PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For Hexagon Finance

29 April 2022

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.

Paladin retains full rights over all intellectual property (including expertise and new attack or exploit vectors) discovered during the audit process. Paladin is therefore allowed and expected to re-use this knowledge in subsequent audits and to inform existing projects that may have similar vulnerabilities. Paladin may, at its discretion, claim bug bounties from third-parties while doing so.

1 Overview

This report has been prepared for Hexagon Finance 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	Hexagon Finance
URL	https://www.hexagonfinance.io/
Network	Avalanche
Language	Solidity

1.2 Contracts Assessed

Name	Contract	Live Code Match
MiniChefV2	MiniChefV2.sol	N/A
lpGauge	lpGauge.sol	N/A
HexagonBoost	HexagonBoost.sol	N/A
hexagonBoostStorage	hexagonBoostStorage.sol	N/A
boostMultiRewarderTime	boostMultiRewarderTime.sol	N/A
VeFlake	VeFlake.sol	N/A
FlakeToken	FlakeToken.sol	N/A
Resolution	https://github.com/HexagonFinance/hexagon-finance-farm/commit/2df6b2c8a6538ebae4df76865805dae006cb0083	

Note: The team did not get back to us with the deployed addresses for the other contracts, so users should check that the code of the contract they are interacting with matches the code in the resolution GitHub commit.

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	7	5	1	1
Medium	7	5	-	2
Low	7	4	-	3
Informational	34	25	6	3
Total	**55**	**39**	**7**	**9**

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 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 MiniChefV2

ID	Severity	Summary	Status
01	HIGH	Governance Privileges: Governance has crucial privileges over the contract which allows for example the governance theft of all staked tokens	PARTIAL
02	HIGH	`deposit` and `boostDeposit` do not support fee-on-transfer tokens	✓ RESOLVED
03	MEDIUM	The rewarder is called with a zero `oldAmount` on `emergencyWithdraw`	✓ RESOLVED
04	MEDIUM	`rewardDebt` is vulnerable to rounding drift	ACKNOWLEDGED
05	LOW	`setFlakePerSecond` has no maximum safeguard	ACKNOWLEDGED
06	LOW	The pending reward function will revert if `totalAllocPoint` is zero	✓ RESOLVED
07	INFO	Various functions should call `massUpdatePools` before adjusting emissions	PARTIAL
08	INFO	Typographical errors	✓ RESOLVED
09	INFO	Inconsistent usage of SafeMath functions	✓ RESOLVED
10	INFO	Unused imports	✓ RESOLVED
11	INFO	Unnecessary marking of `allocPoint` to `uint256` and casting to `uint64`	ACKNOWLEDGED
12	INFO	Several functions can be made external	✓ RESOLVED

1.3.2 IpGauge

ID	Severity	Summary	Status
13	LOW	Governance Privilege: Owner can burn other users' tokens	✓ RESOLVED
14	LOW	`mint` function can be used to pre-mint large amounts of tokens before ownership is transferred to MiniChefV2	✓ RESOLVED
15	INFO	Ambiguous errors and inconsistent non-zero check	PARTIAL
16	INFO	Unused variables: `nonces`	✓ RESOLVED
17	INFO	Variables can be cached within transferFrom	ACKNOWLEDGED
18	INFO	pid can be made immutable	✓ RESOLVED
19	INFO	The `AddAuthorization` and `RemoveAuthorization` events are unused	✓ RESOLVED

1.3.3 HexagonBoost

ID	Severity	Summary	Status
20	HIGH	Governance Privilege: setMulsigAndFarmChef can be used to change the farmChef address to an EOA or a malicious contract which could cause the theft of users their `boostedTokens`	✓ RESOLVED
21	MEDIUM	`minBoostAmount` can only be 0 or 500 ether	✓ RESOLVED
22	MEDIUM	Various variables will affect user rewards in hindsight if ever changed by governance	ACKNOWLEDGED
23	INFO	`getTeamRatio`, `getTotalBoostedAmount`, `boostStakedFor` and boostTotalStaked can be made `external`	✓ RESOLVED
24	INFO	Unused imports and approval	✓ RESOLVED
25	INFO	Contract should inherit `IBoost`	ACKNOWLEDGED
26	INFO	Lack of events on privileged functions	✓ RESOLVED

1.3.4 hexagonBoostStorage

ID	Severity	Summary	Status
27	INFO	Typographical errors	PARTIAL
28	INFO	Unused import: SmallNumbers.sol	✓ RESOLVED

1.3.5 boostMultiRewarderTime

ID	Severity	Summary	Status
29	HIGH	The `pending` variable is perpetually boosted and rewards excessive `rewardToken`	✓ RESOLVED
30	LOW	Governance Privilege: The function `reclaimTokens` allow the contract owner to withdraw any reward tokens to any address	ACKNOWLEDGED
31	LOW	`setRewardPerSecond` has no maximum safeguard	ACKNOWLEDGED
32	INFO	Typographical errors	✓ RESOLVED
33	INFO	`chefPid` and `masterLpToken` can be made immutable	PARTIAL
34	INFO	Inconsistent usage of SafeMath functions	✓ RESOLVED

1.3.6 VeFlake

ID	Severity	Summary	Status
35	HIGH	Users might not be able to leave the vesting contract due to the large memory caching of arrays and iterations	ACKNOWLEDGED
36	HIGH	`cancelLeave` transfers from `msg.sender` to `msg.sender`	✓ RESOLVED
37	HIGH	Governance Privilege: `setFlake` can be set again and prevent users from depositing or withdrawing from the contract	✓ RESOLVED
38	MEDIUM	`getLeaveApplyHistory` does not work as intended	✓ RESOLVED
39	MEDIUM	Governance Privilege: `setLeavingTerm` can be set to an arbitrarily high value by the governance	✓ RESOLVED
40	MEDIUM	Unsafe casts are made throughout the contract that can cause issues	✓ RESOLVED
41	LOW	The `searchPendingIndex` function might not pick the last index if that value occurs multiple times	✓ RESOLVED
42	INFO	The first user could steal the tokens deposited by the next users by manipulating the share value	ACKNOWLEDGED
43	INFO	Inconsistent usage of SafeMath functions	PARTIAL
44	INFO	`enter` function does not support fee-on-transfer tokens	✓ RESOLVED
45	INFO	The `Enter`, `Leave`, `ApplyLeave`, and `CancelLeave` events are indexing amounts which are unnecessary	✓ RESOLVED
46	INFO	Unnecessary casting of `tokenDecimal` to `uint8`	✓ RESOLVED
47	INFO	Several functions can be made external	✓ RESOLVED
48	INFO	Lack of `safeTransfer` usage within `enter` and `leave`	✓ RESOLVED
49	INFO	Typographical errors	PARTIAL
50	INFO	Lack of events for function `setFlake`, `setLeavingTerm`, `setMulsig`	✓ RESOLVED
51	INFO	`_amount` and `_share` should be required to be greater than zero in the various external functions	✓ RESOLVED

1.3.7 FlakeToken

ID	Severity	Summary	Status
52	INFO	Inconsistent usage of `uint256`	✓ RESOLVED
53	INFO	Unnecessary typing of `tokenDecimal` to `uint256` while casting it to `uint8`	✓ RESOLVED
54	INFO	Token amounts can be made more readable by separating the digits into thousands	✓ RESOLVED
55	INFO	`name`, `symbol` and `decimals` can be made external	✓ RESOLVED

2 Findings

2.1 MiniChefV2

MiniChefV2 is a staking contract where users can stake tokens to earn FlakeToken. It can also distribute additional reward currencies if this is enabled by the governance.

The contract contains a series of pools that accepts different tokens and has an allocation point as to how much FlakeToken is distributed to each pool as the reward. Individual users are rewarded based on the time of deposit until they choose to withdraw their stakes.

The MiniChefV2 utilizes `lpGauge` to account for the total supply and receipt tokens. It uses `HexagonBoost` to allow for the rewards to be boosted by depositing an additional boost token. It finally allows for linking `boostMultiRewarderTime` contracts to pools to distribute multiple rewards. There is no deposit/withdrawal fee to take note of, and the emission rate of the native token has not been capped.

It should be noted that the Hexagon repository contains various boosting contracts but only `boostMultiRewarderTime` is considered within scope.

2.1.1 Privileged Roles

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

- `setMulsigAndRewardToken`
- `add`
- `set`
- `setFlakePerSecond`
- `setBooster`
- `setRoyaltyReciever`
- `setBoostFunctionPara`
- `setBoostFarmFactorPara`
- `setWhiteListMemberStatus`
- `setWhiteList`
- `setFixedWhitelistPara`
- `setFixedTeamRatio`
- `transferOwnership`
- `claimOwnership`

2.1.2 Issues & Recommendations

Issue #01	**Governance Privileges: Governance has crucial privileges over the contract which allows for example the governance theft of all staked tokens**
Severity	HIGH SEVERITY
Description	The contract contains several governance privileges: 1. **Governance can replace FLAKE with any token contract through `setMulsigAndRewardToken`.** FLAKE acts as a reward token in this contract and poses no issue to user deposits. The contract governance can change the FLAKE address to a different token. This ability to change the address is a very rare and not recommended practice. This practice may confuse third-party auditors and developers. It can furthermore misinform the user if they suddenly stop receiving reward tokens. More importantly, having a pool of the same token as the reward token will risk user deposits being drained as rewards. For example, If a pool accepts FLAKE token deposits, other pools will be able to drain and distribute the deposited FLAKE as rewards. 2. **`emergencyWithdraw` can be blocked if a malicious rewarder is set** Governance is able to set the rewarder of a certain pool through the `add()` function. In case of incorrect/bug rewarder implementation, `emergencyWithdraw` will always revert. 3. **Governance can replace the booster contract, which stores user boost token deposits** The governance can replace the `booster` variable with any address. A changeable `booster` address can result in user deposits of `boostToken` being stuck in the old booster address until governance rolls the address back to the last booster.

Recommendation	1. Consider not allowing the FLAKE address to be changed if this is not strictly necessary. 2. Consider being forthright with your community if the rewarder contract has been tested, preferably by third-party auditors. 3. Consider making the `setBooster` function callable only once and be forthright with your community to allow them to withdraw their deposits before the booster change.
Resolution	PARTIALLY RESOLVED 1. `setMulsigAndRewardToken` has been removed. 2. Client has stated that the `safeMulsig` will be a multi-signature wallet that involves several parties and that the rewarder contract setting will be controlled and reviewed by the aforementioned parties. In addition, the `emergencyWithdraw` function now has a try-catch statement which handles the exception of a wrong/bugged rewarder implementation. 3. `setBooster` is unchanged.

Issue #02	**`deposit` and `boostDeposit` do not support fee-on-transfer tokens**
Severity	HIGH SEVERITY
Description	Within the `deposit` and `boostDeposit` functions, there is no logic that supports fee-on-transfer tokens. Therefore, during a deposit, the MiniChefV2 will receive less tokens than the user will get credited. This can lead to an exploit where a malicious user can drain the whole pool, which results in absurd reward minting.
Recommendation	Consider adding logic for token with a fee on transfer that checks the balance before deposit and checks it again afterwards to account for any balance difference. Such a mechanism should always be accompanied with reentrancy guards.
Resolution	RESOLVED The client indicated that they will not be using tokens with a fee on transfer.

Issue #03	**The rewarder is called with a zero `oldAmount` on `emergencyWithdraw`**
Severity	MEDIUM SEVERITY
Location	Line 392 `_rewarder.onFlakeReward(pid, msg.sender, to, 0, 0,false);`
Description	The rewarder documentation state that the two amounts should be passed: the old and new amount of tokens the user has. However on `emergencyWithdraw`, this old amount is always zero. This could be useful to avoid as much rewarder logic as possible and maximize the chances of `emergencyWithdraw` working, but it could allow for exploitation if the rewarder does not know about this case.
Recommendation	Consider whether this `oldAmount` should be set to the `amount` variable instead.
Resolution	RESOLVED The client indicated that the rewarder is aware about the `oldAmount` being zero.

Issue #04	`rewardDebt` is vulnerable to rounding drift
Severity	MEDIUM SEVERITY
Location	Line 270 `user.rewardDebt = user.rewardDebt.add(int256(amount.mul(pool.accFlakePerShare) / ACC_FLAKE_PRECISION));`
Description	Solidity is prone to rounding drift issues where rounding errors can be accumulated over time. This issue can be exploited by continuously depositing or withdrawing in small amounts. Moreover, the chance of this getting abused is very common on cheap/gasless chains.
Recommendation	Consider not dividing with `ACC_FLAKE_PRECISION` on every update. Precision should only be stored on-chain but calculated off-chain.
Resolution	ACKNOWLEDGED

Issue #05	**`setFlakePerSecond` has no maximum safeguard**
Severity	LOW SEVERITY
Description	It is common for projects to accidentally update their emission rate to a severely high number either by accident or with malicious intent. By having a maximum value, the code itself enforces the reward rate to always be within a reasonable range, preventing mistakes which cannot be reverted once they are made. This might also boost investor confidence as they know that the governance cannot suddenly set the emission rate to a very high value.
Recommendation	Consider adding a `MAX_EMISSION_RATE` variable and setting it to a reasonable value. `require(_flakePerSecond <= MAX_EMISSION_RATE,"Too high");`
Resolution	ACKNOWLEDGED

Issue #06	**The pending reward function will revert if `totalAllocPoint` is zero**
Severity	LOW SEVERITY
Description	In the pending reward 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 adding a non-zero check to the `if` statement. `if (block.timestamp > pool.lastRewardTime && lpSupply != 0 && totalAllocPoint > 0) {`
Resolution	RESOLVED

Issue #07	Various functions should call `massUpdatePools` before adjusting emissions
Severity	INFORMATIONAL

Location	Line 142
	`function add(uint256 allocPoint, IERC20 _lpToken, IRewarder _rewarder) public onlyOrigin {`
	
	Line 165
	`function set(uint256 _pid, uint256 _allocPoint, IRewarder _rewarder, bool overwrite) public onlyOrigin {`
	
	Line 174
	`function setFlakePerSecond(uint256 _flakePerSecond) public onlyOrigin {`

Description	The `add`, `set` and `setFlakePerSecond` functions all adjust emission distributions. The new values will however affect the rewards in hindsight.
Recommendation	To avoid rewards being adjusted in hindsight, the client should add a `withUpdate` boolean argument that if set to `true`, calls `massUpdatePools`. This call is made optional to ensure that these functions can still be called even when `massUpdatePools` reverts or runs out of gas. `onBalanceChange` should also call an update before changing the balances.
Resolution	PARTIALLY RESOLVED The set function now does an `updatePool` call before proceeding with the pool setting modification. `add` and `setFlakePerSecond` remains unchanged.

Issue #08	Typographical errors
Severity	INFORMATIONAL

Description	Line 36
	`/// Also known as the amount of FLAKE to distribute per block.`
	
	This comment is not in the right location.
	
	Line 44
	`address public royaltyReciever;`
	
	This should be spelled as `royaltyReceiver`.
	
	Line 47
	`modifier onlyOrigin() {`
	
	Consider revising to `onlyMultisig` to prevent confusion.
	
	Line 54
	`// @notice The migrator contract. It has a lot of power. Can only be set through governance (owner).`
	
	Consider removing this comment to prevent misunderstandings with investors. Containing references to a migrator can instantiate fear.
	
	Line 265
	`function depoistPending(PoolInfo memory pool,uint256 pid, uint256 amount, address to)internal`
	
	This should be spelled as `depositPending`.

	Line 101
	`function setMulsigAndRewardToken(address _multiSignature, address _flake) onlyOrigin public {}`
	
	`_flake` can be marked as IERC20.
	
	Line 415
	`//init to defualt vault`
	
	This should be spelled default.

Recommendation	Consider revising the typographical errors.
Resolution	RESOLVED

Issue #09	**Inconsistent usage of SafeMath functions**
Severity	INFORMATIONAL
Description	There are several lines that still use arithmetic operations to compute mathematical equations and this could expose the contract to overflow/underflow risks as there are no internal checks yet for the used solidity version.
Recommendation	Consider using the SafeMath functions to compute instead of the arithmetic operations.
Resolution	RESOLVED

Issue #10	Unused imports
Severity	INFORMATIONAL
Location	`import "@boringcrypto/boring-solidity/contracts/BoringBatchable.sol";` `import "@boringcrypto/boring-solidity/contracts/BoringOwnable.sol";`
Description	Files that are imported in a contract but not used within said contract could confuse third-party auditors. They also increase the contract length and bytecode size unnecessarily.
Recommendation	Consider removing the import to keep the contract short and simple.
Resolution	RESOLVED

Issue #11	Unnecessary marking of `allocPoint` to `uint256` and casting to `uint64`
Severity	INFORMATIONAL

Location	Line 152
	`allocPoint: allocPoint.to64(),`
	
	Line 167
	`poolInfo[_pid].allocPoint = _allocPoint.to64();`

Description	`allocPoint` is marked as `uint256` in the `add` and `set` functions. However, the `allocPoint` variable is a `uint64` which means the input needs to be cast to `uint64`. Casting would not be necessary if the constructor provided the variable as `uint64` instead of `uint256`.
Recommendation	Consider marking the `allocPoint` input as `uint64` to avoid unnecessary casting.
Resolution	ACKNOWLEDGED

Issue #12	Several functions 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: - `setMulsigAndRewardToken` - `poolLength` - `add` - `set` - `setFlakePerSecond` - `deposit` - `withdraw` - `emergencyWithdraw` - `setBooster` - `setRoyaltyReciever` 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 functions as external.
Resolution	RESOLVED

2.2 lpGauge

`lpGauge` is an `ERC20` contract and an extension of the `MiniChefV2` contract for tracking each pool's deposit supply. In the `MiniChefV2` contract, each pool is assigned an `lpGauge` contract that mints `lpGauge` tokens to account for the amount deposited by the user in the `MiniChefV2`, and burns the same whenever there are withdrawals. This contract is deployed whenever there are additional pools in the `MiniChefV2`.

Being an `ERC20` contract, it can also be transferred to other users upon setting the allowance. When a transfer occurs, the `lpGauge` token calls the `onTransfer` function of the `MiniChefV2` contract to also transfer the user deposit to the recipient of the `lpGauge` tokens.

It can also be noted that the `onTransfer` function does not support tokens with a fee on transfer and can transfer more user deposits than expected.

2.2.1 Privileged Roles

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

- `mint`
- `burn`

2.2.2 Issues & Recommendations

Issue #13	Governance Privilege: Owner can burn other users' tokens
Severity	LOW SEVERITY
Location	Line 102 `function burn(address usr, uint256 amount) onlyOwner external {`
Description	Upon contract deployment, the owner of the contract is still the deployer and the `burn` function can be called to burn `lpGauge` tokens of other accounts by using the address input. This can only be resolved when the ownership has been transferred to the MiniChefV2 contract.
Recommendation	Consider being forthright with the community and auditors that ownership has been transferred to the MiniChefV2 contract.
Resolution	RESOLVED The client has indicated that `Owner` will only be `MiniChefV2`.

Issue #14	`mint` function can be used to pre-mint large amounts of tokens before ownership is transferred to MiniChefV2
Severity	LOW SEVERITY
Description	The `mint` function allows the owner (contract deployer) to mint tokens before ownership is transferred to MiniChefV2. This could be used to mint a large amount of tokens and potentially dump them on user generated liquidity 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 the tokens are currently stored, if a vesting contract was used for token unlocking, and finally the purpose of the mints.
Resolution	RESOLVED The client has indicated that `Owner` will only be `MiniChefV2`.

Issue #15	Ambiguous errors and inconsistent non-zero check
Severity	INFORMATIONAL

Description	Lines 74-76
	`require(dst != address(0), "Coin/null-dst");`
	`require(dst != address(this), "Coin/dst-cannot-be-this-contract");`
	`require(balanceOf[src] >= amount, "Coin/insufficient-balance");`
	
	The contract contains code which do not revert with an error message, instead they revert ambiguously, leaving users to potentially wonder what happened with their transaction. It also makes writing coverage tests difficult as these cannot explicitly check for the reversion method. Additionally, the function `transferFrom` did not include a non-zero check for the `src` value.
	
	Lines 102-103
	`function burn(address usr, uint256 amount) onlyOwner external {`
	`    require(balanceOf[usr] >= amount, "Coin/insufficient-balance");`
	
	The `burn` function lacks a non-zero check or a `notZeroAddress` modifier.

Recommendation	Consider adding non-zero checks to the above functions and adding explicit reversion messages to the above locations.
Resolution	PARTIALLY RESOLVED The `burn` function still lacks a non-zero check or a `notZeroAddress` modifier.

Issue #16	Unused variables: nonces
Severity	INFORMATIONAL
Description	Variables defined in a contract but not used within said contract could confuse third-party auditors. They furthermore increase the contract length and bytecode size for no reason.
Recommendation	Consider removing the variable to keep the contract short and simple.
Resolution	RESOLVED

Issue #17	Variables can be cached within `transferFrom`
Severity	INFORMATIONAL
Description	Caching storage data into a variable is a good practice to not only save gas upon transaction but also make the code clearer. Various variables can be cached within `transferFrom`: `allowance[src][msg.sender]` and `balanceOf[src]`.
Recommendation	Consider using the OpenZeppelin reference ERC20 implementation instead. `_transfer` can then be overridden to add the minichef hook in.
Resolution	ACKNOWLEDGED

Issue #18	`pid` 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 variable explicitly immutable.
Resolution	RESOLVED

Issue #19	`AddAuthorization` and `RemoveAuthorization` events are unused
Severity	INFORMATIONAL

Location	Lines 34, 35
	`event AddAuthorization(address account);`
	`event RemoveAuthorization(address account);`

Description	Events which are defined in a contract but remain unused could confuse third-party auditors. They also increase the contract length unnecessarily.
Recommendation	Consider removing the events to keep the contract short and simple.
Resolution	RESOLVED

2.3 HexagonBoost

`HexagonBoost` is a supplementary contract to the `MiniChefV2` and inherits the `HexagonBoostStorage` for calling several variables/mappings. The contract is used as the boost contract for the `MiniChefV2` and allows users to boost their rewards. It mostly contains privileged functions that deal with whitelisting users to receive an additional boost in the rewards, enabling/disabling boosting for a specific pool ID in the `MiniChefV2` contract, and several `view` functions that compute how much the boost amount is for whitelisted users and regular users. This contract also computes the project team's "royalty" fees on the boosted amount if `isTeamRoyalty` has been enabled for a specific pool ID.

To receive a boosted stake, users must deposit boost tokens into the contract through the `boostDeposit` function. The contract stores those tokens until the user calls the `boostWithdraw` function to retrieve their tokens from the contract.

2.3.1 Privileged Roles

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

- `setMulsigAndFarmChef`
- `setFixedTeamRatio`
- `setFixedWhitelistPara`
- `setWhiteListMemberStatus`
- `setBoostFarmFactorPara`
- `setBoostFunctionPara`
- `boostDeposit`
- `boostWithdraw`

2.3.2 Issues & Recommendations

Issue #20	**Governance Privilege: `setMulsigAndFarmChef` can be used to change the `farmChef` address to an EOA or a malicious contract which could cause the theft of users their `boostedTokens`**
Severity	HIGH SEVERITY
Description	The contract owner is able to modify the `farmChef` variable to any address. Modifying this address disconnects the contract from the `MiniChefV2`. The new `farmChef` address can then use any `onlyMCV2` privileged functions. Such a function could be `boostDeposit` which can add balances to the account of the transaction sender. If combined with a `boostWithdraw` on the original contract, the owner can steal the `boostToken` stored in this original contract.
Recommendation	Consider making the `farmChef` address immutable and indicating to the the community if the owner address is set to a multi-signature wallet.
Resolution	RESOLVED The client has removed the above functions.

Issue #21	**`minBoostAmount` can only be 0 or 500 ether**
Severity	MEDIUM SEVERITY

Location	Lines 76-80
	`if(_minBoostAmount==0) {`
	`  boostPara[_pid].minBoostAmount = _minBoostAmount;`
	`} else {`
	`  boostPara[_pid].minBoostAmount = 500 ether;`
	`}`

Description	If `_minBoostAmount` is set to an amount other than 0, the `minBoostAmount` will always be 500 ether. This might not be the expected mechanism since the `_minBoostAmount` value is practically unused.
Recommendation	Consider inverting the condition by assigning `boostPara[_pid].minBoostAmount = _minBoostAmount;` within the `else` clause. Assign the correct value if `_minBoostAmount` is 0.
Resolution	RESOLVED

Issue #22	**Various variables will affect user rewards in hindsight if ever changed by governance**
Severity	MEDIUM SEVERITY
Description	The purpose of the `HexagonBoost` contract is to increase the rewards of users according to various boosting parameters. Presently, the contract contains many functions for the owner to adjust these parameters. However, if they are ever changed, any unharvested rewards will eventually be harvested with the new parameters. This means if a user does not harvest for a year and the boosting parameters are suddenly increased, this user will be able to harvest all their rewards with the new parameters.
Recommendation	This issue is quite fundamental — the reason most staking contracts use the `accRewardPerShare` and `rewardDebt` mechanisms is exactly to resolve this issue. As rewriting the booster in this matter is exceptionally tedious, we believe this issue is difficult to solve without very intrusive code changes.
Resolution	ACKNOWLEDGED

Issue #23	**`getTeamRatio`, `getTotalBoostedAmount`, `boostStakedFor` and `boostTotalStaked` 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 lower gas usage in certain cases.
Recommendation	Consider marking the functions above as external.
Resolution	RESOLVED

Issue #24	Unused imports and approval
Severity	INFORMATIONAL

Description	`import "@boringcrypto/boring-solidity/contracts/libraries/BoringMath.sol";`
	`import "@boringcrypto/boring-solidity/contracts/BoringBatchable.sol";`
	`import "@boringcrypto/boring-solidity/contracts/BoringOwnable.sol";`
	
	Files that are imported in a contract but not used within said contract could confuse third-party auditors. They also increase the contract length unnecessarily.
	
	Line 88
	`IERC20(boostPara[_pid].boostToken).approve(farmChef,uint256(-1));`
	
	This approval is also not necessary.

Recommendation	Consider removing the aforementioned import to keep the contract short and simple. Consider also removing the approval.
Resolution	RESOLVED Client indicated that `BoringBatchable.sol` will be used.

Issue #25	Contract should inherit `IBoost`
Severity	INFORMATIONAL
Description	`HexagonBoost` is supposed to be called through the `IBoost` interface on `MiniChefV2`. Therefore, it should inherit `IBoost` for code consistencies and to ease third-party auditing processes.
Recommendation	Consider inheriting the required interface whenever possible.
Resolution	ACKNOWLEDGED

Issue #26	Lack of events on privileged functions
Severity	INFORMATIONAL
Description	The functions mentioned below are privileged functions which can change sensitive variables of the contract therefore should emit events as notifications. - `setMulsigAndFarmChef` - `setFixedTeamRatio` - `setFixedWhitelistPara` - `setWhiteListMemberStatus` - `setBoostFarmFactorPara` - `setBoostFunctionPara`
Recommendation	Add events for the above functions.
Resolution	RESOLVED

2.4 hexagonBoostStorage

`hexagonBoostStorage` is a simple data repository inherited by the HexagonBoost contract.

The `hexagonBoostStorage` contract keeps track of the total token supply for each pool ID, user balances, and whitelisted user addresses. The events that are emitted by `HexagonBoost` are also listed within the contract.

2.4.1 Issues & Recommendations

Issue #27	Typographical errors
Severity	INFORMATIONAL

Description	The contract contains a number of typographic mistakes which we have consolidated below in a single issue in an effort to keep the report size reasonable.
	
	Line 34
	`uint256 maxIncRatio;//5.5 multiple`
	
	The default value is `50 * SmallNumbers.FIXED_ONE`, instead of `5.5 multiple`.
	
	Line 38
	`uint256 log_para2;// 329*SmallNumbers.FIXED_ONE/10;`
	
	The default value is `329*rayDecimals/10`.
	
	Lines 20-23 & 44-52
	`//  log(LOG_PARA0)(amount+LOG_PARA1)- LOG_PARA2`
	`//  uint256 public LOG_PARA0 = 5;`
	`//  uint256 public LOG_PARA1 = 500000e18;`
	`//  uint256 public LOG_PARA2 = 329*SmallNumbers.FIXED_ONE/10;`
	
	`//uint256 public fixedTeamRatio = 80;  //default 8%`
	`//uint256 public fixedWhitelistRatio = 200;  //default 20%`
	`//uint256 public whiteListfloorLimit = 500000 ether; //default 500 thousands`
	`//uint256 constant internal rayDecimals = 1000e18;//100%`
	`//uint256 public BaseBoostTokenAmount = 1000e18;//1000 ether;`
	`//uint256 public BaseIncreaseRatio = 30e18; //3%`
	`//uint256 public RatioIncreaseStep = 10e18;// 1%`
	`//uint256 public BoostTokenStepAmount = 1000e18;//1000 ether;`
	`//uint256 public MaxFactor = 5500e18;//5.5 multiple`
	
	These comments are unused.

Recommendation	Consider fixing the typographical errors.

Resolution	PARTIALLY RESOLVED The comments on line 20-23 remain.

Issue #28	**Unused import: SmallNumbers.sol**
Severity	INFORMATIONAL
Location	Line 4 `import "../libraries/SmallNumbers.sol";`
Description	Files that are imported in a contract but not used within said contract could confuse third-party auditors. They furthermore increase the contract length for no reason.
Recommendation	Consider removing the import above to keep the contract short and simple.
Resolution	RESOLVED

2.5 boostMultiRewarderTime

The `boostMultiRewarderTime` contract is an extension of the `MiniChefV2`. The `MiniChefV2` mainly uses the `boostMultiRewarderTime` to distribute additional reward currencies apart from FLAKE.

Each pool in the `MiniChefV2` contract has a rewarder property which can be set to the zero address or an existing address. If set to a `boostMultiRewarderTime`, the `boostMultiRewarderTime` helps record users' reward eligibility by executing the `onFlakeReward` function and transfers `FlakeToken` as a reward or another token when `harvest` or `withdrawAndHarvest` is called in the `MiniChefV2` contract.

Additionally, if the booster has been defined in the `boostMultiRewarderTime` contract by the governance, users can receive boosted rewards similarly to how the rewards are boosted for the main reward currency.

Each rewarder should be used for at most one pool.

2.5.1 Privileged Roles

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

- `setRewardPerSecond`
- `add`
- `reclaimTokens`
- `setBooster`

2.5.2 Issues & Recommendations

Issue #29	**The `pending` variable is perpetually boosted and rewards excessive `rewardToken`**
Severity	HIGH SEVERITY
Location	Line 92 `(pending,,) = boostRewardAndGetTeamRoyalty(index,_user,oldAmount,pending);`
Description	If a rewarder is enabled, each deposit/withdrawal call on the `MiniChefV2` contract does an external call to the `boostMultiRewarderTime` contract's `onFlakeReward` function which caches the boosted pending variable to `unpaidRewards`. Consequently, users can harvest excessive reward tokens on top of the pending rewards each time because this `unpaidRewards` variable is perpetually boosted. For example, if the boost ratio is set to 2x, for each 2 reward tokens harvested, users can get an additional 2 tokens. Because the `unpaidRewards` are boosted on every deposit, if the user deposits again, those 2 tokens become 4. If called again, this then becomes 8, and so on. Using this flaw, a malicious party can amplify their rewards are barely any cost and drain the contract of all reward tokens.
Recommendation	Consider revising the function's logic to boost the pending rewards once The client can fix this by adding the `unpaidTokens` after the boosting.
Resolution	RESOLVED

Issue #30	**Governance Privilege: The function `reclaimTokens` allow the contract owner to withdraw any reward tokens to any address**
Severity	LOW SEVERITY
Description	The Owner can call the `reclaimTokens` function to withdraw any reward token stored in the contract to an address. Even though it might be unlikely that this is done by the owner, if the gov is an EOA there's always the risk of this private key getting stolen and a malicious party taking these tokens.
Recommendation	Consider being forthright with the community before calling reclaimTokens. Be clear on which address you are sending the tokens to and which tokens you will take.
Resolution	ACKNOWLEDGED The client has indicated that the owner will be a multi-signature wallet. This issue will be marked as resolved once this has been done.

Issue #31	**setRewardPerSecond has no maximum safeguard**
Severity	LOW SEVERITY
Description	The function to update rewards currently has no safeguard on its maximum value. Projects sometimes accidentally update their emission rate to a severely high number either by accident or malicious intent. By having a maximum value, the code itself enforces the reward rate to always be within a reasonable range, preventing mistakes which cannot be reverted once they are made. This might also boost investor confidence as they know that the governance cannot suddenly set the emission rate to a very high value.
Recommendation	Consider adding a `MAX_EMISSION_RATE` variable and setting it to a reasonable value. `require(_rewardPerSecond <= MAX_EMISSION_RATE,"Too high");` Consider also updating all pools before changing the emission rate to ensure this change does not affect rewards in hindsight.
Resolution	ACKNOWLEDGED

Issue #32	Typographical errors
Severity	INFORMATIONAL

Description	The contract contains several typographical errors that we have consolidated below in a single issue to keep the report size reasonable.
	
	Line 22
	`/// \`amount\` LP token amount the user has provided.`
	
	No `amount` variable exists in the struct.
	
	Line 30
	`/// \`allocPoint\` The amount of allocation points assigned to the pool.`
	
	No `allocPoint` variable exists in the struct.
	
	Line 46
	`address private immutable MASTERCHEF_V2;`
	
	Should be made `public` so users can check if the Masterchef is legitimate.
	
	Lines 172-179
	`// function set(uint256 _pid, uint256 _rewardPerSecond) public onlyOwner{}`
	
	This is an unused comment and therefore should be removed.
	
	Line 160
	`function add(address rewardToken,uint256 _rewardPerSecond) public onlyOwner {}`
	
	`rewardToken` should have the `IERC20` type.

	Line 249-250
	`function setBooster(address _booster) public onlyOwner {`
	`booster = IBoost(_booster);`
	`_booster` input can be marked as `IBoost` instead of `address` to avoid casting later.

Recommendation	Consider fixing the typographical errors.
Resolution	RESOLVED

Issue #33	**`chefPid` and `masterLpToken` 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 variables above explicitly `immutable`.
Resolution	PARTIALLY RESOLVED Only `masterLpToken` was made `immutable`.

Issue #34	Inconsistent usage of SafeMath functions
Severity	INFORMATIONAL
Description	Several lines still use arithmetic operations to compute mathematical equations. This could expose the contract to overflow/underflow risks as there are no internal checks for the used solidity version.
Recommendation	Consider using the SafeMath functions to compute instead of the arithmetic operations.
Resolution	RESOLVED

2.6 VeFlake

`VeFlake` is a staking contract that accepts `FlakeToken` as the deposit token to earn more `FlakeToken` over time. If users wish to withdraw the staked `FlakeToken` plus the accumulated interest, they will need to wait for the `LeavingTerm`, which is set by the governance.

`VeFlake` is also an `ERC20` contract that mints and burns tokens upon deposit and withdrawal. Tokens minted when using the `enter` function are equivalent to a user's share in the vesting pool. On the other hand, if users opt to cancel the vesting, they can use the `leaveApply` function to queue their cancellation. This queued leave can then be confirmed by calling the `leave` function. The user can then claim back their tokens according to their share in the pool. If the user decided to retract the leave request, they could do so by using the `cancelLeave` function, and they will get back their `VeFlake` tokens that were transferred upon calling `leaveApply`.

Users should note that the governance is able to use `setFlake` to modify the Flake token address that is accepted or returned by the contract, and `setLeavingTerm` which modifies the vesting period.

2.6.1 Privileged Roles

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

- `setFlake`
- `setLeavingTerm`
- `setMulsig`

2.6.2 Issues & Recommendations

Issue #35	**Users might not be able to leave the vesting contract due to the large memory caching of arrays and iterations**
Severity	HIGH SEVERITY

Location	Line 143
	`function searchPendingIndex(pendingItem[] memory pendingAry,uint64 firstIndex,uint64 searchTime) internal pure returns (int256){`
	
	Line 182
	`function getAllPendingAmount(pendingGroup memory userPendings) internal pure returns (uint256){`
	
	Line 190
	`function getReleasePendingAmount(pendingGroup memory userPendings,uint64 releaseTerm) internal view returns (uint256){`
	
	Line 243
	`pendingGroup memory userPendings = userLeavePendingMap[account];`

Description	Caching the `pendingItem` array and `pendingGroup` struct into memory will eventually cause all calls to the functions that do this to revert for users with many elements in their withdrawal array. The whole array is copied into memory, costing 800 gas per cached element after the 2019 hard fork (200 before this). Another thing to note is that iterations are an anti-pattern in Solidity due to the unpredictable amount of gas required to loop over an array. Depending on how the transactions are handled, problems such as out-of-gas or expensive gas costs can be the side effect.
Recommendation	Consider doing the binary search off-chain or allowing for this. Consider allowing users to specify their unlock index. Otherwise, consider revising the logic of the above functions and their dependencies to avoid caching large amounts of information. Copying an array into memory should be avoided.
Resolution	ACKNOWLEDGED

Issue #36	**`cancelLeave` transfers from `msg.sender` to `msg.sender`**
Severity	HIGH SEVERITY
Description	The `cancelLeave` function involves the ERC20 `transfer(to, amount)` function: `transfer(msg.sender,amount);` In the case of `cancelLeave`, the implementation goes as `transfer(msg.sender, amount)` which results in a `_transfer(msg.sender, msg.sender, amount)` call and therefore does not actually transfer tokens to the user. Therefore, users cannot presently receive any tokens if they try to leave the contract. This also shows that the current codebase is insufficiently tested.
Recommendation	Consider using `_transfer` to send tokens from the contract to `msg.sender`.
Resolution	RESOLVED The recommendation has been implemented.

Issue #37	**Governance Privilege: `setFlake` can be set again and prevent users from depositing or withdrawing from the contract**
Severity	HIGH SEVERITY

Location	Lines 47-48
	`function setFlake(IERC20 _flake) external onlyOrigin{`
	`    flake = _flake;`
	`}`

Description	The FLAKE token address is initially set by the governance to set up the contracts to accept the FLAKE token for deposits. However, there are no safeguards to prevent it from being called again. This can create a situation where user funds are stuck. The `setFlake` function could also be abused to empty the contract of all the FLAKE token by setting FLAKE to a fake token and depositing that fake token in order to record deposit balances. If the address is then changed to the real token, the governance can withdraw the real token.
Recommendation	Consider making the function callable only once or move the FLAKE set up in the constructor, make it immutable and remove the `setFlake` function.
Resolution	RESOLVED The function has been removed.

Issue #38	**`getLeaveApplyHistory` does not work as intended**
Severity	MEDIUM SEVERITY
Description	`getLeaveApplyHistory` has an unusable calculation because it loops only until `len`, and not until `userPendings.pendingAry.length`. `for(uint256 i=firstIndex;i<len;i++)` `getLeaveApplyHistory` does not work at all for any `firstIndex` that is non-zero. This is because the `i` variable will continue looping until `len` instead of the correct `userPendings.pendingAry.length`.
Recommendation	Consider looping until `userPendings.pendingAry.length` instead.
Resolution	RESOLVED The recommendation has been implemented.

Issue #39	**Governance Privilege: `setLeavingTerm` can be set to an arbitrarily high value by the governance**
Severity	MEDIUM SEVERITY

Location	`Lines 51-52`
	`function setLeavingTerm(uint64 _leavingTerm) external onlyOrigin{`
	`    LeavingTerm = _leavingTerm;`
	`}`

Description	Governance can set the `leavingTerm` to another period which can shorten or lengthen the vesting period of the FlakeToken.
Recommendation	Consider setting the `LeavingTerm` as constant and removing the function above. Alternatively, consider including a requirement that there should be no existing FLAKE tokens that are currently vesting before being able to modify the `LeavingTerm`.
Resolution	RESOLVED The function has been removed.

Issue #40	Unsafe casts are made throughout the contract that can cause issues
Severity	MEDIUM SEVERITY
Location	Line 171 `userPendings.pendingAry.push(pendingItem(uint192(amount),currentTime()));`
Description	Casting to another data type directly does not check for overflows and could cause unintended overflows. Helper libraries prevent this unchecked operation by reverting overflow transactions and preventing undesired exploits or bugs.
Recommendation	Consider using the SafeCast library from OpenZeppelin or moving all integer types to `uint256`.
Resolution	RESOLVED The client has added a casting function that checks for amount overflows and is used in the line above.

Issue #41	**The `searchPendingIndex` function might not pick the last index if that value occurs multiple times**
Severity	LOW SEVERITY
Location	Line 143 `function searchPendingIndex(pendingItem[] memory pendingAry,uint64 firstIndex,uint64 searchTime) internal pure returns (int256){`
Description	The binary search algorithm is currently incomplete in supplying the requirements to find the latest index that is unlockable. This is due to the binary search picking an arbitrary number in repeated sequences. For example, if we're searching for the index of "3" in the sequence 12334..., the binary search could pick the index 2 or 3 depending on the array length.
Recommendation	Consider revising the algorithm to be calculated off-chain. Alternatively, this issue can be resolved if this behaviour is acceptable, as the user would simply need to call the function a second time. Finally, the client may opt to extend the algorithm with a small for loop that increases the index for every duplicate behind it.
Resolution	RESOLVED The client has modified `addPendingIndex` to prevent element duplicates.

Issue #42	The first user could steal the tokens deposited by the next users by manipulating the share value
Severity	INFORMATIONAL
Location	Line 754 `uint256 what = _amount.mul(totalShares).div(totalFlake);`
Description	When the `totalShares` is really low, especially upon creation, an user can mint an infinitesimal amount of `veFLAKE` at a 1:1 ratio and send a big amount of FLAKE to the contract to make the `totalFlake` amount really big. When other users try to enter and stake their tokens, they may receive 0 `veFLAKE`, or in the best case, a rounded down number. And the first users that were able to get some `veFLAKE` will be able to steal the user's tokens deposited to the contract.
Recommendation	Consider permanently locking 1 FLAKE and 1 veFLAKE by minting it to this contract in the constructor, so `totalFlake` will always be at least equal to 1e18, ensuring no rounding down.
Resolution	RESOLVED The client plans to lock the tokens immediately to avoid this. The first enter will also need to enter with at least 1e18 as a small extra safeguard (small as they could still leave with 1e18 - 1).

Issue #43	Inconsistent usage of SafeMath functions
Severity	INFORMATIONAL
	Line 191 `uint64 curTime = currentTime()-releaseTerm;`
Description	Several lines still use arithmetic operations to compute mathematical equations and this could expose the contract to overflow/underflow risks as there are no internal checks yet for the used solidity version.
Recommendation	Consider using the SafeMath functions to compute instead of the arithmetic operations to prevent functions from overflow/underflow.
Resolution	PARTIALLY RESOLVED Other functions still utilize arithmetic operations to compute.

Issue #44	`enter` function does not support fee-on-transfer tokens
Severity	INFORMATIONAL
Description	Within the `enter` function, there is no logic that supports fee-on-transfer tokens. Therefore, during a deposit, the contract will receive fewer tokens than the user will get credited. This leads to an exploitation issue, where a malicious user can drain the whole contract.
Recommendation	Consider adding the logic for fee on transfer tokens.
Resolution	RESOLVED The client has stated that they will not be using any fee-on-transfer for FlakeToken.

Issue #45	**The `Enter`, `Leave`, `ApplyLeave`, and `CancelLeave` events are indexing amounts which are unnecessary**
Severity	INFORMATIONAL
Description	Indexing should be used on unique identifiers e.g. address or IDs to help with data queries. The events mentioned above are indexing token amounts which are not unique identifiers. This wastes gas as indexing is not free.
Recommendation	Consider removing indexed on token amount keys on the events mentioned above.
Resolution	RESOLVED

Issue #46	**Unnecessary casting of `tokenDecimal` to `uint8`**
Severity	INFORMATIONAL

Location	Line 40
	`uint256 tokenDecimal`
	
	Line 44
	`decimals_ = uint8(tokenDecimal);`

Description	`tokenDecimal` can be provided as `uint8` to avoid casting later on.
Recommendation	Consider casting `tokenDecimal` as `uint8` in the constructor to avoid unnecessary casting.
Resolution	RESOLVED

Issue #47	Several functions 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: - `name` - `symbol` - `decimals` - `enter` - `leaveApply` - `cancelLeave` - `leave` - `getUserReleasePendingAmount` - `getFlakeAmount` - `getVeFlakeShare` 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 functions as external.
Resolution	RESOLVED

Issue #48	Lack of `safeTransfer` usage within `enter` and `leave`
Severity	INFORMATIONAL

	Line 99
	`flake.transferFrom(msg.sender, address(this), _amount);`
	
	Line 136
	`flake.transfer(msg.sender, what);`

Description	The `transferFrom`/`transfer` method is used to transfer tokens from `msg.sender` to the contract and vice-versa. This will not work for tokens that return `false` on `transfer` (or malformed tokens that do not have a return value).
Recommendation	Consider importing OpenZeppelin's SafeERC20 from their repository and adding `using SafeERC20 for IERC20;` statement to your contract. This will allow the contract to use the `safeTransferFrom` instead of `transferFrom` and `safeTransfer` instead of `transfer`.
Resolution	RESOLVED

Issue #49	Typographical errors
Severity	INFORMATIONAL

Description	Line 14
	`require(msg.sender==safeMulsig, "not mulsafe");`
	Consider revising the error message of this requirement to be more specific: not multisig.
	
	Line 27
	`uint64 public LeavingTerm = 90 days;`
	Using smaller types of integers is usually more expensive if it is outside of a struct.
	
	Line 30
	`uint64 releaseTime;`
	The `releaseTime` variable indicates the time a leave is queued instead of the time of release. Consider renaming it to `timestamp`.
	
	Line 33
	`pendingItem[] pendingAry;`
	Renaming the variable can avoid confusion for third-party validators reading this contract. Consider renaming the above variable to `pendingArray` to make it more readable.
	
	Line 82
	`// Gets the amount of locked in the contract`
	Consider completing the comment by stating "locked flake", instead of "locked".
	
	Line 109
	`// require(getAllPendingAmount(userLeavePendingMap[msg.sender])>=_share,"veFlake: Leave insufficient amount");`
	This comment can be removed as it is unused.

	Line 214, 218
	`// If no veFlake exists, mint it 1:1 to the amount put in`
	
	`// Calculate and mint the amount of veFlake the flake is worth. The ratio will change overtime, as veFlake is burned/ minted and flake deposited + gained from fees / withdrawn.`
	
	The `getVeFlakeShare` function refers to minting multiple times while the actual logic does not mint.

Recommendation	Consider fixing the typographical errors.
Resolution	PARTIALLY RESOLVED `LeavingTerm` remains unchanged.

Issue #50	**Lack of events for function `setFlake`, `setLeavingTerm`, `setMulsig`**
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 The functions were removed.

Issue #51	`_amount` and `_share` should be required to be greater than zero in the various external functions
Severity	INFORMATIONAL

Location	Line 81, 104, 129
	`function enter(uint256 _amount) public {`
	`function leaveApply(uint256 _share) public {`
	`uint256 _share = updateUserPending(userLeavePendingMap[msg.sender],LeavingTerm);`

Description	Transferring a zero amount is ambiguous for the above functions their purpose and it is more practical to revert these transactions by requiring the user to enter an amount greater than zero.
Recommendation	Consider adding a requirement to the functions above that _amount and _share must be greater than zero.
Resolution	RESOLVED

2.7 FlakeToken

The FlakeToken contract will serve as the main reward token for the project. The contract follows a standard ERC20 token implementation which has `transfer`, `transferFrom`, `_mint`, `_burn`, `approve`, `increaseAllowance`, and `decreaseAllowance` functions.

The contract is a simple token that does not have a public/external function to `mint` and burn. Instead, tokens are pre-minted via `constructor` to several addresses: `initHolder`, `reserveHolder`, and `businessHolder`.

`initHolder` receives a net of 85 million tokens, the `reserveHolder` receives 10 million tokens, while `businessHolder` receives 5.0 million tokens.

2.7.1 Issues & Recommendations

Issue #52	Inconsistent usage of `uint256`
Severity	INFORMATIONAL
Location	Line 7 `using SafeMath for uint;`
Description	Within the contract, the types `uint256` and `uint` are both used. However, we recommend remaining consistent and only use `uint256`. Being consistent shows third-party validators that the code has been carefully thought through.
Recommendation	Consider using `uint256` throughout the contract.
Resolution	RESOLVED

Issue #53	Unnecessary typing of `tokenDecimal` to `uint256` while casting it to `uint8`
Severity	INFORMATIONAL

	Line 21
	`uint256 tokenDecimal`
	
	Line 30
	`decimals_ = uint8(tokenDecimal)`

Description	`tokenDecimal` in the constructor is marked as `uint256`. However, `decimals_` is only a `uint8` variable based on the initial declaration. Therefore, the result of the constructor input needs to be cast to `uint8`.
Recommendation	Consider marking the `tokenDecimal` as `uint8` in the constructor to avoid unnecessary casting.
Resolution	RESOLVED

Issue #54	Token amounts can be made more readable by separating the digits into thousands
Severity	INFORMATIONAL

	Lines 14, 15, 16
	`uint256 constant public MAX_TOTAL_TOKEN_AMOUNT = 100000000 ether;`
	`uint256 constant public MAX_RESERVE_AMOUNT = 10000000 ether; //10% for reserve`
	`uint256 constant public MAX_BUSINESS_EXPANDING = 5000000 ether; //5% for business expanding`

Description	The token amounts contain several trailing zeros which can confuse some users or third-party readers in determining the decimal places.
Recommendation	Consider adding an underscore for every thousand places to make the token amounts more readable.
Resolution	RESOLVED

Issue #55	`name`, `symbol` and `decimals` 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 functions as external.
Resolution	RESOLVED


PALADIN
BLOCKCHAIN SECURITY