Cyberscope

Audit Report

FOMO BULL CLUB

June 2024

Repository https://github.com/artiffine-vojtech/fmbc-contracts-tmp/tree/main

Commit 9c4e860ae2b86ec43d8060cbaf69dd442cf9af3c

Audited by © cyberscope

Table of Contents

Review

Contract Name	ControllerFactory
Testing Deploy	https://testnet.bscscan.com/address/0xFCA301E16B71100f5cB5c295c79247b388829775

Contract Name	ERC20MEME
Testing Deploy	https://testnet.bscscan.com/address/0x4A3c26c438882fa6E4C42aF90CD541C20ED51F24

Contract Name	FomoBullClubNFT
Testing Deploy	https://testnet.bscscan.com/address/0x18a7042221D296e42d6E93997Af76C921F2d04F1

Contract Name	TokenEmissionsController
Testing Deploy	https://testnet.bscscan.com/address/0x9C33e02072412742700710866 00B65B159d47B1e

Contract Name	TokenIncentivesController
Testing Deploy	https://testnet.bscscan.com/address/0xc20215948bbC56E01e6904E968608855afBdF8a5

Contract Name	TokenProxy

Testing Deploy	https://testnet.bscscan.com/address/0xA9823721C89078c1493BE4752c9544D4afeac2b3

Contract Name	BalancerDexProvider
Testing Deploy	https://testnet.bscscan.com/address/0x09BF0DF8dDDE3C92d59C816Db59a678aCE830FCD

Contract Name	UniswapV2DexProvider
Testing Deploy	https://testnet.bscscan.com/address/0xC47aa3268D036f87aE1C5fC5289d46264DA849C5

Contract Name	IdentityVerifier
Testing Deploy	https://testnet.bscscan.com/address/0xf6FACc0467b376AE52011B0D0AE6c6b6a43900f1

Contract Name	MEMEVesting
Testing Deploy	https://testnet.bscscan.com/address/0x6DA7117122436BFDF989753eB4f0f048ccd1e5aE

Contract Name	NFTChecker
Testing Deploy	https://testnet.bscscan.com/address/0x72Ba58b0Ed933f5B9284f83aB53d8ec390BC7D98

Contract Name	Launchpad

Testing Deploy	https://testnet.bscscan.com/address/0x1018bD920744320CEf0c2007adF5763cd30EaCb0

Contract Name	LaunchControl
Testing Deploy	https://testnet.bscscan.com/address/0x0E20832566eb9cb3F5036A284b961C4a5F248d02

Audit Updates

Initial Audit	05 Jun 2024

Source Files

Filename	SHA256
packages/hardhat/contracts/launchpad/NFTChecker.sol	39e8c61a1334b36d1834bd312ece017d8f974cb5ac4f8fee3163ebb532c6e340
packages/hardhat/contracts/launchpad/MEMEVesting.sol	934d888aba18f096427749fcb93c8ffc2f8f0f19a1fccdef9af5f4d82caccfe9
packages/hardhat/contracts/launchpad/Launchpad.sol	5e3edcaa20699dde96c86f1698f508adbc324639fa1019db9e9d155a7d3b4a8f
packages/hardhat/contracts/launchpad/IdentityVerifier.sol	2489f3e1622ddc2c971a699496dceef05658308696cb8f58f76409544ae6b285
packages/hardhat/contracts/launchpad/ERC20MEME.sol	469bc78d95c1f7f332aac3e52a0d4b004eb0d3c038e27ff6f3bbc82189a16036
packages/hardhat/contracts/launchpad/providers/UniswapV2DexProvider.sol	4c696300585189e11b13af68b9c872e5eb6613ff3fdc776da24492ecbe304c85

packages/hardhat/contracts/launchpad/providers/BalancerDexProvider.sol	0aa7e8f933ebe93b3b8ef6d515157d5909da9ad08c75a410518ad0a93081c031
packages/hardhat/contracts/launchpad/libraries/LaunchControl.sol	64df0d3531a5d9c37549565b5c8ca7be0bec2a9661796d7360b8c89f69ff98dc
packages/hardhat/contracts/launchpad/controllers/TokenProxy.sol	bb4a6c94c261a3ecfb2c42e82f0caed3fcf1f3126b7d6c750ba6e6b187bba32a
packages/hardhat/contracts/launchpad/controllers/TokenIncentivesController.sol	0a151edbbcd14450f0b7dbf9917b1c195e05acfee89fc084ac4daab7a89da1d7
packages/hardhat/contracts/launchpad/controllers/TokenEmissionsController.sol	4eade30eb1cce3cc2c5e18db856b5316fd8cdf84fbb3f5dfda141143c21ac78e
packages/hardhat/contracts/launchpad/controllers/ControllerFactory.sol	de52934970e05e16fa97d288474dcf89379920ee0174b82abac40ec2c7c6f245

Overview

FOMO BULL CLUB is a decentralized launchpad and liquidity hub designed to facilitate the seamless launch and support of new cryptocurrency tokens. The protocol's primary functionalities encompass three core components: the launchpad, staking, and vesting.

Launchpad

The launchpad is responsible for the initial deployment of new tokens. Users can pledge liquidity or their staked NFTs to participate in the launch. Upon meeting predefined conditions, a new token is created, liquidity is added to a decentralized exchange (DEX), staking contracts are established, and the vesting for Key Opinion Leaders (KOL) allocations commences.

Staking

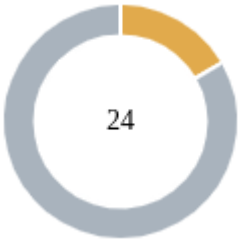
The staking contract offers a lock period feature, where rewards are scaled based on the duration of the lock. Additionally, users can stake their NFTs to receive boosted rewards. These rewards can be distributed in various tokens, and the system allows the administrator to add more reward options at any time.

Vesting

Vesting is managed through a contract that locks tokens for a specified duration, ensuring a controlled and gradual release.

FOMO BULL CLUB aims to provide a robust and flexible environment for launching new tokens, incentivizing participation through staking, and ensuring orderly token distribution through vesting.

Findings Breakdown



Critical	0
Medium	4
Minor / Informative	20

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	4	0	0	0
Minor / Informative	20	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
Medium	ITAT	Improper Token Allocation Tracking	Unresolved
Medium	PIB	Potential Incorrect Burning	Unresolved
Medium	PIM	Potential Incorrect Minting	Unresolved
Medium	PPI	Potential Precision Issue	Unresolved
Minor / Informative	IID	Improper Interface Declaration	Unresolved
Minor / Informative	APW	Admin Privileged Withdrawals	Unresolved
Minor / Informative	APIT	Allocation Parameter Inefficient Type	Unresolved
Minor / Informative	CCR	Contract Centralization Risk	Unresolved
Minor / Informative	DAU	Direct Address Usage	Unresolved
Minor / Informative	HD	Hardcoded Decimals	Unresolved
Minor / Informative	IDH	Improper dexIndex Handling	Unresolved
Minor / Informative	MEM	Misleading Error Messages	Unresolved
Minor / Informative	MEE	Missing Events Emission	Unresolved

●	MSC	Missing Sanity Check	Unresolved
●	PEVE	Potential Early Vesting Exit	Unresolved
●	PIAA	Potential Invalid Array Access	Unresolved
●	PRAV	Potential Replay Attack Vector	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	USEA	Unrestricted Start Emissions Access	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L18	Multiple Pragma Directives	Unresolved

ITAT - Improper Token Allocation Tracking

Criticality	Medium
Location	packages/hardhat/contracts/launchpad/Launchpad.sol#L303
Status	Unresolved

Description

The `claimTokens` function in the contract is responsible for allowing users to claim their allocated tokens from the launchpad. However, if the contract does not hold enough tokens to fulfill the user's total allocation, it transfers as many tokens as it currently has but does not keep track of the remaining tokens that the user is still entitled to claim. As a result, users may lose the rest of their allocation if the contract does not have sufficient tokens at the time of the claim.

```
function claimTokens(uint256 _launchId) external {
    LaunchConfig storage launchConfig = launches[_launchId];
    if (launchConfig.status != LaunchStatus.LAUNCHED) revert
LaunchIsNotLaunched();
    TokenAddressess storage addrs = tokenAddresses[_launchId];
    Pledge storage userPledge =
launchToUserPledge[_launchId][msg.sender];
    if (userPledge.claimed) revert AlreadyClaimed();

    // User allocation % multiplied by % of total sale allocation
    uint256 tokenAlloc = (
        (IERC20(addrs.token).totalSupply() * launchConfig.allocations[4]
* userPledge.lp) / launchConfig.values[9]
    ) / DEN;
    if (IERC20(addrs.token).balanceOf(address(this)) <= tokenAlloc) {
        tokenAlloc = IERC20(addrs.token).balanceOf(address(this));
    }
    IERC20(addrs.token).safeTransfer(msg.sender, tokenAlloc);
    userPledge.claimed = true;
}
```

Recommendation

To address this issue, the team is advised to implement a mechanism to track any remaining unclaimed tokens for each user. When the contract does not have enough tokens to fulfill a user's claim, a variable should be updated to reflect the amount of tokens still

owed. Additionally, the `claimTokens` function should be modified to allow users to claim their remaining tokens in future transactions once the contract has sufficient balance.

PIB - Potential Incorrect Burning

Criticality	Medium
Location	packages/hardhat/contracts/launchpad/controllers/TokenProxy.sol#L36
Status	Unresolved

Description

The contract `TokenProxy` facilitates depositing and withdrawing tokens to and from a single-staked incentive controller. However, a flaw exists in the withdrawal mechanism. In the `withdraw` function, the contract mistakenly burns tokens from its own balance (`address(this)`) instead of the balance of the caller (`msg.sender`).

```
function withdraw(uint256 _amount) external {
    controller.withdraw(_amount, msg.sender);
    _burn(address(this), _amount);
    proxiedToken.safeTransfer(msg.sender, _amount);
}
```

Recommendation

The team is advised to ensure tokens are burned from the balance of the caller (`msg.sender`) rather than from the contract's balance (`address(this)`).

PIM - Potential Incorrect Minting

Criticality	Medium
Location	packages/hardhat/contracts/launchpad/controllers/TokenProxy.sol#L26
Status	Unresolved

Description

The TokenProxy contract, which serves as an intermediate contract for depositing and withdrawing tokens to/from a staking contract, contains a flaw in the minting of "proof of deposit" tokens. Upon depositing tokens to the contract, the contract mints proof of deposit tokens not for the `msg.sender` but for the address of the contract it deposits to. Consequently, users who interact with this contract do not receive proof of deposit tokens.

```
function deposit(uint256 _amount, ITokenEmissionsController.LockTime
_lock) external {
    proxiedToken.safeTransferFrom(msg.sender, address(this), _amount);
    _mint(address(controller), _amount);
    controller.deposit(_amount, _lock);
}
```

Recommendation

The team is advised to correct the minting logic within the deposit function of the TokenProxy contract. Ensure that proof of deposit tokens are correctly minted for the `msg.sender` rather than the address of the staking contract.

PPI - Potential Precision Issue

Criticality	Medium
Location	packages/hardhat/contracts/launchpad/MEMEVesting.sol#L95
Status	Unresolved

Description

The `availableToClaim` function in the contract calculates the amount of tokens unlocked for a given account based on a predefined precision and unlock rate. However, due to precision limitations, there are cases where the `amountUnlocked` can be slightly larger than the originally vested amount. This discrepancy can cause subsequent claims to fail due to an insufficient balance.

The issue arises due to the precision factor. The computed value for `amountUnlocked` might end up slightly higher than the total vested amount (`vestingPosition.amount`). This happens because of the precision loss in the integer division and multiplication operations. The specific precision calculation for `PRECISION/UNLOCKED_MONTHLY` is `6.00240096` , which should be 6 instead.

When `amountUnlocked` exceeds the `vestingPosition.amount` , subsequent claims will attempt to withdraw more tokens than are actually vested. This will lead to a failure in the token transfer due to an insufficient balance.

```
uint256 constant UNLOCKED_MONTHLY = 1666;
...
uint256 constant PRECISION = 10000;
...
function availableToClaim(address _account, uint256 _positionIndex,
uint256 _timestamp) public view returns (uint256) {
    if (_timestamp < tgeTimestamp) return 0;
    VestingPosition storage vestingPosition =
vestingPositions[_account][_positionIndex];
    if (vestingPosition.cancelled) return 0;
    uint256 timeSinceStart = _timestamp -
vestingPosition.startTimestamp;
    uint256 numberOfUnlocks = timeSinceStart / 30 days;
    uint256 amountUnlocked = numberOfUnlocks >= NO_MONTHLY_UNLOCKS
        ? vestingPosition.amount
        : (vestingPosition.amount * (UNLOCKED_AT_TGE + (numberOfUnlocks
* UNLOCKED_MONTHLY))) / PRECISION;
    uint256 amountToClaim = amountUnlocked -
vestingPosition.amountClaimed;
    return amountToClaim;
}
```

Recommendation

To mitigate this issue, the team is advised to ensure that `amountUnlocked` never exceeds the initially vested amount. This can be achieved by adding a boundary check.

This boundary check ensures that the `amountUnlocked` will not exceed the original vesting amount, thus preventing the subsequent claims from failing due to insufficient balance.

IID - Improper Interface Declaration

Criticality	Minor / Informative
Location	packages/hardhat/contracts/launchpad/providers/BalancerDexProvider.sol#L8,10,64,68 packages/hardhat/contracts/launchpad/libraries/LaunchControl.sol#L16 packages/hardhat/contracts/launchpad/NFTChecker.sol#L9
Status	Unresolved

Description

It was observed that the contracts contain interface definitions directly within the contract files. This approach increases complexity, introduces duplicated code, and raises the risk of errors.

```
interface IAsset {}

interface IVault {
    ...
}

interface IWeightedPool {
    ...
}

interface IBalancerFactory {
    ...
}
...
interface IVesting {
    ...
}
...
interface IBalanceController {
    ...
}
```

Recommendation

To enhance code maintainability and reduce the risk of errors, we recommend the following best practices:

1. **Separate Interface Files:** Define each interface in its own separate file. This promotes modularity and makes it easier to manage changes.
2. **Use Import Statements:** Import the required interfaces into contract files using `import` statements. This reduces duplication and ensures consistency across the codebase.

By implementing these recommendations, the codebase will be more modular, maintainable, and less prone to errors.

APW - Admin Privileged Withdrawals

Criticality	Minor / Informative
Location	packages/hardhat/contracts/launchpad/controllers/TokenEmissionsController.sol#L111 packages/hardhat/contracts/launchpad/controllers/TokenIncentivesController.sol#L73
Status	Unresolved

Description

The `withdraw` function in the staking contract allows the `withdrawingAdmin` to withdraw staked tokens on behalf of any staker after the expiration of the locking period. This functionality can be exploited if the `withdrawingAdmin` account is compromised or misused.

```
function withdraw(uint _amount, address _onBehalfOf) external {
    require(msg.sender == _onBehalfOf || msg.sender == withdrawingAdmin,
'Not withdrawing admin');
    require(userLockTime[_onBehalfOf] <= block.timestamp, 'Locked');
    Balances storage bal = balances[_onBehalfOf];
    require(_amount <= bal.staked, 'Amount greater than staked');
    _updateReward(_onBehalfOf, rewardTokens);
    if (msg.sender == _onBehalfOf) {
        _getReward(rewardTokens);
    }
    uint scaled = _amount.mul(bal.lockBoost).div(10);
    if (bal.boosted) {
        uint multiplier = _getMultiplier(bal.nftId);
        scaled = scaled.mul(multiplier).div(10);
    }
    if (_amount == bal.staked) {
        scaled = bal.scaled;
        bal.lockBoost = 0;
    }
    bal.staked = bal.staked.sub(_amount);
    bal.lockScaled = bal.staked.mul(bal.lockBoost).div(10);
    bal.scaled = bal.scaled.sub(scaled);
    totalScaled = totalScaled.sub(scaled);
    stakingToken.safeTransfer(msg.sender, _amount);
    emit Withdrawn(_onBehalfOf, _amount, scaled);
}
```

Recommendation

To mitigate this issue, the privilege of the withdrawingAdmin should be restricted to prevent unauthorized withdrawals. Consider implementing a multisignature (multisig) mechanism where multiple trusted parties must approve an action. Additionally, role-based access control could be used to segregate duties and limit the scope of administrative actions.

APIT - Allocation Parameter Inefficient Type

Criticality	Minor / Informative
Location	packages/hardhat/contracts/launchpad/Launchpad.sol#L121
Status	Unresolved

Description

The contract implements a launchpad mechanism where allocations for a new token are defined. The total sum of these allocations must equal a constant value, `1e4`. The current implementation uses the `uint256` type for the allocation parameters, which introduces unnecessary inefficiency in terms of storage and gas consumption. Given that the maximum value for allocations is `1e4`, a smaller data type such as `uint16` would be sufficient and more optimal.

```
if (
    _config.allocations[0] + _config.allocations[1] +
_config.allocations[2] + _config.allocations[3]
        + _config.allocations[4] + _config.allocations[5] +
PLATFORM_MEME_FEE != DEN
) revert InvalidAllocations();
```

Recommendation

The team is advised to change the data type of the allocation parameters from `uint256` to `uint16`. By making this adjustment, the contract will utilize storage and gas more efficiently, ultimately providing better performance and lower operational costs for users and developers.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	packages/hardhat/contracts/launchpad/Launchpad.sol#L335,358,367,375,383,391,399,406,413,420 packages/hardhat/contracts/launchpad/NFTChecker.sol#L50,61 packages/hardhat/contracts/launchpad/MEMEVesting.sol#L140 packages/hardhat/contracts/launchpad/IdentityVerifier.sol#L29 packages/hardhat/contracts/launchpad/controllers/TokenProxy.sol#L44 packages/hardhat/contracts/launchpad/controllers/TokenIncentivesController.sol#L179,187,197 packages/hardhat/contracts/launchpad/controllers/TokenEmissionsController.sol#L223,232
Status	Unresolved

Description

The contract's functionality and behavior are heavily dependent on external parameters or configurations. While external configuration can offer flexibility, it also poses several centralization risks that warrant attention. Centralization risks arising from the dependence on external configuration include Single Point of Control, Vulnerability to Attacks, Operational Delays, Trust Dependencies, and Decentralization Erosion.

```
function setKolAddresses(address[] memory _kolAddresses, bool[] memory
_isKol) function setSoftCapAndFees(uint256 _softCap, uint256 _launchFee)
external onlyOwner
function setPledgeLimits(uint256 _min, uint256 _max) external onlyOwner
function setPledgeLimitsForKOLs(uint256 _min, uint256 _max) external
onlyOwner
function setSteakPlatformFee(uint256 _fee) external onlyOwner
function setMemePlatformFee(uint256 _fee) external onlyOwner
function setControllerFactory(address _controllerFactory) external
onlyOwner
function setSteakIC(address _steakIC) external onlyOwner
function setFomoIC(address _fomoIC) external onlyOwner
function addDexProvider(address _dexProvider) external onlyOwner
...
function addIncentivesController(address _controller) external onlyAdmin
function removeIncentivesController(uint index) external onlyOwner
...
function cancelVesting(address _account) external onlyOwner
...
function setSigner(address signer) public onlyAdmin
...
function setController(address _controller) external onlyOwner
...
function addReward(address _rewardToken) external onlyAdmin
function setWithdrawingAdmin(address _withdrawingAdmin) external
onlyOwner
function notifyReward(address[] calldata _rewardTokens, uint[] calldata
_amounts, uint _rewardsDuration) external onlyAdmin
...
function addReward(address _rewardToken) external onlyOwner
function notifyReward(address[] calldata _rewardTokens, uint[] calldata
_amounts, uint _rewardsDuration) external onlyAdmin
```

Recommendation

To address this finding and mitigate centralization risks, it is recommended to evaluate the feasibility of migrating critical configurations and functionality into the contract's codebase itself. This approach would reduce external dependencies and enhance the contract's self-sufficiency. It is essential to carefully weigh the trade-offs between external configuration flexibility and the risks associated with centralization.

DAU - Direct Address Usage

Criticality	Minor / Informative
Location	packages/hardhat/contracts/launchpad/libraries/LaunchControl.sol#L192,231,235 packages/hardhat/contracts/launchpad/controllers/ControllerFactory.sol#L31,38
Status	Unresolved

Description

In the contract, the address `0x000000000000000000000000000000000000dEaD` is used directly in multiple functions. This address is often referred to as the "dead address".

```
IERC20(_vars.fomo).safeTransfer(0x000000000000000000000000000000000000dEaD, fomoToBurn);
...
// Transfer LP tokens to dead address
IERC20(_addrs.usdcLP).safeTransfer(
    0x000000000000000000000000000000000000dEaD,
    IERC20(_addrs.usdcLP).balanceOf(address(this))
);
IERC20(_addrs.fomoLP).safeTransfer(
    0x000000000000000000000000000000000000dEaD,
    IERC20(_addrs.fomoLP).balanceOf(address(this))
);
...
TokenEmissionsController usdcLPController = new TokenEmissionsController(
    IERC20(_usdcLP),
    INFTWithLevel(_memberNFT),
    _token,
    0x000000000000000000000000000000000000dEaD
);
...
TokenEmissionsController fomoLPController = new TokenEmissionsController(
    IERC20(_fomoLP), INFTWithLevel(_memberNFT), _token,
0x000000000000000000000000000000000000dEaD
);
```

Recommendation

To enhance code readability and maintainability, it is recommended to define the dead address as a constant variable. This approach centralizes the address definition, making it easier to manage and reducing the likelihood of errors.

HD - Hardcoded Decimals

Criticality	Minor / Informative
Location	packages/hardhat/contracts/launchpad/Launchpad.sol#L28,30,32,34,36,38,113,186,357,359,366,367,380,381
Status	Unresolved

Description

A potential issue was identified related to the handling of decimal values. The contract utilizes hardcoded decimal values instead of dynamically retrieving them through the `.decimals()` function. This practice can introduce risks and inefficiencies into the contract's functionality.

Various variables such as `USDC_SOFT_CAP`, `LAUNCH_FEE`, `USDC_MIN`, and `USDC_MAX` are declared with values that include decimal points multiplied by a factor of 1e6, indicating an assumed precision of six decimal places. However, the decimals for the underlying token are not dynamically fetched from the token contract.

Using hardcoded decimal values can lead to discrepancies and inaccuracies, especially if the underlying token's decimals differ from the assumed value. This approach also lacks flexibility, as it requires manual adjustments if the token's decimal precision changes in the future.

```
uint256 public USDC_SOFT_CAP = 100_000 * 1e6;
uint256 public LAUNCH_FEE = 5_000 * 1e6;
uint256 public USDC_MIN = 50 * 1e6;
uint256 public USDC_MAX = 1_000 * 1e6;
uint256 public USDC_KOL_MIN = 500 * 1e6;
uint256 public USDC_KOL_MAX = 5_000 * 1e6;
if (_config.hardCap * 1e6 <= USDC_SOFT_CAP) revert InvalidHardCap();
_config.hardCap * 1e6,
if (_softCap > 0) USDC_SOFT_CAP = _softCap * 1e6
LAUNCH_FEE = _launchFee * 1e6;
if (_min > 0) USDC_MIN = _min * 1e6;
if (_max > 0) USDC_MAX = _max * 1e6;
if (_min > 0) USDC_KOL_MIN = _min * 1e6;
if (_max > 0) USDC_KOL_MAX = _max * 1e6;
```

Recommendation

To enhance the robustness and adaptability of the contract, the team is advised to dynamically fetch the decimals of the underlying token using the `.decimals()` function provided by the ERC-20 standard. By doing so, the contract can accurately handle values based on the actual decimal precision of the token, mitigating potential risks associated with hardcoded decimal assumptions.

IDH - Improper dexIndex Handling

Criticality	Minor / Informative
Location	packages/hardhat/contracts/launchpad/Launchpad.sol#L157
Status	Unresolved

Description

In the `createLaunch` function of the contract, the `dexIndex` parameter from the `LaunchConfigVars` struct is used to select a DEX provider from the `dexProviders` array. The `dexIndex` parameter is not validated before it is used, which can lead to various out-of-bounds access or other unexpected behavior.

```
function createLaunch(LaunchConfigVars memory _config, bool _staked,
bytes calldata _data) external {
    ...
    launches.push(
        LaunchConfig({
            ...
            dexProvider: address(dexProviders[_config.dexIndex]),
            ...
        })
    );
    ...
}
```

Recommendation

The team is advised to add validation logic to ensure that the `dexIndex` is within the valid range of the `dexProviders` array. Implement robust error handling to manage scenarios where `dexIndex` might be invalid, ensuring that the contract fails gracefully without causing unexpected states or vulnerabilities. By incorporating the appropriate checks, you can prevent invalid `dexIndex` values from causing issues, ensuring the stability and security of the contract.

MEM - Misleading Error Messages

Criticality	Minor / Informative
Location	packages/hardhat/contracts/launchpad/Launchpad.sol#L358,374,382,411 packages/hardhat/contracts/launchpad/controllers/TokenIncentivesController.sol#L257 packages/hardhat/contracts/launchpad/controllers/TokenEmissionsController.sol#L300
Status	Unresolved

Description

The contract is using misleading error messages. These error messages do not accurately reflect the problem, making it difficult to identify and fix the issue. As a result, the users will not be able to find the root cause of the error.

```
require(_launchFee < USDC_SOFT_CAP)
require(_fee <= 2000)
require(_dexProvider != address(0))
require(rewardData[_rewardToken].lastUpdateTime == 0)
```

Recommendation

The team is suggested to provide a descriptive message to the errors. This message can be used to provide additional context about the error that occurred or to explain why the contract execution was halted. This can be useful for debugging and for providing more information to users that interact with the contract.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	packages/hardhat/contracts/launchpad/IdentityVerifier.sol#L29 packages/hardhat/contracts/launchpad/NFTChecker.sol#L50,61 packages/hardhat/contracts/launchpad/controllers/TokenProxy.sol#L43 packages/hardhat/contracts/launchpad/Launchpad.sol#L389,396,403,410
Status	Unresolved

Description

The contract performs actions and state mutations from external methods that do not result in the emission of events. Emitting events for significant actions is important as it allows external parties, such as wallets or dApps, to track and monitor the activity on the contract. Without these events, it may be difficult for external parties to accurately determine the current state of the contract.

```
function setSigner(address signer) public onlyAdmin
...
function addIncentivesController(address _controller) external onlyAdmin
function removeIncentivesController(uint index) external onlyOwner
...
function setController(address _controller) external onlyOwner
...
function setControllerFactory(address _controllerFactory) external
onlyOwner
function setSteakIC(address _steakIC) external onlyOwner
function setFomoIC(address _fomoIC) external onlyOwner
function addDexProvider(address _dexProvider) external onlyOwner
```

Recommendation

It is recommended to include events in the code that are triggered each time a significant action is taking place within the contract. These events should include relevant details such as the user's address and the nature of the action taken. By doing so, the contract will be more transparent and easily auditable by external parties. It will also help prevent potential issues or disputes that may arise in the future.

MSC - Missing Sanity Check

Criticality	Minor / Informative
Location	packages/hardhat/contracts/launchpad/IdentityVerifier.sol#L29 packages/hardhat/contracts/launchpad/providers/UniswapV2DexProvider.sol#L21 packages/hardhat/contracts/launchpad/Launchpad.sol#L76,389
Status	Unresolved

Description

The contract does not properly check for the validity of the initialized address in the constructor. If the addresses are not initialized correctly, the contract will not function as intended.

```
constructor(address signer)
function setSigner(address signer) public onlyAdmin
...
constructor(address _router, address _factory)
...
constructor(
    address _fomoUsdcLp,
    address _steakIC,
    address _fomoIC,
    address _memberNFT,
    address _nftChecker,
    address _controllerFactory,
    address _identityVerifier,
    address _dexProvider
)
function setControllerFactory(address _controllerFactory) external
onlyOwner
```

Recommendation

It is recommended that the contracts implement proper sanity check to ensure that parameters addresses are correct. By adding a verification process, the contract can ensure that the contract will function as intended.

PEVE - Potential Early Vesting Exit

Criticality	Minor / Informative
Location	packages/hardhat/contracts/launchpad/MEMEVesting.sol#L140
Status	Unresolved

Description

The `cancelVesting` function in the vesting contract allows the contract owner to cancel the vesting for any specified account prematurely. This function is controlled by the `onlyOwner` modifier, meaning only the owner of the contract has the authority to execute it.

```
function cancelVesting(address _account) external onlyOwner {
    VestingPosition[] storage positions = vestingPositions[_account];
    for (uint256 i = 0; i < positions.length; i++) {
        if (positions[i].cancelled) continue;
        positions[i].cancelled = true;
        uint256 cancelledAmount = positions[i].amount -
positions[i].amountClaimed;
        positions[i].amountClaimed = positions[i].amount;
        memeToken.safeTransfer(owner(), cancelledAmount);
    }
}
```

Recommendation

To mitigate the risks associated with the `cancelVesting` function, consider implementing one or more of the following improvements:

1. **Multi-Signature Authorization**: Require multiple signatures from a predefined set of trusted parties to authorize the cancellation of vesting positions. This reduces the risk of a single point of control.
2. **Time-Locked Cancellations**: Implement a time delay between the initiation of the cancellation and its execution, allowing beneficiaries to prepare or challenge the action if necessary.

PIAA - Potential Invalid Array Access

Criticality	Minor / Informative
Location	packages/hardhat/contracts/launchpad/Launchpad.sol#L333
Status	Unresolved

Description

The contract contains a function named `setKolAddresses`, which is designed to set Kol addresses along with their corresponding boolean values. While the function seems to perform this task, it contains a vulnerability in the form of an unverified array length within a nested loop.

Specifically, the function starts by iterating over the `_kolAddresses` array and checks if each address corresponds to the boolean value in the `_isKol` array. However, if the boolean value for an address is false, the function enters a nested loop to remove that address from the `kolAddresses` array. This nested loop iterates over the `_kolAddresses` array again, using a different iterator `j`, searching for the index of the address to remove from `kolAddresses`.

The issue arises because the nested loop iterates up to `_kolAddresses.length` without verifying if this length matches the actual length of the `kolAddresses` array. This discrepancy could lead to out-of-bounds access.

```
function setKolAddresses(address[] memory _kolAddresses, bool[] memory
_isKol) external onlyOwner {
    if (_kolAddresses.length != _isKol.length) revert
ArraysLengthMismatch();
    for (uint256 i = 0; i < _kolAddresses.length; i++) {
        if (isKOL[_kolAddresses[i]] != _isKol[i]) {
            isKOL[_kolAddresses[i]] = _isKol[i];
            if (_isKol[i]) {
                kolAddresses.push(_kolAddresses[i]);
            } else {
                for (uint256 j = 0; j < _kolAddresses.length; j++) {
                    if (kolAddresses[j] == _kolAddresses[i]) {
                        kolAddresses[j] =
kolAddresses[kolAddresses.length - 1];
                        kolAddresses.pop();
                        break;
                    }
                }
            }
        }
    }
}
```

Recommendation

To enhance the security and efficiency of the contract, it is recommended to verify the length of the `kolAddresses` array before entering the nested loop. This ensures that the loop iterates over valid indices, mitigating the risk of out-of-bounds access.

PRAV - Potential Replay Attack Vector

Criticality	Minor / Informative
Location	packages/hardhat/contracts/launchpad/IdentityVerifier.sol#L37
Status	Unresolved

Description

The `verify` function in the contract implementation performs signature verification to ensure that messages are signed by a specific signer and are recent. However, it does not ensure that a message with the same timestamp and identity cannot be reused within that window. This omission can potentially allow an attacker to reuse a valid signature within the 24-hour period.

Additionally, the function does not account for the possibility of the same contract being deployed on multiple chains, which can further expose it to replay attacks across different chains.

```
function verify(address identity, bytes calldata data) external view
override returns (bool) {
    (uint40 sigTimestamp, bytes32 message, bytes memory signature) =
abi.decode(data, (uint40, bytes32, bytes));
    bytes32 expectedMessage = keccak256(abi.encodePacked('\x19Ethereum
Signed Message:\n25', identity, sigTimestamp));
    if (message != expectedMessage) return false;
    if (message.recover(signature) != _signer) return false;
    if (uint256(sigTimestamp) < block.timestamp - 1 days) return false;
    return true;
}
```

Recommendation

To fully mitigate the risk of replay attacks, it is recommended to incorporate nonce and chain ID into the message being signed. This ensures that each signature is unique and can only be used once, and it also binds the signature to a specific blockchain, preventing cross-chain replay attacks.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	packages/hardhat/contracts/launchpad/controllers/TokenIncentivesController.sol packages/hardhat/contracts/launchpad/controllers/TokenEmissionsController.sol
Status	Unresolved

Description

SafeMath is a popular Solidity library that provides a set of functions for performing common arithmetic operations in a way that is resistant to integer overflows and underflows.

Starting with Solidity versions that are greater than or equal to 0.8.0, the arithmetic operations revert to underflow and overflow. As a result, the native functionality of the Solidity operations replaces the SafeMath library. Hence, the usage of the SafeMath library adds complexity, overhead and increases gas consumption unnecessarily in cases where the explanatory error message is not used.

```
library SafeMath {...}
```

Recommendation

The team is advised to remove the SafeMath library in cases where the revert error message is not used. Since the version of the contract is greater than `0.8.0` then the pure Solidity arithmetic operations produce the same result.

If the previous functionality is required, then the contract could exploit the `unchecked { ... }` statement.

Read more about the breaking change on https://docs.soliditylang.org/en/v0.8.16/080-breaking-changes.html#solidity-v0-8-0-breaking-changes.

USEA - Unrestricted Start Emissions Access

Criticality	Minor / Informative
Location	packages/hardhat/contracts/launchpad/controllers/TokenEmissionsController.sol#L66
Status	Unresolved

Description

The function `startEmissions` within the contract poses a risk due to its lack of access control mechanisms. This function allows any user to trigger emissions and set the emission parameters, including the emissions amount and duration.

```
function startEmissions(EmissionPoint[] memory _emissions) external {
    require(emissions.length == 0, 'Emissions already started');
    require(_emissions.length > 0, 'No emissions');
    uint256 length = _emissions.length;
    uint256 emissionsSum;
    for (uint256 i = 0; i < length; i++) {
        require(_emissions[i].duration > 0 && _emissions[i].amount > 0,
'Invalid emission');
        emissionsSum += _emissions[i].amount;
        emissions.push(_emissions[i]);
    }
    emissionsStart = block.timestamp;
    IERC20(rewardTokens[0]).safeTransferFrom(msg.sender, address(this),
emissionsSum);
    _setRewardsDuration(emissions[currentEmissionsIndex].duration);
    Reward storage r = rewardData[rewardTokens[0]];
    r.balance = emissions[currentEmissionsIndex].amount;
    _notifyReward(rewardTokens[0],
emissions[currentEmissionsIndex].amount, rewardsDuration);
}
```

Recommendation

The team is advised to implement robust access controls within the `startEmissions` function. Access should be restricted to authorized addresses, such as the contract owner or designated administrators, who can be entrusted with the responsibility of initiating emissions.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	packages/hardhat/contracts/utils/Adminable.sol#L39,49,59 packages/hardhat/contracts/launchpad/NFTChecker.sol#L50,79 packages/hardhat/contracts/launchpad/MEMEVesting.sol#L50,79,95,111,119,120,140 packages/hardhat/contracts/launchpad/libraries/LaunchControl.sol#L61,62,63,153,154,155 packages/hardhat/contracts/launchpad/Launchpad.sol#L28,30,32,34,36,38,40,101,207,216,217,218,219,220,221,231,250,305,324,333,356,365,373,381,389,396,403,410
Status	Unresolved

Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```
address _admin
address _controller
address _incentivesController
uint256 _tokenId
address _identity
address _to
uint256 _amount
uint256[] calldata _positionIndexes
uint256 _positionIndex
uint256 _timestamp
address _account
ILaunchCommon.LaunchConfig storage _launchConfig
ILaunchCommon.TokenAddressess storage _addrs
Vars memory _vars

...
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.
Find more information on the Solidity documentation
https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	packages/hardhat/contracts/launchpad/Launchpad.sol#L357,366,375,380,381,392 packages/hardhat/contracts/launchpad/controllers/TokenEmissionsController.sol#L251 packages/hardhat/contracts/launchpad/controllers/TokenIncentivesController.sol#L253
Status	Unresolved

Description

Events are a way to record and log information about changes or actions that occur within a contract. They are often used to notify external parties or clients about events that have occurred within the contract, such as the transfer of tokens or the completion of a task.

It's important to carefully design and implement the events in a contract, and to ensure that all required events are included. It's also a good idea to test the contract to ensure that all events are being properly triggered and logged.

```
USDC_SOFT_CAP = _softCap * 1e6
USDC_MIN = _min * 1e6
PLATFORM_STEAK_FEE = _fee
PLATFORM_MEME_FEE = _fee
if (_min > 0) USDC_KOL_MIN = _min * 1e6;
if (_max > 0) USDC_KOL_MAX = _max * 1e6;
...
rewardsDuration = _newRewardsDuration;
```

Recommendation

By including all required events in the contract and thoroughly testing the contract's functionality, the contract ensures that it performs as intended and does not have any missing events that could cause issues with its arithmetic.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	packages/hardhat/contracts/launchpad/MEMEVesting.sol#L100,101 packages/hardhat/contracts/launchpad/libraries/LaunchControl.sol#L96,110,117,188,189,192 packages/hardhat/contracts/launchpad/Launchpad.sol#L104,106,285,291,474,478,480
Status	Unresolved

Description

It is important to be aware of the order of operations when performing arithmetic calculations. This is especially important when working with large numbers, as the order of operations can affect the final result of the calculation. Performing divisions before multiplications may cause loss of prediction.

```
uint256 numberOfUnlocks = timeSinceStart / 30 days
uint256 amountUnlocked = numberOfUnlocks >= NO_MONTHLY_UNLOCKS
            ? vestingPosition.amount
            : (vestingPosition.amount * (UNLOCKED_AT_TGE +
(numberOfUnlocks * UNLOCKED_MONTHLY))) / PRECISION

uint256 platformFeePercent = (_launchConfig.values[12] * DEN) / (DEN -
_launchConfig.values[11])
uint256 usdcForTeam = (usdcAmount * platformFeePercent) / DEN
maxPledgeLP < (((((user.minPledge * 1e12) / 2) / totalUsdc) * totalLP) /
1e12)
```

Recommendation

To avoid this issue, it is recommended to carefully consider the order of operations when performing arithmetic calculations in Solidity. It's generally a good idea to use parentheses to specify the order of operations. The basic rule is that the multiplications should be prior to the divisions.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	packages/hardhat/contracts/launchpad/Launchpad.sol#L93,390 packages/hardhat/contracts/launchpad/IdentityVerifier.sol#L19,30
Status	Unresolved

Description

The contract performs operations on variables that have been configured on user-supplied input. These variables are missing of proper check for the case where a value is zero. This can lead to problems when the contract is executed, as certain actions may not be properly handled when the value is zero.

```
CONTROLLER_FACTORY = _controllerFactory
_signer = signer
```

Recommendation

By adding the proper check, the contract will not allow the variables to be configured with zero value. This will ensure that the contract can handle all possible input values and avoid unexpected behavior or errors. Hence, it can help to prevent the contract from being exploited or operating unexpectedly.

L18 - Multiple Pragma Directives

Criticality	Minor / Informative
Location	packages/hardhat/contracts/launchpad/Launchpad.sol#L3 packages/hardhat/contracts/launchpad/IdentityVerifier.sol#L3
Status	Unresolved

Description

If the contract includes multiple conflicting pragma directives, it may produce unexpected errors. To avoid this, it's important to include the correct pragma directive at the top of the contract and to ensure that it is the only pragma directive included in the contract.

```
pragma solidity 0.8.23;
pragma solidity ^0.8.9;
```

Recommendation

It is important to include only one pragma directive at the top of the contract and to ensure that it accurately reflects the version of Solidity that the contract is written in.

By including all required compiler options and flags in a single pragma directive, the potential conflicts could be avoided and ensure that the contract can be compiled correctly.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
NFTChecker	Implementation	INFTChecker, Adminable		
		Public	✓	-
	addIncentivesController	External	✓	onlyAdmin
	removeIncentivesController	External	✓	onlyOwner
	getIncentivesControllersCount	External		-
	isNftStaked	External		-
	getStakedNFTIds	Public		-
	_verify	Internal		
	_isNftStaked	Internal		
MEMEVesting	Implementation	IMEMEVesting, Ownable		
		Public	✓	-
	vestTokens	External	✓	-
	claimTokens	External	✓	-
	availableToClaim	Public		-
	getVestingPositions	External		-
	getVestingSchedule	External		-
	cancelVesting	External	✓	onlyOwner

Launchpad	Implementation	ILaunchpad, Ownable		
		Public	✓	-
	createLaunch	External	✓	-
	pledge	External	✓	-
	pledgeWithNFT	External	✓	-
	getFundsBack	External	✓	-
	launch	External	✓	-
	claimTokens	External	✓	-
	getLaunchConfig	External		-
	setKolAddresses	External	✓	onlyOwner
	setSoftCapAndFees	External	✓	onlyOwner
	setPledgeLimits	External	✓	onlyOwner
	setSteakPlatformFee	External	✓	onlyOwner
	setMemePlatformFee	External	✓	onlyOwner
	setControllerFactory	External	✓	onlyOwner
	setSteakIC	External	✓	onlyOwner
	setFomoIC	External	✓	onlyOwner
	addDexProvider	External	✓	onlyOwner
	_pledge	Internal	✓	
	_getMultiplier	Internal		
IdentityVerifier	Implementation	Adminable, IIdentityVerifier		
		Public	✓	-

	supportsInterface	Public		-
	setSigner	Public	✓	onlyAdmin
	verify	External		-
ERC20MEME	Implementation	ERC20		
		Public	✓	ERC20
UniswapV2DexProvider	Implementation	IDexProvider		
		Public	✓	-
	createLP	External	✓	-
	getPoolBalance	External		-
	breakLP	External	✓	-
BalancerDexProvider	Implementation	IDexProvider		
	createLP	External	✓	-
	breakLP	External	✓	-
	getPoolBalance	External		-
	_convertERC20sToAssets	Internal		
LaunchControl	Library			
	launch	External	✓	-
	tryToEndLaunch	External	✓	-
	_createLP	Internal	✓	

	_startEmissions	Internal	✓	
TokenProxy	Implementation	ERC20, Ownable, ITokenProxy		
		Public	✓	ERC20
	deposit	External	✓	-
	withdraw	External	✓	-
	setController	External	✓	onlyOwner
TokenIncentivesController	Implementation	ITokenIncentivesController, Adminable		
		Public	✓	Adminable
	deposit	External	✓	-
	withdraw	External	✓	-
	stakeNFT	External	✓	-
	unstakeNFT	External	✓	-
	getReward	External	✓	-
	lastTimeRewardApplicable	Public		-
	claimableRewards	External		-
	addReward	External	✓	onlyAdmin
	setWithdrawingAdmin	External	✓	onlyOwner
	notifyReward	External	✓	onlyAdmin
	_getReward	Internal	✓	
	_rewardPerToken	Internal		

	_earned	Internal		
	_addReward	Internal	✓	
	_notifyReward	Internal	✓	
	_updateReward	Internal	✓	
	_getMultiplier	Internal		
TokenEmissionsController	Implementation	ITokenEmissionsController, Adminable		
		Public	✓	Adminable
	startEmissions	External	✓	-
	deposit	External	✓	-
	withdraw	External	✓	-
	stakeNFT	External	✓	-
	unstakeNFT	External	✓	-
	getReward	External	✓	-
	lastTimeRewardApplicable	Public		-
	claimableRewards	External		-
	addReward	External	✓	onlyOwner
	notifyReward	External	✓	onlyAdmin
	_setRewardsDuration	Internal	✓	
	_getReward	Internal	✓	
	_rewardPerToken	Internal		
	_earned	Internal		
	_addReward	Internal	✓	

	_notifyReward	Internal	✓	
	_updateReward	Internal	✓	
	_getMultiplier	Internal		
ControllerFactory	Implementation	IControllerFactory		
	createNewTokenControllers	External	✓	-

Inheritance Graph

See the detailed images in the github repository.

Flow Graph

See the detailed images in the github repository.

Summary

FOMO BULL CLUB is an interesting project that has a friendly and growing community. Its contracts implement a launchpad for automated meme token launching.The Smart Contract analysis reported no compiler error or critical issues. This audit investigates security issues, business logic concerns and potential improvements.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope's prior written consent. This report is not nor should be considered an "endorsement" or "disapproval" of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

The Cyberscope team

https://www.cyberscope.io