PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For CopyCat Finance

20 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 CopyCat Finance. 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.

This is a best efforts review and Paladin has strived to identify all possible issues in a short period of time. A comprehensive list of issues has been presented, though there may yet be an unspecified amount that remains to be discovered due to the complexity and novelty of the various contracts of the protocol.

1.1 Paladin Round Table

This audit is a Paladin Round Table audit. As Paladin is a user-safety focused auditor, we may identify and mark issues as higher risk than what may be considered the norm in the industry since we tend to take the perspective of the user when approaching the audit.

Projects where the leadership require more governance freedom than usual will fall under our Round Table audit. These projects are usually more complex than most others, and thus the project leadership requires the ability to navigate around and react appropriately to bugs and exploits, such as having the ability to correct mistakes and migrate funds to new contracts. This ability however could mean that the project leadership has full control over staked funds, assets, tokens and many of the key aspects of the protocol.

If the leadership becomes malicious or the private keys get compromised, this could have negative effects on investor funds, including but not limited to loss of all staked or approved funds. Investors should thus carefully consider the reputation, honesty and experience of the teams that participate in our Round Table audits apart from carrying out standard due-diligence.

1.2 Summary

Project Name	CopyCat Finance
URL	https://copycat.finance
Platform	Binance Smart Chain
Language	Solidity

1.3 Contracts Assessed

The contracts were provided to Paladin in a GitHub repository. We will update this report after the contracts have been deployed to the blockchain and verify that the code matches the ones that we have audited.

Name	Contract	Live Code Match
MasterChef	0x9a1B69F216Ea3b9d7c4F938eDAEA0dAe7E758C17	MATCH
CopyCatToken	0xd635B32688F36ee4a7FE117b4C91DD811277ACB6	MATCH
Mintable	in CopyCatToken	MATCH
CopyCatTokenPresale	0x10bc672722ce21f2c09328f4a82eacab00e42e67	MATCH
CopyCatReferral	0x814Ed43b6696d18b58EE8061590Ee87C85585EdE	MATCH
CopyCatPresale	0x82AC62A1105Db799d4bC3F2B40622f0B43E0448d	MATCH
CopyCatSmartDeposit	0x65c68F43b598FA86aEC1D0D56BfFd7b71f29C1f9	MATCH
CopyCatEmergency	in CopyCatLeader	MATCH
CopyCatEmergencyMaster	in CopyCatLeaderFactory	MATCH
CopyCatReserver	0x935D09B9654728D23d6211708B5E9cB3Cf7362FA	MATCH
CopyCatLeader	0xaf596ebeaf7b06571b39a5f88674b65832eaa6b8	MATCH
CopyCatLeaderFactory	0x100D1FC9b25ce37BC01cDa165Aa9374c915672C7	MATCH
CopyCatAdapterFactoryBase	0x9282Ec84A345D864aa0821ea162aCa3ec41E77F3	MATCH
CopycatUniswapV2Adapter	0x23e50319502fbf55a0155191c8544bfc6c3d8dba	MATCH
CopycatUniswapV2AdapterFactory	0x9282Ec84A345D864aa0821ea162aCa3ec41E77F3	MATCH
Timelock	0xDf8Fa02755E4247780B17323299c59B11751eE0A	MATCH

1.4 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	10	8	2	-
Medium	9	5	1	3
Low	9	6	-	3
Informational	32	15	1	16
Round Table	2	-	-	-
Total	**60**	**34**	**4**	**22**

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.

1.4.1 MasterChef

ID	Severity	Summary	Status
01	HIGH	Severely excessive rewards issue when a token with a transfer tax is added	RESOLVED
02	MEDIUM	`harvestLockedReward`, `harvestRefReward` and `harvestToOtherReward` functions are vulnerable to reentrancy	RESOLVED
03	LOW	Overflow possible in `transferReward` function	RESOLVED
04	LOW	Adding an EOA or non-token contract as a pool will break `updatePool`, `massUpdatePools` and `updateEmissionRate`	RESOLVED
05	LOW	Setting `feeAddress` to the zero address will break deposit functionality	RESOLVED
06	LOW	`updateMaxEmissionRate` has no maximum safeguard	RESOLVED
07	INFORMATIONAL	The `pendingCPC` function will revert if `totalAllocPoint` is zero	PARTIALLY RESOLVED
08	INFORMATIONAL	`dev` function and by extension `devAddr` can be removed since they're not used anymore	RESOLVED
09	INFORMATIONAL	Rounding vulnerability to tokens with a very large supply can cause large supply tokens to receive zero emissions	ACKNOWLEDGED
10	INFORMATIONAL	Lack of event on `updateStartBlock`	ACKNOWLEDGED
11	INFORMATIONAL	`// cpc.mint(devaddr, cpcReward.div(10));` comment can be removed	RESOLVED
12	INFORMATIONAL	`cpc`, `cpcPerBlockMax`, `reserver` and `refContract` can be made `immutable`	RESOLVED
13	INFORMATIONAL	`massUpdatePools` can run out of gas in `add` and `set`	RESOLVED
14	INFORMATIONAL	`setAllowOtherTokenHarvest` function should be reordered below	RESOLVED
15	INFORMATIONAL	`BONUS_MULTIPLIER` has no function	ACKNOWLEDGED
16	INFORMATIONAL	Duplicate pools can be added if all other pools of the token have no emissions (this has no bad consequences)	RESOLVED

1.4.2 CopyCatToken

No issues found.

1.4.3 Mintable

ID	Severity	Summary	Status
17	INFORMATIONAL	No easily accessible array with the set of minters	RESOLVED

1.4.4 CopyCatTokenPresale

ID	Severity	Summary	Status
18	LOW	`mint` function can be used to pre-mint large amounts of pre-sale and eventually CopyCat tokens	ACKNOWLEDGED
19	INFORMATIONAL	CopycatTokenPresale (and the underlying CopyCatToken) remains mintable after the presale is concluded	ACKNOWLEDGED
20	INFORMATIONAL	`upgrade` function has no sensible error message while the `realToken` hasn't been set	ACKNOWLEDGED
21	INFORMATIONAL	Lack of event for `setUpgradeAddress`	ACKNOWLEDGED

1.4.5 CopyCatReferral

ID	Severity	Summary	Status
22	MEDIUM	`increaseRefCpc` allows governance to disallow certain addresses the capability to deposit, withdraw or harvest	PARTIALLY RESOLVED
23	INFORMATIONAL	The `refCount` variable that counts the number of active referrals actually only counts the referral codes	ACKNOWLEDGED

1.4.6 CopyCatPresale

ID	Severity	Summary	Status
24	LOW	buyCPC function can be called even if all tokens have been sold	ACKNOWLEDGED
25	INFORMATIONAL	buyCPC function sends referral tokens to the owner function in certain cases	ACKNOWLEDGED
26	INFORMATIONAL	`maxBnb_` can cause buyCPC and `calculateReceive` functions to fail if initialized with too low value, nor does it have a maximum safeguard	ACKNOWLEDGED
27	INFORMATIONAL	Sending too little BNB will cause `calculateReceive` to fail	ACKNOWLEDGED
28	INFORMATIONAL	Lack of constructor parameter validation	ACKNOWLEDGED
29	INFORMATIONAL	`token_`, `startTimestamp_`, `endTimestamp_`, `tokenPerLP_`, `maxBnb_` and `refContract_` can be made `immutable`	ACKNOWLEDGED
30	INFORMATIONAL	Inconsistent notation between parameters	ACKNOWLEDGED
31	INFORMATIONAL	Inconsistent indentation in buyCPC function	ACKNOWLEDGED

1.4.7 CopyCatSmartDeposit

ID	Severity	Summary	Status
32	HIGH	An exploiter can give a router approval to withdraw tokens at any time	RESOLVED
33	MEDIUM	Any token or BNB value can be taken out of the contract due to a reentrancy exploit	RESOLVED
34	INFORMATIONAL	Phishing: User could be mislead in approving a dangerous deposit or withdraw transaction	ACKNOWLEDGED

1.4.8 CopyCatEmergency

ID	Severity	Summary	Status
35	HIGH	Anyone can execute and re-execute a transaction	RESOLVED

1.4.9 CopyCatEmergencyMaster

ID	Severity	Summary	Status
36	INFORMATIONAL	If a timelock allower is added, it can still be circumvented by simply introducing a new allower	ACKNOWLEDGED

1.4.10 CopyCatReserver

ID	Severity	Summary	Status
37	INFORMATIONAL	There is no locking functionality in the CopyCatReserver	RESOLVED
38	INFORMATIONAL	No event on `transfer`	RESOLVED
39	INFORMATIONAL	`cpc` can be made `immutable`	RESOLVED

1.4.11 CopyCatLeader

ID	Severity	Summary	Status
40	HIGH	Price manipulation exploit allows leader owner to profitably drain the leader	RESOLVED
41	HIGH	Withdrawals wrongly affect share value which can lead to exploitation and losses to other stakers	PARTIALLY RESOLVED
42	MEDIUM	Share values may not be strictly safe from various possible methods of exploitations	ACKNOWLEDGED
43	MEDIUM	Malicious contracts are currently unrestricted from executing exploits, if any	RESOLVED
44	MEDIUM	Leader owner can burn the whole leader balance at any time	RESOLVED
45	LOW	Deposit malfunctions (eg. by price manipulation) could give the user more BNB value then they deposited	RESOLVED
46	INFORMATIONAL	Function returns multiple distinct values in an array which misleads reviewers	RESOLVED
47	INFORMATIONAL	getAdapters can run out of memory	RESOLVED
48	INFORMATIONAL	A large amount of comments can be deleted	RESOLVED

1.4.12 CopyCatLeaderFactory

ID	Severity	Summary	Status
49	HIGH	Migration can be done by anyone resulting in anyone being able to steal all funds from another Leader	RESOLVED
50	HIGH	Reentrancy exploit allows for two leaders to exist under the same `leaderId`	RESOLVED
51	HIGH	Malicious code injection exploit allows to execute code before the leader is initialized allowing an exploiter to add malicious adapters and pre-mint to their leader	RESOLVED
52	MEDIUM	No validation done on deposit and withdraw fee could allow governance to lock in all funds	RESOLVED
53	MEDIUM	Setting the `feeAddress` to zero can resulting in blocking all functionality	ACKNOWLEDGED
54	MEDIUM	Governance privilege: Governance could add malicious adapters	ACKNOWLEDGED
55	LOW	No validation done on `setFee` allows governance to update many fee parameters to severe levels	ACKNOWLEDGED

1.4.13 CopyCatAdapterFactoryBase

ID	Severity	Summary	Status
56	HIGH	Governance privilege: Owner can execute any function on the adapter after at any time	PARTIALLY RESOLVED

1.4.14 CopycatUniswapV2Adapter

ID	Severity	Summary	Status
57	LOW	Routes that allow reentrancy could allow for balance inflation	RESOLVED

1.4.15 CopycatUniswapV2Adapter

ID	Severity	Summary	Status
58	HIGH	Governance privilege: Governance can set fee rate up to 100% resulting in all swaps going to the governance	RESOLVED
59	INFORMATIONAL	Lack of validation on trading routes	RESOLVED

1.4.16 Timelock

ID	Severity	Summary	Status
60	INFORMATIONAL	Gas efficiency: Usage of inefficient SafeMath version	RESOLVED

1.4.17 Round Table

ID	Location	Summary
61	CopyCatEmergency	Governance privilege: Governance can execute any function on all contracts
62	CopyCatLeader	Governance risk: Any registered adapter factory can add adapters to any leader allowing governance to drain all leaders

2 Findings

2.1 MasterChef

The CopyCat Masterchef is a staking contract based on Goose Finance's Masterchef with a special referral mechanism and harvest lockups. Emission rewards are minted to the Reserver contract. 2% of harvests is paid out as referral rewards (configurable up to 10%), after which 30% of harvests are unlocked and 70% remains locked. Every time rewards are harvested, `unlockTime` is reset to 3 days in the future. On every deposit and withdrawal, the `unlockTime` is reset to 3 days in the future as well. However, harvesters still get the 30% unlocked rewards instantly.

Rewards are minted to the Reserver, which then transfers out the tokens/rewards, so the Reserver should not have any locking abilities.

2.1.1 Issues & Recommendations

Issue #01	**Severely excessive rewards issue when a token with a transfer tax is added**
Severity	HIGH SEVERITY
Description	When tokens with a transfer tax are added to the pools, this will result in significant excessive rewards. Due to the way the Masterchef handles rewards, rewards can be heavily inflated when the balance of the Masterchef no longer matches that of user deposits. This happens for example with transfer tax tokens. This issue is further amplified on Masterchefs like this one with a referral mechanism, since these can mint tokens directly. This flaw in the Masterchef has recently been exploited on a significant number of projects, the most recent of which was PolyYeld. In all cases, their native tokens went to $0 because the exploit resulted in an egregiously large number of tokens being minted and dumped. This issue was also present in SushiSwap (the original Masterchef) and is present in pretty much every Masterchef since. Since they were never meant to have any tokens but LP tokens, it was not a problem there but has become a problem to projects who have started forking it for usage with less standard tokens.

Recommendation	Consider using the current standard of handling deposits, which is based on how Uniswap handles transfer fees:
	
	`uint256 balanceBefore = pool.lpToken.balanceOf(address(this));`
	`pool.lpToken.transferFrom(msg.sender, address(this), _amount);`
	`_amount =`
	`pool.lpToken.balanceOf(address(this)).sub(balanceBefore);`

Resolution	RESOLVED Resolved using the recommended before-after method.

Issue #02	**harvestLockedReward, harvestRefReward and harvestToOtherReward functions are vulnerable to reentrancy**
Severity	MEDIUM SEVERITY
Description	These functions may be subject to reentrancy attacks, allowing draining of the native token if a bad token is added (eg. an ERC777 token).

Recommendation	Add the nonReentrant feature to each function, and arranging the order such that user balances are set to 0 before transfers are made would be a safe practice in line with the Check Effects Interactions pattern (https://fravoll.github.io/solidity-patterns/checks_effects_interactions.html).
	
	For example, in the harvestLockedReward function, the Check Effects Interactions pattern can be implemented as such:
	
	`event HarvestLockedReward(address indexed harvester, uint256 amount);`
	`function harvestLockedReward() public {`
	`  UserReward storage user = userReward[msg.sender];`
	`  require(user.unlockTime <= now, "Locked");`
	`  user.lockedReward = 0;`
	`  safeCPCTransfer(msg.sender, user.lockedReward);`
	`  emit HarvestLockedReward(msg.sender, user.lockedReward);`
	`}`

Resolution	RESOLVED Resolved by adding nonReentrant to the functions.

Issue #03	Overflow possible in `transferReward` function
Severity	LOW SEVERITY
Description	As this contract uses Solidity version 0.6.12, all mathematical operations in the `transferReward` function may be subject to overflow risks due to the use of the '+' arithmetic operator.
Recommendation	Consider using safeMath's `add` instead of '+'.
Resolution	RESOLVED Resolved by implementing SafeMath's `add` and `sub`.

Issue #04	Adding an EOA or non-token contract as a pool will break `updatePool`, `massUpdatePools` and `updateEmissionRate`
Severity	LOW SEVERITY
Description	`updateEmissionRate` will always call `balanceOf(address(this))` on the token of the pool, and will fail if the token is not an actual token contract address.
Recommendation	Consider simply adding a test line in the add function. If the token does not exist, this will make sure the add function fails. `_lpToken.balanceOf(address(this));`
Comments	RESOLVED

Issue #05	Setting `feeAddress` to the zero address will break `deposit` functionality
Severity	LOW SEVERITY
Description	Within most token contracts, transferring tokens to the zero address will revert the transaction. Deposits will thus break if the `feeAddress` is ever set to the zero address. Deposit based harvests will break as well.
Recommendation	To prevent this from ever happening by accident and to limit governance risks, consider adding a requirement like the following: `require(_feeAddress != address(0), "!nonzero");` to the `setFeeAddress` function.
Resolution	RESOLVED

Issue #06	`updateMaxEmissionRate` has no maximum safeguard
Severity	LOW SEVERITY
Description	Projects sometimes accidentally update their emission rate to a severely high number either by accident or with malicious intent. Currently, `cpcPerBlockMax` can be set to any uncapped number.
Recommendation	Consider adding a `MAX_EMISSION_RATE` variable and setting it to a reasonable value. `require(_cpcPerBlock <= MAX_EMISSION_RATE,"Too high");`
Resolution	RESOLVED The maximum is now 100 tokens per block.

Issue #07	**The `pendingCPC` function will revert if `totalAllocPoint` is zero**
Severity	LOW SEVERITY
Description	In the `pendingCPC` function, at some point a division is made by the `totalAllocPoint` variable. If all pools have their rewards set to zero, this variable will be zero as well. The requests will then revert with a division by zero error.
Recommendation	Consider only calculating the accumulated rewards since the `lastRewardBlock` if the `totalAllocPoint` variable is greater than zero. This check can simply be added to the existing check that verifies the `block.number` and `lpSupply`, like so: `if (block.number > pool.lastRewardBlock && lpSupply != 0 && totalAllocPoint > 0) {`
Resolution	PARTIALLY RESOLVED The client has stated that they will not set `totalAllocPoint` to 0.

Issue #08	**`dev` function and by extension `devAddr` can be removed since they're not used anymore**
Severity	INFORMATIONAL
Description	As both the function and variable do not seem to be used anywhere, this can be removed for improved readability of the contracts.
Recommendation	Consider removing both the `dev` function and `devAddr` variable.
Comments	RESOLVED

Issue #09	Rounding vulnerability to tokens with a very large supply can cause large supply tokens to receive zero emissions
Severity	INFORMATIONAL
Description	Within `updatePool`, `accCPCPerShare` is based upon the `lpSupply` variable. `pool.accCPCPerShare.add(cpcReward.mul(1e12).div(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	Rounding vulnerability to tokens with a very large supply can cause large supply tokens to receive zero emissions.
Resolution	ACKNOWLEDGED

Issue #10	Lack of event on `updateStartBlock`
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation	Add an event for `updateStartBlock`.
Comments	ACKNOWLEDGED

Issue #11	`// cpc.mint(devaddr, cpcReward.div(10));` comment can be removed
Severity	INFORMATIONAL
Description	As this has been commented out, it can be removed from the contract for easier readability.
Recommendation	Consider removing the comment.
Comments	RESOLVED

Issue #12	`cpc`, `cpcPerBlockMax`, `reserver` and `refContract` 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.
Recommendation	Consider making `cpc`, `cpcPerBlockMax`, `reserver` and `refContract` variables `immutable`.
	RESOLVED

Issue #13	`massUpdatePools` can run out of gas in add and `set`
Severity	INFORMATIONAL
Description	Within the `add` and `set` function, a mandatory `massUpdatePools` check is done. This goes through all pools to update the rewards and is done to ensure that the previous rewards are updated properly before the reward allocations are changed. However, if there are many pools, this might cause the function to run out of gas. We are just pointing this out as an informational issue since it might not be bad to not allow too many pools to be added this way. But since we are unsure whether this is intentional we are still raising it as a temporary issue.
Recommendation	The issue will be resolved either by verifying that this behavior is appropriate or by fixing it through a `withUpdate` boolean parameter.
Resolution	RESOLVED The client has stated that this behaviour is appropriate.

Issue #14	`setAllowOtherTokenHarvest` function should be reordered below
Severity	INFORMATIONAL
Description	Functions, especially those that change sensitive state variables, are generally ordered below the constructor, as with all other functions in the contract, for easier readability.
Recommendation	Consider reordering this function below.
Resolution	RESOLVED

Issue #15	`BONUS_MULTIPLIER` has no function
Severity	INFORMATIONAL
Description	The `BONUS_MULTIPLIER` is defined and set to 1 but it serves no function since it is unchangeable. `cpcPerBlock` also already properly indicates the current emission rate so this provides no useful information either.
Recommendation	Consider removing the `BONUS_MULTIPLIER` variable.
Resolution	ACKNOWLEDGED

Issue #16	Duplicate pools can be added if all other pools of the token have no emissions (this has no bad consequences)
Severity	INFORMATIONAL
Description	The code contains a comment that explains why in exceptional cases duplicate pools can be added (two pools with the same token): `// For emergency fix of 100% withdrawal fee` As we do not understand this withdrawal fee issue we are raising this as an issue pending clarification.
Recommendation	This issue can be resolved simply by explaining the use case for this. In case the use case ends up invalid, consider removing this business logic to add multiple pools.
Resolution	RESOLVED The logic has been removed.

2.2 CopyCatToken

The CopyCat token is a simple ERC20 token contract with only 1 privileged function – `mint` – that is performed by designated minters. The other two public functions are `burn` and `burnFrom`. No issues were found.

2.2.1 Token Overview

Address	
Token Supply	200,000,000 (two hundred million)
Decimal Places	18
Transfer Max Size	No maximum
Transfer Min Size	No minimum
Transfer Fees	None

2.2.2 Issues & Recommendations

No issues found.

2.3 Mintable

2.3.1 Issues & Recommendations

Issue #17	No easily accessible array with the set of minters
Severity	INFORMATIONAL
Description	For a third-party reviewer to discover who the active minters are, they need to go through all `AllowMinter` events. This is considered a cumbersome task for simple third-party reviewers and could be made easier with a simple track record array. Many third-party reviewers will be interested in the list of minters.
Recommendation	Consider adding a simple `minterHistory` addresses array which is appended every time a minter change is made. Arrays are easily traversable for third-party reviewers. In case the project is ambitious, a state array `minters` could be set up which removes addresses again once they are removed.
Resolution	RESOLVED The client has added a `minters` array.

2.4 CopyCatTokenPresale

2.4.1 Issues & Recommendations

Issue #18	**`mint` function can be used to pre-mint large amounts of pre-sale and eventually CopyCat tokens**
Severity	LOW SEVERITY

Location	Lines 17-20
	`function mint(address _to, uint256 _amount) public onlyMinter{`
	`  increaseMint(_amount);`
	`  _mint(_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 and dump tokens. 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	ACKNOWLEDGED The client has stated that this contract is to be used only once. This issue shall be marked as resolved after the presale is finished and ownership has been renounced.

Issue #19	CopycatTokenPresale (and the underlying CopyCatToken) remains mintable after the presale is concluded
Severity	INFORMATIONAL
Description	After the presale is concluded, the developer can still mint presale tokens (and by extension Copycat tokens). Investors might find it valuable if this privilege is automatically revoked after the presale.
Recommendation	Consider adding a `finished` boolean that is set to `true` when `setUpgradeAddress` is called. Minting should no longer be allowed once finished is set to true. This can be implemented through an explicit requirement in the `mint` function. `require(finished == true, "Presale is already finished");`
Resolution	ACKNOWLEDGED The client has stated that this contract is to be used only once. This issue shall be marked as resolved after the presale is finished and ownership has been renounced.

Issue #20	**upgrade function has no sensible error message while the realToken hasn't been set**
Severity	INFORMATIONAL

Location	Lines 38-43
	`function upgrade() external {`
	`  uint256 amount = balanceOf(msg.sender);`
	`  _burn(msg.sender, amount);`
	`  realToken.mint(msg.sender, amount);`
	`  emit Upgrade(msg.sender, amount);`
	`}`

Description	The upgrade function has no sensible error message if the token is not yet set. This might cause confusion among investors.

Recommendation	Consider adding the following requirement:
	`function upgrade() external {`
	`  require(realToken != address(0), "Not ready");`
	`  uint256 amount = balanceOf(msg.sender);`
	`  _burn(msg.sender, amount);`
	`  realToken.mint(msg.sender, amount);`
	`  emit Upgrade(msg.sender, amount);`
	`}`

Resolution	ACKNOWLEDGED The client has stated that this contract is to be used only once. This issue shall be marked as resolved after the presale is finished and ownership has been renounced.

Issue #21	Lack of event for `setUpgradeAddress`
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation	Add events for `setUpgradeAddress`.
Resolution	ACKNOWLEDGED The client has stated that this contract is to be used only once. This issue shall be marked as resolved after the presale is finished and ownership has been renounced.

2.5 CopyCatReferral

The CopyCatReferral contract allows for the storage of referral addresses. It furthermore contains some frontend functionality to store shortcodes which are linked to an address.

2.5.1 Issues & Recommendations

Issue #22	**`increaseRefCpc` allows governance to disallow certain addresses the capability to deposit, withdraw or harvest**
Severity	MEDIUM SEVERITY
Description	Since any harvest reward is added to `refCpc`, in case this addition becomes too large and overflows, the harvest (and thus deposits and withdrawals too) will always overflow. This in theory allows a malicious owner to target specific addresses and set their `refCpc` to the maximum value, making them unable to withdraw.
Recommendation	The simplest solution might just be to wrap the `increaseRefCpc` call in a solidity try-catch block within the Masterchef.
Resolution	PARTIALLY RESOLVED The client has reiterated that they will not be maliciously calling this function.

Issue #23	**The `refCount` variable that counts the number of active referrals actually only counts the referral codes**
Severity	INFORMATIONAL
Description	When a new referral code is registered, the `refCount` is incremented. However, the actual referral system in the Masterchef does not use the referral codes, instead it can set referral addresses directly. Thus there will be more referrals in reality then `refCount` indicates. As this might very well be desired behaviour this issue is marked as informational.
Recommendation	Consider whether this is desired behavior and if not a new count could be added.
Resolution	ACKNOWLEDGED

2.6 CopyCatPresale

This is the presale contract which users will interact with in order to purchase presale tokens. There is a minimum of 0.3 BNB and an upper limit defined by `maxBnb`. Referral rewards are paid out on presale purchases. The contract uses `payable` which allows for functions to receive BNB without reverting, and amount = `msg.value`.

2.6.1 Issues & Recommendations

Issue #24	buyCPC function can be called even if all tokens have been sold
Severity	LOW SEVERITY
Description	As tokens are purchased in the presale, the total number of tokens purchased is not recorded. As such, the `buyCPC` function can be called even after all tokens have been sold but before `endTimestamp` has been reached, resulting in transactions that revert as `token.safeTransfer(_msgSender(), rec);` throws an exception. This would result in wasted gas for users.

Recommendation	Consider adding a require statement in the `buyCPC` function that prevents it being called if there are no tokens remaining :
	`require(token.balanceOf(address(this)) > 0);`
	Recording the amount of tokens sold would also be desirable, as such :
	`tokensRemaining -= rec;`
	Consider adding in a refund mechanism should there be insufficient tokens in a purchase transaction. For example, if there were only 25 tokens remaining but a user has sent a purchase amount for 100 tokens, then `buyAmount[msg.sender]` records the entire 100 tokens as purchased but is only able to transfer 25 tokens.

Resolution	ACKNOWLEDGED The client has stated that this contract is to be used only once. This issue shall be marked as resolved after the presale is finished and ownership has been renounced.

Issue #25	buyCPC function sends referral tokens to the owner function in certain cases
Severity	LOW SEVERITY
Description	In the following code snippet, if the referrer address is the zero address, then referral rewards are transferred to the owner address. `if (_ref != address(0)) { token.safeTransfer(_ref, refRec); refContract.increaseRefCpc(msg.sender, _ref, refRec); } else { token.safeTransfer(owner(), refRec);` These tokens may then be dumped on the market.
Recommendation	Consider sending them to the burn address instead to ensure that token dumping is no longer possible.
Resolution	ACKNOWLEDGED The client has stated that this contract is to be used only once. This issue shall be marked as resolved after the presale is finished and ownership has been renounced.

Issue #26	**maxBnb_ can cause buyCPC and calculateReceive functions to fail if initialized with too low value, nor does it have a maximum safeguard**
Severity	LOW SEVERITY
Description	In the `calculateReceive` function, there is a minimum requirement of 0.3 BNB for purchase amounts. However, in the constructor, `maxBnb_` can be initialized to be any value including a value below minimum, which may become an issue if `maxBnb` is below 0.3. Should that happen, then any user sending in any amount below 0.3 BNB or above `maxBnb` will cause the `calculateReceive` function to fail. Additionally, since there is no maximum safeguard, a well-funded may purchase all tokens and thus monopolize the circulating token supply.
Recommendation	`maxBnb_` can cause `buyCPC` and `calculateReceive` functions to fail if initialized with too low value, nor does it have a maximum safeguard.
Resolution	ACKNOWLEDGED The client has stated that this contract is to be used only once.

Issue #27	Sending too little BNB will cause `calculateReceive` to fail
Severity	INFORMATIONAL
Description	In the `buyCPC` function, there is only the requirement that `amount_` be a positive integer, though in calculating how many tokens will be purchased in `calculateReceive`, there is a minimum of 0.3 BNB that is hard-coded. Should a user send in less than 0.3 BNB, the `calculateReceive` function will fail. This is currently marked as Informational because the `buyCPC` function cannot be called by contracts, and is very likely that the UI will specify the 0.3 BNB minimum. It is only an issue if the function is called via direct contract interaction.
Recommendation	Consider moving this requirement to the top of `buyCPC` to reduce gas wasted and more important making this check immediately clear to third party reviewers.
Resolution	ACKNOWLEDGED The client has stated that this contract is to be used only once. This issue shall be marked as resolved after the presale is finished and ownership has been renounced.

Issue #28	Lack of constructor parameter validation
Severity	INFORMATIONAL
Description	Currently the constructor allows for `endTimestamp` to be set before the `startTimestamp`. It is always recommended to add some basic constructor validation to prevent mistakes like this from happening and being missed.
Recommendation	Consider adding some basic validation on the `require(endTimestamp_ > startTimestamp_);`
Resolution	ACKNOWLEDGED The client has stated that this contract is to be used only once. This issue shall be marked as resolved after the presale is finished and ownership has been renounced.

Issue #29	`token_`, `startTimestamp_`, `endTimestamp_`, `tokenPerLP_`, `maxBnb_` and `refContract_` 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.
Recommendation	Consider making the above variables `immutable`.
Resolution	ACKNOWLEDGED The client has stated that this contract is to be used only once. This issue shall be marked as resolved after the presale is finished and ownership has been renounced.

Issue #30	Inconsistent notation between parameters
Severity	INFORMATIONAL
Description	Certain parameters have been notarized as `token_` and `maxBnb_`, whilst others were written as `_ref` and `_rateLimit`. This can cause some confusion to readers.
Recommendation	Consider standardizing the parameter notations, ideally as `_token` and `_maxBnb` to facilitate easier reading by future reviewers.
Resolution	ACKNOWLEDGED The client has stated that this contract is to be used only once. This issue shall be marked as resolved after the presale is finished and ownership has been renounced.

Issue #31	Inconsistent indentation in buyCPC function
Severity	INFORMATIONAL
Description	Readability of the contracts could be greatly improved if the indentations were consistent.
Recommendation	Consider making the indentations more consistent in the `buyCPC` function to facilitate easier reading by future reviewers.
Resolution	ACKNOWLEDGED The client has stated that this contract is to be used only once. This issue shall be marked as resolved after the presale is finished and ownership has been renounced.

2.7 CopyCatSmartDeposit

The CopyCatSmartDeposit contract is a helper contract that allows zapping into a leader with any token and through any router.

2.7.1 Issues & Recommendations

Issue #32	**An exploiter can give a router approval to withdraw tokens at any time**
Severity	HIGH SEVERITY
Description	Even though the router is approved, it does not actually have to take the funds out if it is malicious. An exploiter can create a deposit but not actually swap it out (with a malicious router) and combine it with the next issue to retrieve the deposit again.
Recommendation	Consider the implications of this approval, at the very least we recommend to include `reentrancyGuards` to limit potential issues. We also recommend resetting the approval after the transaction is finished.
Resolution	RESOLVED Both reentrancy guards and approval resetting has been implemented.

Issue #33	**Any token or BNB value can be taken out of the contract due to a reentrancy exploit**
Severity	MEDIUM SEVERITY
Description	Since the user can define the swap path themselves, they can include malicious tokens that allow them to reenter in any part of the contract or even transfer in more tokens. This allows an exploiter to reenter into the deposit or withdrawal function and inflate their balances due to the before-after methodology used. An exploiter can use this method to drain the CopyCatSmartDeposit contract of any token present in it. This issue is marked as just medium risk as this appears to be a utility contract.
Recommendation	Consider adding `reentrancyGuards` to the `deposit` and `withdraw` functions.
Resolution	RESOLVED Reentrancy guards are added to both `deposit` and `withdraw`.

Issue #34	**Phishing: User could be mislead in approving a dangerous deposit or withdraw transaction**
Severity	INFORMATIONAL
Description	Since the parameters allow for such great freedom, a malicious frontend could show an innocent looking transaction to the user which actually changes one of the parameters like the router to a router which simply takes all funds sent to it.
Recommendation	Consider carefully securing the frontend against any possible hijacks.
Resolution	ACKNOWLEDGED

2.8 CopyCatEmergency

The CopyCatEmergency contract executes a transaction like a Timelock using `isAllowEmergency[txHash] = true;` with a 14-day grace period.

2.8.1 Issues & Recommendations

Issue #35	**Anyone can execute and re-execute a transaction**
Severity	HIGH SEVERITY
Description	Although a transaction has to be allowed first (eg. through the timelock queueing in CopyCatEmergencyMaster), once it is allowed, anyone can call `executeTransaction` to actually execute the transaction. This issue would be considered as medium severity but since anyone can actually re-execute the transaction as many times as they want, it is elevated to high severity since this will likely have many side-effects.
Recommendation	Consider adding a desired executor to the `txHash`.
Resolution	RESOLVED An `allowEmergencyCall` hook is now called on the contracts that implement the Emergency contract – these hooks require the message sender to be the contract owner and thus prevent this issue from occurring.

2.9 CopyCatEmergencyMaster

The CopyCatEmergencyMaster is a generic contract for the emergency functionality within contracts like the leader. The emergency functionality allows the client to execute any code from the various contracts in case of emergencies. The `emergencyMaster` provides the owner with abilities to restrict this functionality further.

2.9.1 Issues & Recommendations

Issue #36	**If a timelock allower is added, it can still be circumvented by simply introducing a new allower**
Severity	INFORMATIONAL
Description	There are no special safeguards on adding new allowers, an allower can be used to prevent emergency requests so an obvious allower would be a multisig or timelock allower. However, adding new allowers can be done simply by the owner of the EmergencyMaster contract, circumventing much of this protection if this owner is just an EOA.
Recommendation	Consider having an existing allower approve any `allowEmergency` calls. This way, adding new allowers has to be approved by a previous one.
Resolution	ACKNOWLEDGED

2.10 CopyCatReserver

The CopyCatReserver is a simple CopyCat token holding contract without locking functionality.

2.10.1 Issues & Recommendations

Issue #37	There is no locking functionality in the CopyCatReserver
Severity	INFORMATIONAL

Location	Lines 15-17
	`function transfer(address to, uint256 amount) external onlyOwner returns(bool) {`
	`  return cpc.transfer(to, amount);`
	`}`

Description	In case the goal of the CopycatReserver is to lock in funds as a vesting contract, this contract requires this logic to be implemented on a higher level like a timelock contract. The current implementation of CopyCatReserver allows the contract owner to take out funds at any time.
Recommendation	If this contract is meant for locking in funds, consider adding this functionality in explicitly.

```
contract CopycatReserver is Ownable {
 IERC20 public cpc;
 uint256 public immutable vestTimestamp;
  constructor(IERC20 _cpc, uint256 _vestTimestamp) {
    cpc = _cpc;
    vestTimestamp = _vestTimestamp;
 }

 function transfer(address to, uint256 amount) external onlyOwner returns(bool) {
    require(block.timestamp >= vestTimestamp, "Not vested yet");
    return cpc.transfer(to, amount);
 }
}
```

Resolution	RESOLVED The client has indicated that this is by design.

Issue #38	No event on `transfer`
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation	Add events for `transfer`.
Resolution	RESOLVED

Issue #39	`cpc` 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.
Recommendation	Consider making `cpc` explicitly `immutable`.
Resolution	RESOLVED

2.11 CopyCatLeader

A CopyCatLeader represents a vault with a collection of strategies (adapters). The Leader owner can rebalance between the strategies freely and other investors can deposit and withdraw from the vault.

2.11.1 Issues & Recommendations

Issue #40	**Price manipulation exploit allows leader owner to profitably drain the leader**
Severity	HIGH SEVERITY
Description	Since the leader owners will likely be anonymous and untrusted, they should probably not have the ability to drain the tokens of their leader. However, due to the `toAdapter` and `toLeader` functions, they can be drained through price manipulation: Within a single transaction: 1. Pick the pair with the lowest liquidity of the leader. 2. Purchase tokens approximately equivalent to the leaders value in that pair. 3. Use `toLeader` and `toAdapter` to make the leader use its whole balance to purchase these tokens. 4. Sell tokens again at a much higher price.
Recommendation	In case leader owners are trusted and should be allowed to drain all tokens, this issue will be marked as resolved; otherwise the protocol will have to be carefully inspected for all potential ways the leader owners can drain or burn the value of their leader. Potentially the system could be rethought to include some form of reputation or to be mainly used by smart contract owners that define their trading behavior in the code.
Resolution	RESOLVED The Leaders will undergo a Know-Your-Customer (KYC) process where their identities will be disclosed to the protocol, which should disincenitivse malicious actions by the Leaders.

Issue #41	**Withdrawals wrongly affect share value which can lead to exploitation and losses to other stakers**
Severity	HIGH SEVERITY
Description	The client has not provided proof that the way share values are calculated are safe from price manipulation or other ways of exploitation. Thus it was up to us to try to find a way to formally prove that under the current code, deposits and withdrawals are done correctly and soundly. One of the first obvious requirements is that ceteris paribus and under no friction (e.g. fees), a deposit followed by a withdrawal of all the received shares should result in no profit or loss. Sadly enough, this fundamental property does not hold.

Example

INITIAL STATE

We demonstrate a scenario where the current pricing mechanism is insufficient. Assume a leader with a single adapter holding BNB, and the following properties of the leader and the associated pair. Note that the BNB value incorporates the slippage and there is an initial shareholder depositor 1.

- LP reserves: [1 DEMO, 1 BNB]
- Leader balance: 1 DEMO
- BNB value: 0.5 BNB
- Shares: 0.5 shares [depositor 1: 0.5, depositor 2: 0]

Share value: 1 BNB/share

DEPOSITOR 2 JOINS

Assume that depositor 2 deposits 1 BNB in the leader.

- LP reserves: [0.5 DEMO, 2 BNB]
- Leader balance: 1.5 DEMO
- BNB value: 1.5 BNB
- Shares: 1.5 shares [depositor 1: 0.5, depositor 2: 1]

Share value: 1 BNB/share

CASE A - DEPOSITOR 1 WITHDRAWS

- LP reserves: [1 DEMO, 1 BNB]
- Leader balance: 1 DEMO
- BNB value: 0.5 BNB
- Shares: 1 [depositor 1: 0, depositor 2: 1]

Share value: 0.5 BNB/share
The property is violated.

CASE B - DEPOSITOR 2 WITHDRAWS

- LP reserves: [1.5 DEMO, 2/3 BNB]
- Leader balance: 0.5 DEMO
- BNB value: 1/6 BNB
- Shares: 0.5 [depositor 1: 0.5, depositor 2: 0]

Share value: 1/3 BNB/share
The property is violated

Recommendation	A fundamentally sound resolution involves revamping the withdrawal mechanism. Instead of withdrawing a percentage of tokens to the user, withdraw the percentage of BNB value so that at the end, the BNB value has decreased correctly. Additionally, we recommend that the client put forth formal validation rules for which adapters should adhere to, with particular emphasis on the deposit and withdrawal functions, to ensure that these adapters reduce or eliminate the risk of profitable and unprofitable exploitation. These may include but are not limited to: • The deposit and withdrawal functions should not affect share values. • Possible price manipulation of share values through other means. Alternatively, the client may consider acknowledging the issue, though this may leave the project vulnerable to a plethora of potentially high-severity side effects from not solving this issue.
Resolution	PARTIALLY RESOLVED The `withdraw` function now incorporates `shareRatio`, so this specific issue should no longer be possible. Once the formal set of validation rules has been dictated and the Round Table discussions adequately address this issue, we may mark the issue as Resolved then.

Post-resolution Example

Scenario

- LP reserves: [1 DEMO, 1 BNB]
- Leader balance: 1 DEMO
- BNB value: 0.5 BNB
- Shares: 0.5 shares [depositor 1: 0.5, depositor 2: 0]

Share value: 1 BNB/share

DEPOSITOR 2 JOINS

Assume that depositor 2 deposits 1 BNB in the leader.

- LP reserves: [0.5 DEMO, 2 BNB]
- Leader balance: 1.5 DEMO
- BNB value: 1.5 BNB
- Shares: 1.5 shares [depositor 1: 0.5, depositor 2: 1]

Share value: 1 BNB/share

CASE A - DEPOSITOR 1 WITHDRAWS

- LP reserves: [4/6 DEMO, 1.5 BNB]
- Leader balance: 8/6 DEMO
- BNB value: 1 BNB
- Shares: 1 [depositor 1: 0, depositor 2: 1]

Share value: 1 BNB/share.

The property is satisfied.

CASE B - DEPOSITOR 2 WITHDRAWS

- LP reserves: [1 DEMO, 1 BNB]
- Leader balance1 DEMO
- BNB value: 0.5 BNB
- Shares: 0.5 [depositor 1: 0.5, depositor 2: 0]

Share value: 1 BNB/share.

The property is satisfied.

We recommend the project to carefully redo these tests given the addition of fees. Since there is a fee on withdrawals before the actual swap is made, the vault value might even slightly increase on withdrawal sine these tokens do no longer create slippage.

Issue #42	**Share values may not be strictly safe from various possible methods of exploitations**
Severity	MEDIUM SEVERITY
Description	As this is a highly complex protocol with customised contracts, we cannot say with a reasonable degree of certainty that the current share price calculation is free from profitable or unprofitable exploitation. Due to the rushed nature of this audit, we have made our best efforts to identify possible risk vectors, though there may yet remain some avenues for which malicious actors may attempt to manipulate or exploit the protocol.
Recommendation	We strongly urge the CopyCat team to rigorously test their protocol, especially on edge cases, to identify and patch any possible loopholes that may remain, as well as ensure that the business logic of the protocol works as intended. Additionally, we recommend that the Copycat team set formal rules and requirements that clearly specifies the intended functionality of the protocol, which can then be methodically and extensively tested prior to live deployment. This issue will be marked as Resolved once we have confirmed from the team that they have undergone sufficient extensive testing, and that proper documentation and testing procedures of the protocol's intended purposes, functionalities and edge cases have been duly noted.
Resolution	ACKNOWLEDGED

Issue #43	**Malicious contracts are currently unrestricted from executing exploits, if any**
Severity	MEDIUM SEVERITY
Description	Should there be vectors for exploitation, malicious contracts have unrestricted access to perform attacks on the protocol. Functions including `deposit` and `withdrawal`, amongst others, may be more prone to these vectors. Should there be a restriction that these external functions may only be called by EOA, then the magnitude and severity of possible exploits may be less incentivised, or made more difficult to execute.

Recommendation	Create and include an onlyEOA modifier to all external functions in the contract, which may be enabled or disabled should there be a need to include certain contract interactions.
	
	An example modifier is provided below, for reference purposes :
	`modifier onlyEOA(){`
	`  bool isEOA = tx.origin == msg.sender && ! msg.sender.isContract();`
	`  require(whitelisted[msg.sender] || eoaDisabled || isEOA, "You must be whitelisted or an EOA.");`
	`  _;`
	`}`

Resolution	RESOLVED The `deposit`, `withdraw`, `toLeader` and `toAdapter` functions are now only callable by user wallets (EOAs).

Issue #44	Leader owner can burn the whole leader balance at any time
Severity	MEDIUM SEVERITY
Description	Since there is no rate limit on `toLeader` and `toAdapter`, it can be called in great succession by a leader to burn the whole balance through the transfer fees associated with this.
Recommendation	Potentially the system could be rethought to include some form of reputation or to be mainly used by smart contract owners that define their trading behavior in the code.
Resolution	RESOLVED The Leaders will undergo a Know-Your-Customer (KYC) process whereby their identities will be disclosed to the protocol, which should disincentivize malicious actions by the Leaders.

Issue #45	**Deposit malfunctions (eg. by price manipulation) could give the user more BNB value then they deposited**
Severity	LOW SEVERITY
Description	The `depositTo` function allows the user's share to increase by a greater amount then the BNB sent to the contract. This is likely undesirable and might be due to a miscalculation or abusing the function.

Recommendation	Within deposit, consider adding
	`if(msg.value < totalReceived) {`
	`  totalReceived = msg.value;`
	`}`

Resolution	RESOLVED The recommendation has been added.

Issue #46	**Function returns multiple distinct values in an array which misleads reviewers**
Severity	INFORMATIONAL
Description	Within the `getAdaptersBnb` function, a large array is returned with the two last values being very different from the rest. The last value is the total BNB in the adapters + leader and the previous to last is the BNB in the leader.
Recommendation	Consider polishing the information that is returned, so as to be presented in a more reader-friendly manner should there be a need to retain them.
Resolution	RESOLVED The client has stated that this is intended for optimisation purposes.

Issue #47	`getAdapters` can run out of memory
Severity	INFORMATIONAL
Description	As the adapters grow in size, they may exceed the maximum number of bytes a call can return making the `getAdapters` function revert on every call.
Recommendation	Consider whether the `getAdapters` function is important and consider adding a maximum number of adapters.
Resolution	RESOLVED The client has stated that they will not add in too many adapters.

Issue #48	A large amount of comments can be deleted
Severity	INFORMATIONAL
Description	A large part of the lines inside this contract are commented out code which is not relevant to the behavior of the contract. This makes it more difficult for third-party reviewers to get a quick idea what the contract does. Some comments also indicate that the share ratio is denominated in 1e9 while it is actually in 1e18.
Recommendation	Consider removing the obsolete comments.
Resolution	RESOLVED The client has stated that they wish to retain them for future use.

2.12 CopyCatLeaderFactory

The CopyCatLeader factory allows anyone to create a new leader. The leader receives 40% of the leader fees, the remaining goes to the owner. During the creation an initial mandatory deposit of 0.3 BNB is made by the owner. Deposit and withdrawal fees can be scaled by the governance.

2.12.1 Issues & Recommendations

Issue #49	**Migration can be done by anyone resulting in anyone being able to steal all funds from another Leader**
Severity	HIGH SEVERITY
Description	Currently the `migrateFrom` parameter can be set to any address which allows anyone to migrate funds to their newly created leader in `deployLeader`. We are unsure about the value of being able to migrate leaders to a new leader in the first place.
Recommendation	Consider removing the migration functionality completely if it has no specific reason as this adds a lot of complexity in the protocol which is hard to assert safety over.
Resolution	RESOLVED The migration functionality has been commented out, though this can be removed entirely since they no longer have any functionality.

Issue #50	**Reentrancy exploit allows for two leaders to exist under the same `leaderId`**
Severity	HIGH SEVERITY

Description	Currently a `leaderId` can be added twice through reentrancy. This is because the `nextLeaderId` is only incremented at the end of the `deployLeader` function. Note that two contracts with the same salt cannot be deployed but since `msg.sender` can be adjusted in the reentrancy, the salt can be changed as well.
	
	`bytes32 salt = keccak256(abi.encodePacked(msg.sender, nextLeaderId));`
	
	Note that reentrancy is trivial through the `migrateTo` function, which can be executed on any contract provided by the user.
	
	`if (_migrateFrom != address(0)) { CopycatLeader(_migrateFrom) .migrateTo(CopycatLeader(copycatLeaderAddress)); }`

Recommendation	Consider adding reentrancy guards to all functions to prevent reentrancy and remove any way for exploiters to execute malicious code like this migrateTo function. Consider using the checks-effects-interactions pattern to prevent reentrancy.
Resolution	RESOLVED The `deployLeader` function now includes `reentrancyGuard`.

Issue #51	**Malicious code injection exploit allows to execute code before the leader is initialized allowing an exploiter to add malicious adapters and pre-mint to their leader**
Severity	HIGH SEVERITY
Description	The `migrateTo` function is called after the leader is created but before it is initialized, as described in the previous issue this function allows any code to be injected by a malicious party. Since the leader is not yet initialized, this code allows the malicious party to for example add malicious adapters and pre-mint a huge amount of shares.
Recommendation	Consider removing the migration functionality.
Resolution	RESOLVED The migration functionality has been commented out, though this can be removed entirely since they no longer have any functionality.

Issue #52	**No validation done on deposit and withdraw fee could allow governance to lock in all funds**
Severity	MEDIUM SEVERITY
Description	Currently there is no validation done on the deposit and withdrawal fee.
Recommendation	Consider adding validation checks as to keep the deposit and withdrawal fees within bounds.
Resolution	RESOLVED Deposit fees are calculated from `FEE_LIST`. Withdrawal fees have been removed. Note that the `setFee` function allows the Owner to modify those fee amounts in the `FEE_LIST`, which have updated figures.

Issue #53	**Setting the `feeAddress` to zero can resulting in blocking all functionality**
Severity	MEDIUM SEVERITY
Description	Transfers to a zero address revert, and since there is no validation on the `setFeeAddress` function the governance could set the `feeAddress` to zero and break all withdraws and deposits.
Recommendation	Consider adding a non-zero requirement. `require(_feeAddress != 0);`
Resolution	ACKNOWLEDGED The client has stated that they will not set `feeAddress` to the zero address.

Issue #54	**Governance privilege: Governance could add malicious adapters**
Severity	MEDIUM SEVERITY
Description	The governance is able to add malicious adapters (refer to Issue #49).
Recommendation	Consider putting the governance behind a sufficiently long timelock.
Resolution	ACKNOWLEDGED The client will put Governance behind a sufficiently long timelock after launch. Once this has happened, we will mark this issue as Resolved.

Issue #55	**No validation done on `setFee` allows governance to update many fee parameters to severe levels**
Severity	LOW SEVERITY
Description	Currently there is no validation done on the `setFee` function which allows governance to update the fees. This allows them to set the fees to absurd levels.
Recommendation	Consider adding validation checks to the `newFee` and index parameters as to keep the `setFee` function within bounds.
Resolution	ACKNOWLEDGED The client has stated that they will not set fees above 20%.

2.13 CopyCatAdapterFactoryBase

2.13.1 Issues & Recommendations

Issue #56	**Governance privilege: Owner can execute any function on the adapter after at any time**
Severity	HIGH SEVERITY
Description	All `adapterFactories` that extend the base allow the adapter factory owner to execute code at any time. This issue is marked as high risk for now since our audits are investor safety oriented. This issue will be moved to an informational section once this audit is moved to the "Paladin Round Table" label, which is simply a disclaimer that we have audited the code with governance trust in mind.
Recommendation	Consider putting the factory behind a timelock and clearly indicating this to the users.
Resolution	PARTIALLY RESOLVED The client has stated that they will transfer ownership of the contract to a sufficiently long Timelock. Once that has occurred, we will mark this issue as Resolved.

2.14 CopycatUniswapV2Adapter

The CopycatUniswapV2 adapters are strategies that simply hold a specific token and will use a Uniswap router to exchange it to BNB when a withdrawal is made.

2.14.1 Issues & Recommendations

Issue #57	Routes that allow reentrancy could allow for balance inflation
Severity	LOW SEVERITY
Description	In case a route allows for reentrancy, a deposit can be made twice and the wrapper deposit return amount can be inflated.
Recommendation	Consider adding reentrancy guards to all relevant functions (`toAdapter`, `toLeader`).
Resolution	RESOLVED The `toAdapter`, `toLeader` and `toggleZeroMode` functions have reentrancy guards.

2.15 CopycatUniswapV2AdapterFactory

The CopycatUniswapV2 adapter allows leaders to include adapter strategies too hold a specific currency. Only whitelisted currencies (pairs) can be used. If ever a pair is added that allows for reentrancy, this will highly complicate the code further as it will allow potential reentrancy vectors throughout the system.

2.15.1 Issues & Recommendations

Issue #58	**Governance privilege: Governance can set fee rate up to 100% resulting in all swaps going to the governance**
Severity	HIGH SEVERITY
Description	The swap fee rate that goes to the governance is initially 1%, however, governance can adjust it up to any value
Recommendation	Consider fundamentally rethinking the functionality of CopyCat. Due to the current setup, adapters will always be prone to price manipulation exploits. Some ways to decrease the profitability of this is having a large enough minimal deposit fee.
Resolution	RESOLVED The `feeRate` now has an upper limit of 1%.

Issue #59	**Lack of validation on trading routes**
Severity	INFORMATIONAL
Description	There is currently no validation on the trading routes. This means if they start or end with the wrong token by accident, the route can still be added.
Recommendation	Consider validating the start and ends of the trading routes
Resolution	RESOLVED

2.16 Timelock

The Timelock contract is a clean fork of Compounder Finance's timelock. This is the most common contract used in DeFi to time lock governance access and is thus compatible with most third-party tools.

Parameter	Value	Description
Delay	1 day	The `delay` indicates the time the administrator has to wait after queuing a transaction to execute it.
Minimum Delay	1 day	The `minDelay` indicates the lowest value that the `delay` can minimally be set. Sometimes, projects will queue a transaction that sets the `delay` to zero with the hope that nobody notices it. However, because of the minimum delay parameter, the value of `delay` can never be lower than that of the `minDelay` value. Note that the administrator could still queue a transaction to simply transfer the ownership back to their own account so it is still important to inspect every transaction carefully.
Grace Period	14 days	After the delay has expired after queueing a transaction, the administrator can only execute it within the grace period. This is to prevent them from hiding a malicious transaction among much earlier transactions, hoping that it goes unnoticed or buried, which can be executed in the future.

*This table is based on communication with the developer and is still pending on-chain verification.

2.16.1 Issues & Recommendations

Issue #60	Gas efficiency: Usage of inefficient SafeMath version
Severity	INFORMATIONAL

Location	Lines 19-186
	`library SafeMath {`
	…
	`}`

Description	Within older versions of SafeMath, errors are passed on as a parameter. This behavior leads to unnecessary memory allocation and was fixed in a later update of the OpenZeppelin SafeMath library.
Recommendation	Consider moving to a later version of SafeMath like version 3.4.
Resolution	RESOLVED None of the inefficient functions are used.

2.17 Round Table

The issues listed below are governance-related issues which we would normally mark as high severity; however, as the project team has communicated to us that they require these escalated privileges to operate the protocol, we have listed them here for reference.

2.17.1 Issues & Recommendations

Issue #61	**Governance privilege: Governance can execute any function on all contracts**
Location	CopyCatEmergency
Description	All relevant contracts extend `CopyCatEmergency` which is a special governance contract that allows for governance. This issue is marked as high risk for now since our audits are investor safety oriented. This issue will be moved to an informational section once this audit is moved to the "Paladin Round Table" label, which is simply a disclaimer that we have audited the code with governance trust in mind.

Issue #62	**Governance risk: Any registered adapter factory can add adapters to any leader allowing governance to drain all leaders**
Location	CopyCatLeader
Description	Governance can drain any leader by adding a new adapter factory which automatically can call addAdapter on any leader.


PALADIN
BLOCKCHAIN SECURITY