `withdraw` function, this modifier is not present in the similar `withdrawAll` utility function. This allows an exploiter to potentially drain a strategy through reentrancy since the stake subtraction only happens after the withdrawal is called on the strategy. This issue is marked as Medium risk instead of High risk since the odds of a strategy allowing a non-privileged user to reenter are low.
Recommendation	Consider implementing the checks-effects-interactions pattern to prevent reentrancy, or introducing reentrancy guards to the function.
Resolution	RESOLVED `withdrawAll` has been guarded with a `nonReentrant` modifier.

Issue #24	Phishing: Malicious strategies could be added as a pool
Severity	MEDIUM SEVERITY
Description	There are no safeguards in place for investors to know if the underlying strategy contract in the pool which they stake their funds in is malicious, which may result in the loss of funds if they do not carefully inspect the poolId parameter in their staking function (which is a difficult thing to do on MetaMask). It should be noted that the existing stakes of users in old strategies are not affected if a malicious strategy would be added in later on, only depositors in this malicious strategy can be affected.
Recommendation	Consider adding a list of trusted strategy pool ids which have been either peer-reviewed or audited. Furthermore, consider educating the user base on how to validate the strategy contract against the trusted ones themselves. Finally, consider investing in reputation building mechanisms like publishing identities of team members or using multisig contracts to reduce the chances that an individual party would suddenly add bad strategies that could allow theft of user funds.
Resolution	RESOLVED The client has indicated that they are in fact KYC'd with our partner RugDoc which historically has reduced the chances of such behavior. The client has also indicated that the KYC'd party is the only one that can add strategies.

Issue #25	No token validation is done on strategy addition, potentially causing `resetAllowances` to irreversibly fail
Severity	LOW SEVERITY
Description	When an EOA or non-ERC20 contract is used as the `wantAddress` of a pool by accident within addPool, `resetAllowances` will fail forever since the `safeApprove` method will fail due to the allowance check in it, which would be called on a non-existent contract.
Recommendation	Consider adding in a test line to the `addPool` function, which will fail if the want token of the strategy does not contain a functioning allowance function. This should catch most of the cases where `resetAllowances` might fail. `IERC20(IStrategy(_strat).wantAddress()).allowance(address(this), address(_strat));`
Resolution	RESOLVED

Issue #26	VaultChef does not work with reflective tokens
Severity	LOW SEVERITY
Description	During the deposit function, the `_wantAmt` (amount of tokens to deposit) is forwarded to the strategy without validating if all these tokens actually arrived in the VaultChef. This will cause the implementation of any strategy with a transfer-tax token to fail and prevent deposits from happening.
Recommendation	Consider using the before-after pattern of handling deposits to find out how many tokens are actually transferred in through the `safeTransferFrom` function.
Resolution	RESOLVED The deposit function now incorporates the before-after pattern.

Issue #27	**Withdrawals could become irreversibly impossible due to overflow reversion for tokens with a very high supply**
Severity	LOW SEVERITY
Location	Line 122 `uint256 amount = (user.shares * wantLockedTotal) / sharesTotal;`
Description	The nominal usage balance calculation on line 122 stores a product of two large variables, if these two individual variables are too large this product might overflow causing the withdrawal to revert. This overflow occurs roughly when both components exceed 10**38. Most tokens we know of have a supply way lower than this variable making this scenario improbable.
Recommendation	As a somewhat inaccurate way of preventing this scenario from occurring, consider reverting deposits in case they cause `wantLockedTotal` to exceed 10**34. Although in theory the withdrawal case could still occur through `wantLockedTotal` increasing naturally over time, such a limit would make this scenario especially improbable. If the client would like to completely remove the possibility of this scenario, they can look at the `mulDiv` function introduced in Uniswap-v3 which does the arithmetic mentioned above without overflow concerns by strategically making use of assembly code. This code however could scare third-party investors as it is very low-level and to our knowledge not incorporated in the OpenZeppelin contracts yet.
Resolution	RESOLVED The client has indicated that they have no intention to supply any sort of token with such high supply, especially not meme coins.

Issue #28	`resetAllowances` and `resetSingleAllowance` should emit events
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation	Add events for the above functions.
Resolution	RESOLVED

Issue #29	`EmergencyWithdraw` event is unused
Severity	INFORMATIONAL
Description	The VaultChef contains an unused `EmergencyWithdraw` event. This could mislead third-parties into thinking an emergency withdraw function is present.
Recommendation	Consider removing the unused event.
Resolution	RESOLVED

Issue #30	Events do not contain important variables
Severity	INFORMATIONAL
Description	A portion of the events miss important variables that should likely be included in these events. The `AddPool` event does not contain a parameter indicating the pool id. The `Deposit` event does not contain `msg.sender`. The `Withdraw` event does not contain `to`.
Recommendation	Consider adding a pid parameter to the event and setting it to `poolInfo.length.sub(1)`.
Resolution	PARTIALLY RESOLVED The AddPool event has been updated, Deposit and Withdraw are left unchanged.

Issue #31	Gas optimization: Inconsistent usage of local scoped variables
Severity	INFORMATIONAL

Location	Lines 54-55
	`uint256 sharesTotal = IStrategy(pool.strat).sharesTotal();`
	`uint256 wantLockedTotal = IStrategy(poolInfo[_pid].strat).wantLockedTotal();`

Description	The code sometimes references storage variables by the cached "pool" variable and sometimes directly through `poolInfo[_pid]`. This functionality is inconsistent and might leave bad impressions with third-party reviewers.
Recommendation	Consider being consistent within the stakedWantTokens function, the `_deposit` function and `_withdraw` function by always referencing the local pool variable directly.
Resolution	RESOLVED

2.5 BaseStrategy

The BaseStrategy contract is owned by the VaultChef contract. It contains the fundamental logic that all other strategies use to successfully compound their staked token in underlying protocols and furthermore correctly interact with the management VaultChef contract.

2.5.1 Privileged Roles

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

- `deposit (vaultchef)`
- `withdraw (vaultchef)`
- `resetAllowances`
- `pause`
- `unpause`
- `panic`
- `setGov`
- `setSettings`

2.5.2 Issues & Recommendations

Issue #32	**If `panic()` and `unpause()` is ever called in sequence by the governance, an exploiter or governance can abuse the deposit function to steal half of all funds (all funds if repeated many times)**
Severity	HIGH SEVERITY
Description	After `panic()` is called by governance to withdraw all funds from the underlying staking contract, the funds remain idle in the strategy. However, when a deposit is made, the `_farm` function assumes that all funds added to the underlying staking contract were added by the user. An exploiter will simply call `deposit()` after a panic() and unpause() call is done to double all shares in the vault and be able to withdraw half of the vault's balance.
Recommendation	Consider removing the `unpause` function and renaming the unpanic function to `unpause` since it will correctly stake the idle balance again. In general, consider only calling `panic()` once to permanently disable the strategy. The reasoning is because calling `panic()` and `deposit()`/`harvest()` repeatedly could cause loss of funds by users and possibly gain by governance (for example in a strategy where the `earn` token address is equal to the staking token address, through performance fees). Finally it is considered very bad practice within vaults to use the total `.balanceOf` to figure out what needs to be staked, swapped... as this can often take amounts from other allocations. Consider adjusting the contract to only stake the actual deposited amount (while correctly accounting for transfer taxes).
Resolution	RESOLVED The recommendation to stake all tokens during the `unpause` has been added. To further limit governance risk, the client does no longer allow unpausing after a panic has occurred.

Issue #33	**`sharesRemoved` on withdrawal is based upon `wantLockedTotal` after the underlying withdrawal occurs which causes vault value to decrease in case there are withdrawal fees or transfer taxes**
Severity	HIGH SEVERITY
Description	The contract bases the `sharesRemoved` in the `withdraw` function on the ratio of `totalShares` over total stake tokens after the desired tokens are removed from the underlying protocol. As there are often frictions with unstaking tokens (withdrawal fees and transfer taxes), the `wantLockedTotal()` variable will often be smaller than at the beginning of the `withdraw` function which causes the `sharesRemoved` variable to be incorrect. Furthermore and potentially more importantly, it bases `sharesRemoved` on the `_wantAmt` that was actually received after a bunch of adjustments were made, not the `_wantAmt` that was actually to be removed from the users' share. Again, if there are transfer taxes or withdrawal fees, a `_wantAmt` way smaller than the truly withdrawn amount will be used to calculate the withdrawn shares causing the vault to lose value when withdrawals are done over strategies with withdrawal frictions.
Recommendation	Consider calculating the `sharesRemoved` variable based on the `_wantAmt` that was actually requested, instead of the amount that was received from the underlying protocol.
Resolution	RESOLVED

Issue #34	Router can be swapped to siphon transfer fees to any EOA
Severity	MEDIUM SEVERITY
Description	The Gov address can update the router that is used for LP pair generation to any contract they desire. This could very well be a malicious contract that takes all transfer fees and sends them to the operator. A malicious router could also allow reentrancy and manipulation through code injection.
Recommendation	Consider removing this ability to change routers, if this is not possible consider setting the Gov privilege behind a sufficiently long Timelock of for example 7 day to give users sufficient time to react appropriately if required.
Resolution	RESOLVED The router can no longer be swapped.

Issue #35	**Contract might be incompatible with strategies where the earned token is equal to the staked token**
Severity	MEDIUM SEVERITY
Description	The contract does deposits which are not preceded by harvests. As with most staking contracts, the deposit does in fact do a harvest which causes there to be idle earned tokens in the strategy contract after a deposit occurs. These tokens could then be taken by the next depositor. It should be noted that these tokens being equal might cause further side-effects like with the `earn` function in the derivative contracts, we have not pointed out this case in every instance we found it as we believe it is either best avoided, or fundamentally incorporated in the business logic.
Recommendation	Consider either fundamentally redesigning the contract to make it compliant with the earned and staked token being identical, or simply adding a requirement in the constructors that this case is not allowed.
Resolution	RESOLVED This has been resolved at the StrategyMasterchefLP level.

Issue #36	Contract contains many variables which are exclusively used in the derivative protocols
Severity	LOW SEVERITY
Description	The contract contains many variables like `token0address`, token1address and others which are more adequality stored in the derivative contracts like BaseStrategyLP and BaseStrategyLPSingle as they are unused within this contract. Although this issue does not have direct risk related to it, we decided to raise it to low severity as we saw it as not impossible that at some point some variable might be forgotten to be initialized, or some side-effect might be overlooked with this setup.
Recommendation	Consider moving all variables which are unused within the BaseStrategy to a more suitable upper layer contract.
Resolution	PARTIALLY RESOLVED The major variables which are the LP related ones have been moved to a higher level. However, other variables have not been moved to the correct code layers.

Issue #37	No non-zero validation before swaps can result in the transaction failing
Severity	LOW SEVERITY
Description	Uniswap will revert swaps with a zero amount.
Recommendation	Consider adding an `if (amountIn > 0)` check before all Uniswap router interactions.
Resolution	RESOLVED

Issue #38	Lack of events for `panic`, `unpanic`, `resetAllowances` and `setGov`
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation	Add events for the above functions.
Resolution	RESOLVED

Issue #39	The `panic` function does not revoke approvals
Severity	INFORMATIONAL
Description	Currently `panic` does not revoke approvals to the staking contracts and router. This is often done just in case it ever turns out that these staking contracts could transfer tokens from wallets without the wallets consent.
Recommendation	Consider revoking the approvals in the `panic` function.
Resolution	RESOLVED A `revokeAllowances` function has been added which is called during the `panic` function.

Issue #40	Unnecessary addition in `_safeSwap` timestamp
Severity	INFORMATIONAL
Description	Within the `_safeSwap` function, an unnecessary addition of 600 seconds is done to the timestamp. This is not necessary and wastes gas for no reason since the timestamp check in a Uniswap router is greater or equal then the present timestamp (and timestamps do not change within a single block).
Recommendation	Replace `now + 600` with `now`.
Resolution	RESOLVED

Issue #41	Calling the funds staked in the underlying staking contract shares is a misnomer
Severity	INFORMATIONAL
Description	The contract uses the term "shares" for funds staked in the QuickSwap staking contract. This is misleading for third party auditors since these tokens are in fact not shares but instead simply staked tokens. The function `vaultSharesTotal()` is misleading as well.
Recommendation	Consider renaming the `vaultSharesTotal()` function to for example `totalInUnderlying()`. It should be noted that there are variables that are correctly named shares and incorrectly named shares so the client should be careful to only rename the ones which denote actual `want` tokens.
Resolution	RESOLVED The client has renamed the relevant variables and functions to 'underlying' terminology.

Issue #42	Slippage factor misses the intended point of slippage guards in uniswap protocols
Severity	INFORMATIONAL
Description	The contract contains functionality to set a slippage factor, the maximum slippage that can occur during a swap while compounding. However, the minimum amounts in the Uniswap router interface were not really created to let derivative contracts prevent excessive slippage from occurring, instead the minimum amounts were created to allow you to define them off-chain, before actually creating the transfer as a way to combat "sandwich" attacks that wrap your swap and extract value from it through the created slippage.
Recommendation	Consider whether these slippage factors really provide any of the intended value.
Resolution	RESOLVED The client has indicated that they understand the limitations of this approach.

Issue #43	`withdrawFeeFactor` is initially set outside of the permitted withdrawal fee range
Severity	INFORMATIONAL
Description	The initial withdrawal fee is set to 10%, which is not allowed by the range of at most a 1% withdrawal fee. This will however not revert, but if the client wants to reduce the withdrawal fee they immediately will have to change it to 1% or lower.
Recommendation	Consider either broadening the range or increasing the `withdrawFeeFactor` to be within the range.
Resolution	RESOLVED The withdraw fee is now set to 1%, which is within the allowed range.

Issue #44	Contract is likely incompatible with deflationary tokens
Severity	INFORMATIONAL
Description	The contract is likely incompatible with deflationary tokens as it requires the `_wantAmt` in the deposit function to be equal to the tokens actually received by the contract, not the amount sent as is currently done.
Recommendation	Consider whether deflationary tokens need to be supported and then consider setting `_wantAmt` to the actually received tokens in the derivative protocols or fundamentally redesigning this vault system which might not be practical for most modern vault needs.
Resolution	RESOLVED Deposits now use the before-after pattern. Additionally, the Uniswap swapping functions have been adjusted to support deflationary tokens.

2.6 BaseStrategyLP

The BaseStrategyLP is a dependency contract which extends the BaseStrategy with functionaly to convert the dust tokens from lp swaps (the remaining values of the individual tokens after LP creation) back to the staking token.

2.6.1 Issues & Recommendations

Issue #45	**convertDustToEarned is completely broken as it tries to assign uninitialized memory**
Severity	HIGH SEVERITY
Description	The `convertDustToEarned` function will not work at all as it tries to assign variables to uninitialized memory arrays. This will always revert and cause this function to be unusable for any practical usage.
Recommendation	Consider initializing the two memory arrays.
Resolution	RESOLVED

2.7 BaseStrategyLPSingle

The BaseStrategyLPSingle contract is a dependency contract which extends BaseStrategyLP with the necessary compounding logic to harvest and compound into LP tokens.

2.7.1 Privileged Roles

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

- `earn`

2.7.2 Issues & Recommendations

Issue #46	Unnecessary addition in `_safeSwap` timestamp
Severity	INFORMATIONAL
Description	Within the `addLiquiditycall`, an unnecessary addition of 600 seconds is done to the timestamp. This is not necessary and wastes gas for no reason since the timestamp check in a Uniswap router is greater or equal then the present timestamp (and timestamps don't change within a single block).
Recommendation	Replace `now + 600` with `now`.
Resolution	RESOLVED

2.8 StrategyMasterChefLP

The StrategyMasterChefLP is a strategy contract which implements BaseStrategyLPSingle and is intended to compound an LP position within a Masterchef. It is owned by the VaultChef and users can deposit and withdraw into and out of the strategy through the VaultChef. Once deposited, users own a percentage of the strategy holdings and its compounded income.

2.8.1 Issues & Recommendations

Issue #47	**Lack of sanity checks in the constructor for `_wantAddress` and `_earnedAddress`**
Severity	MEDIUM SEVERITY
Description	There are no safeguards to ensure that the `_wantAddress` and `_earnedAddress` addresses are not set to the same token address, which may result in unintended negative consequences in the harvesting and withdrawal logic.
Recommendation	Consider setting the requirement in the constructor that prevents these address from overlapping, as such: `require(address(_wantAddress) != address(_earnedAddress), "wantAddress and earnedAddress cannot be the same");`
Resolution	RESOLVED

Issue #48	**Contract contains variables which are left uninitialized which could cause certain functionality to remain unusable**
Severity	LOW SEVERITY
Description	The contract contains multiple values which have not been initialized, the ones we've spotted are: - `token0ToEarnedPath` - `token1ToEarnedPath`
Recommendation	Consider following up on the initial recommendation that many values are wrongly located within the BaseStrategy contract as they are not used there. Consider removing any variable from the baseStrategy that is not used there and only defining it wherever it is used. Then consider validating which contracts have been forgotten to set and either remove or initialize them. The two variables we have identified seem to be obsolete and could be removed from the base contract.
Resolution	RESOLVED The variables have been removed.

2.9 StrategyMasterChef

2.9.1 Issues & Recommendations

Issue #49	The earn function tries to assign to an uninitialized array which causes the earn function to revert
Severity	HIGH SEVERITY
Description	The earn function will not work at all as it tries to assign variables to uninitialized memory arrays. This will always revert and cause this function to be unusable for any practical usage.
Recommendation	Consider initializing the two memory arrays.
Resolution	RESOLVED

Issue #50	The earn function only swaps half or the earned amount to want tokens
Severity	HIGH SEVERITY
Location	Line 64 `_safeSwap(earnedAmt / 2, path, address(this));`
Description	The earn function does not swap the complete earned amount to want tokens which causes there to be idle earned tokens in the strategy. This is very likely a mistake done by eagerly copy pasting from the LP contract without adjusting this division.
Recommendation	Consider removing the division by two and more importantly carefully testing all contracts extremely carefully before deploying them into production as this step seems to be lacking at this point.
Resolution	RESOLVED

2.10 Operators

The Operators contract is a dependency used to provide governance access to addresses other than the owner. It allows the owner to approve and disapprove operators.

2.10.1 Privileged Roles

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

- `updateOperator`
- `transferOwnership`
- `renounceOwnership`

2.10.2 Issues & Recommendations

Issue #51	`operators` mapping can not be iterated over
Severity	LOW SEVERITY
Description	The contract does not contain functionality to iterate over the `operators` mapping. This might cause distrust amongst investors as there is no easy way for them to inspect which wallets have operator privileges. The only way for an investor to discover this is to either go through all owner transactions, or inspect events using web3. Both these methods are cumbersome and unlikely ideal for most investors.
Recommendation	Consider using an EnumerableSet to list the operators and allow third-parties to easily iterate over them by both exposing the length of this set and a function to get the operator at a specific index. This way investors can easily just go through the list of operators one by one.
Resolution	ACKNOWLEDGED


PALADIN
BLOCKCHAIN SECURITY