Competitive Security Assessment

MagpieLaunchpad

Nov 22nd, 2023

Secure3

secure3.io

Summary

This report is prepared for the project to identify vulnerabilities and issues in the smart contract source code. A group of NDA covered experienced security experts have participated in the Secure3's Audit Contest to find vulnerabilities and optimizations. Secure3 team has participated in the contest process as well to provide extra auditing coverage and scrutiny of the finding submissions.

The comprehensive examination and auditing scope includes:

- Cross checking contract implementation against functionalities described in the documents and white paper disclosed by the project owner.
- Contract Privilege Role Review to provide more clarity on smart contract roles and privilege.
- Using static analysis tools to analyze smart contracts against common known vulnerabilities patterns.
- Verify the code base is compliant with the most up-to-date industry standards and security best practices.
- Comprehensive line-by-line manual code review of the entire codebase by industry experts.

The security assessment resulted in findings that are categorized in four severity levels: Critical, Medium, Low, Informational. For each of the findings, the report has included recommendations of fix or mitigation for security and best practices.

Overview

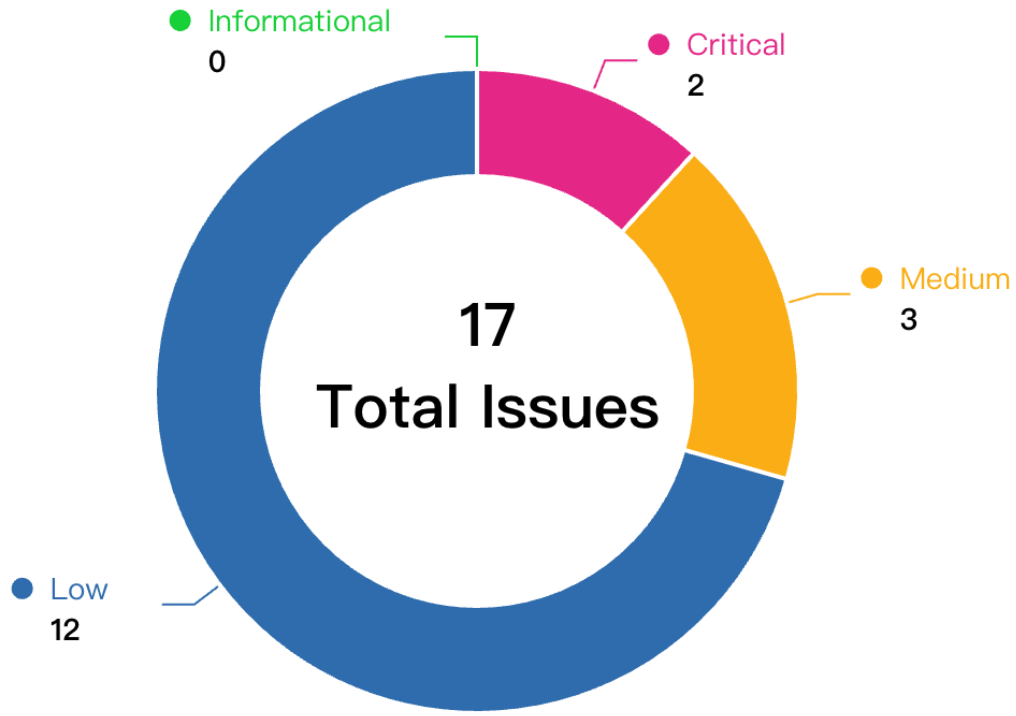
Project Detail

Project Name	MagpieLaunchpad
Platform & Language	Solidity
Codebase	• https://github.com/magpiexyz/magpie_contracts • audit commit - 736ef3cb796f8f9cb3d85aec9df4af89966916aa • final commit - 32f32614cc4c75ecc0a8edb9f0115684c761f8f3
Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

Audit Scope

File	SHA256 Hash
./contracts/launchpad/Launchpad.sol	abc32c37f067b7533b14b3ac6e89b4fa9f766fe7a2700db5005c3c34f154ab71
./contracts/launchpad/LaunchpadVesting.sol	d16b9160f8c719082b558e02d4023e6db40d57f31b0ac812ad7edebfbdedd20b
./contracts/interfaces/ILaunchpadVesting.sol	abf3ec06c8e3325f2a394757fbdb2be6a450835ca3e795bf7b773fa9f381cd58

Code Assessment Findings



ID	Name	Category	Severity	Client Response	Contributor
MGP-1	Potential risk of funds being locked up	Logical	Critical	Fixed	Kong7ych3
MGP-2	`endtime` needs to be strictly incremented	Logical	Critical	Fixed	Kong7ych3
MGP-3	Potential rounding down in the function `_tokenAllocBySale()`	Logical	Medium	Fixed	biakia, Yaodao
MGP-4	Potential incorrect updating `lowFDVAmount` and `highFDVAmount`	Logical	Medium	Fixed	Yaodao

MGP-5	Ownership change should use two-step process	Code Style	Medium	Mitigated	biakia
MGP-6	`isLofFDV` need to infer	Logical	Low	Fixed	0xhuy0512
MGP-7	Missing input validation	Logical	Low	Fixed	biakia
MGP-8	Centralization Risks	Privilege Related	Low	Mitigated	Secure3
MGP-9	Incorrect index check	Logical	Low	Fixed	0xhuy0512, biakia, Kong7ych3
MGP-10	Owner can still call to `Launchpad.addPhase()`, `Launchpad.setPhase()` and `Launchpad.configLaunchpad()` after sale active time	Logical	Low	Acknowledged	Yaodao, 0xhuy0512, Kong7ych3, infinityhacker
MGP-11	No guarantee about sending project token to Launchpad contract	Logical	Low	Fixed	0xhuy0512
MGP-12	Unnecessary decimals input in `Launchpad.configLaunchpad()` function can lead to very bad result	Logical	Low	Fixed	0xhuy0512
MGP-13	Raised tokens will not be sent to the treasury	Logical	Low	Fixed	biakia
MGP-14	DENOMINATOR constant is too small, lead to unwanted sale price	Logical	Low	Fixed	0xhuy0512
MGP-15	Allow owner to call `renounceOwnership()` can lead the contracts to ownerless	Privilege Related	Low	Mitigated	0xhuy0512
MGP-16	Miss 0 amount check	Code Style	Low	Fixed	NoodleDonn212
MGP-17	Set time variable should more than `block.timestamp`	Code Style	Low	Fixed	NoodleDonn212

MGP-1:Potential risk of funds being locked up

Category	Severity	Client Response	Contributor
Logical	Critical	Fixed	Kong7ych3

Code Reference

- code/contracts/launchpad/Launchpad.sol#L168

```
168:if (user.tokenClaimed) return (0, 0);
```

Description

Kong7ych3 : In the Launchpad contract, the owner can use the startClaimingPhase function to start the claim cycle after all Phases are completed. At this time, the user can claim the projectToken through the claim function. After the user successfully claims it, his tokenClaimed will be set to true, which prevents the user from claiming the token repeatedly. But at this time, the owner may still conduct a new round of Phase through the addPhase function (maybe the tokens have not been sold out, and other users have the intention to buy after the phase ends). In the new Phase, any user can perform the buy operation (including users who have already claimed it). Unfortunately, when a user who has claimed successfully purchases, he will not be able to claim after the Phase ends, because the protocol will check its tokenClaimed status, so the funds subscribed in the new Phase will be locked and cannot You cannot receive the projectToken even if you take it out.

Here is a simple example:

1. When all Phases end, the owner calls the startClaimingPhase function to set canClaimTokens to true.
2. User 1 calls the claim function to claim the token, and its tokenClaimed status is set to true.
3. The owner calls the addPhase function to add a new Phase.
4. User 1 calls the buy function to purchase again.
5. After the Phase ends, user 1 calls the claim function to claim the token. Since its tokenClaimed status is true, getExpectedClaimAmount will return 0.
6. User 1 cannot claim the tokens at this time, nor can he get back the purchased funds.

Recommendation

Kong7ych3 : If re-opening purchases during the claim cycle is not allowed, it is recommended to check that canClaimTokens must be false in the addPhase and setPhase functions. If reopening is allowed, checking the user's tokenClaimed status in the isSaleActive modifier must be false.

Client Response

Fixed. Added variables for public/priority claimed amounts. So now users tokens will not be locked and can be claimed correctly.

MGP-2: `endtime` needs to be strictly incremented

Category	Severity	Client Response	Contributor
Logical	Critical	Fixed	Kong7ych3

Code Reference

- code/contracts/launchpad/Launchpad.sol#L155
- code/contracts/launchpad/Launchpad.sol#L254
- code/contracts/launchpad/Launchpad.sol#L277

```
155:if (currentBlockTimestamp < phaseInfo.endTime) return (i + 1, phaseInfo);

254:endTime: endTime,

277:phase.endTime = endTime;
```

Description

Kong7ych3 : In the Launchpad contract, users can add a new phase through the addPhase function and modify the phase through the setPhase function, but it does not strictly check whether endTime is reasonable. If endTime is not realistic, the phase obtained by the getCurrentPhaseInfo function will not conform to common sense.

For example: phaseInfos[0].endTime is set to 1, phaseInfos[1].endTime is set to 10, phaseInfos[2].endTime is set to 5, phaseInfos[3].endTime is set to 12. The current time is 4, then the current phase obtained by the getCurrentPhaseInfo function is phaseInfos[1], and the next phase will be phaseInfos[3]. the phaseInfos[2] will be skipped.

Recommendation

Kong7ych3 : It is recommended to perform strict checks on endTime:

1. When the phaseInfos length is 0, you should check whether the first added phase endTime is greater than startTime. This will effectively avoid the situation where the wrong endTime makes phase unusable.
2. When adding a new phase, you should check whether the incoming endTime is greater than the endTime of the previous phase (i.e. `endTime > phaseInfos[phaseInfos.length-1].endTime). This will effectively prevent new phases from conflicting with previous phase cycles.
3. When performing the setPhase operation, you should check whether the incoming endTime is greater than the endTime of the previous phase, and whether it is less than the endTime of the next phase. This will prevent the currently ongoing phase from suddenly moving forward or backward due to modification of endTime.

Client Response

Fixed. Added proper checks for ensuring that endTime parameter value is correct in addPhase and setPhase functions.

MGP-3:Potential rounding down in the function `_tokenAllocBySale()`

Category	Severity	Client Response	Contributor
Logical	Medium	Fixed	biakia, Yaodao

Code Reference

- code/contracts/launchpad/Launchpad.sol#L385-L393

```
385:function _tokenAllocBySale(
386:        uint256 _saleTokenAmount,
387:        PhaseInfo memory phaseInfo
388:    ) internal view returns (uint256) {
389:        uint256 numerator = _saleTokenAmount * phaseInfo.tokenPerSaleToken * projectTokenDecima
l;
390:        uint256 denominator = DENOMINATOR * saleTokenDecimal;
391:
392:        return numerator / denominator;
393:    }
```

Description

biakia : There is a potential rounding down issue in the function `_tokenAllocBySale`:

```
function _tokenAllocBySale(
        uint256 _saleTokenAmount,
        PhaseInfo memory phaseInfo
    ) internal view returns (uint256) {
        uint256 numerator = _saleTokenAmount * phaseInfo.tokenPerSaleToken * projectTokenDecimal;
        uint256 denominator = DENOMINATOR * saleTokenDecimal;

        return numerator / denominator;
    }
```

Let's say the `saleToken` is `WETH` and the `saleTokenDecimal` is 1e18. The `projectToken` has a decimal of 6 so the `projectTokenDecimal` is 1e6 and we assume that the `phaseInfo.tokenPerSaleToken` is equal to `DENOMINATOR`. The `numerator` will be `_saleTokenAmount * DENOMINATOR * 1e6` and the `denominator` will be `DENOMINATOR*1e18`. The result will be `_saleTokenAmount /1e12`. When the `_saleTokenAmount` is less than 1e12`WETH`, the result will be 0 due to the rounding down issue.

Yaodao : The function `_tokenAllocBySale()` is used to calculate the amount of project amount. The decimals of

sale tokens and project tokens is handled in the function. However, due to the division truncation problem in solidity. The result may rounding down.

```
    function _tokenAllocBySale(
        uint256 _saleTokenAmount,
        PhaseInfo memory phaseInfo
    ) internal view returns (uint256) {
        uint256 numerator = _saleTokenAmount * phaseInfo.tokenPerSaleToken * projectTokenDecimal;
        uint256 denominator = DENOMINATOR * saleTokenDecimal;

        return numerator / denominator;
    }
```

For example, the decimal of `projectToken` is 6 and the decimal of `saleToken` is 18, and the `tokenPerSaleToken` is 1e10. As a result, the project token amount should be `_saleTokenAmount * 1e10 * 1e6 / (10000 * 1e18)`. When the `_saleTokenAmount` is below 1e6, the result should be 0.

Recommendation

biakia : Consider reverting the call when the result of `_tokenAllocBySale` is 0.
Yaodao : Recommend adding the minimum value limit for the `_saleTokenAmount` and setting the `tokenPerSaleToken` to be a suitable value.

Client Response

Fixed. Added revert when _tokenAllocBySale = 0 and also added min_sale_token_amount to contract.

MGP-4:Potential incorrect updating `lowFDVAmount` and `highFDVAmount`

Category	Severity	Client Response	Contributor
Logical	Medium	Fixed	Yaodao

Code Reference

- code/contracts/launchpad/LaunchpadVesting.sol#L167-L173
- code/contracts/launchpad/Launchpad.sol#L187-L204

```
167:function _processVesting(bool isLowFDVedVesting, uint256 amount, address vestFor) internal {
168:        if (isLowFDVedVesting) {
169:            vestingInfo[vestFor].lowFDVAmount = amount;
170:        } else {
171:            vestingInfo[vestFor].highFDVAmount = amount;
172:        }
173:    }

187:function claim() external whenNotPaused isClaimable nonReentrant {
188:        (uint256 lowFDVPurchased, uint256 highFDVPurchased) = getExpectedClaimAmount(msg.sende
r);
189:
190:        if (lowFDVPurchased == 0 && highFDVPurchased == 0) revert InvalidAmount();
191:
192:        UserInfo storage user = userInfo[msg.sender];
193:        user.tokenClaimed = true;
194:
195:        if (user.lowFDVPurchased != 0) {
196:            _processClaims(true, user.lowFDVPurchased, msg.sender);
197:        }
198:
199:        if (user.highFDVPurchased != 0) {
200:            _processClaims(false, user.highFDVPurchased, msg.sender);
201:        }
202:
203:        emit Claim(msg.sender, lowFDVPurchased, highFDVPurchased);
204:    }
```

Description

Yaodao : The function `_processVesting()` is used to update the `lowFDVAmount` and `highFDVAmount` of the given address. Since it is updated by direct overrides but not cumulatively, the amount will be incorrect when the update times is not only 1.

```
    function _processVesting(bool isLowFDVedVesting, uint256 amount, address vestFor) internal {
        if (isLowFDVedVesting) {
            vestingInfo[vestFor].lowFDVAmount = amount;
        } else {
            vestingInfo[vestFor].highFDVAmount = amount;
        }
    }
```

The function `_processVesting()` is called by the function `vestTokens()`, which is only can be called by the contract `Launchpad`. In the contract `Launchpad`, the function `LaunchpadVesting.vestTokens()` is called in the function `Launchpad.claim()` and the `user.tokenClaimed` will be used to limit the call times of `LaunchpadVesting.vestTokens()` for each address to be 1.

However, in the contract `LaunchpadVesting`, the function `setLaunchpad()` can be called to update the address of `launchpadContract`, and the `user.tokenClaimed` will be false again in the new `Launchpad` contract.

As a result, the call times of `vestTokens()` for the same address can be over 1 and the `lowFDVAmount` and `highFDVAmount` will be incorrect.

Recommendation

Yaodao : Recommend updating the `lowFDVAmount` and `highFDVAmount` via `+=` instead of direct override.

Consider the follow fix:

```
    if (isLowFDVedVesting) {
        vestingInfo[vestFor].lowFDVAmount += amount;
    } else {
        vestingInfo[vestFor].highFDVAmount += amount;
    }
```

Client Response

Fixed. Instead of direct override we are cumulating values, so it will work correctly if the user claims more than one time (in case the user added a new phase afterwards ending the claiming process of previous phases).

MGP-5:Ownership change should use two-step process

Category	Severity	Client Response	Contributor
Code Style	Medium	Mitigated	biakia

Code Reference

- code/contracts/launchpad/LaunchpadVesting.sol#L11-L16
- code/contracts/launchpad/Launchpad.sol#L14-L19

```
11:contract LaunchpadVesting is
12:    Initializable,
13:    OwnableUpgradeable,
14:    ReentrancyGuardUpgradeable,
15:    PausableUpgradeable
16:{

14:contract Launchpad is
15:    Initializable,
16:    OwnableUpgradeable,
17:    ReentrancyGuardUpgradeable,
18:    PausableUpgradeable
19:{
```

Description

biakia : The contracts `Launchpad` and `LaunchpadVesting` do not implement a two-step process for transferring ownership. So ownership of the contract can be easily lost when making a mistake when transferring ownership.

Recommendation

biakia : Consider using `Ownable2StepUpgradeable` (https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/blob/master/contracts/access/Ownable2StepUpgradeable.sol) instead.

Client Response

Mitigated. For every project we are already using multisig as Owner so this one is already resolved.

MGP-6: `isLofFDV` need to infer

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	0xhuy0512

Code Reference

- code/contracts/launchpad/Launchpad.sol#L246
- code/contracts/launchpad/Launchpad.sol#L265

```
246:function addPhase(

265:function setPhase(
```

Description

0xhuy0512 : As the comment in the codebase said:

- `PhaseInfo.priorityMultiplier`: `> 0` for priority sale, `= 0` for public sale in DENOMINATOR
- `PhaseInfo.isLofFDV`: is `true` for priority sale, is `false` for priority sale

Based on this, the conclusion is:

- Whenever the phase is for priority sale, `priorityMultiplier` must be greater than `0` AND `isLofFDV` must be `true`
- Whenever the phase is for public sale, `priorityMultiplier` must be equal to `0` AND `isLofFDV` must be `false` But in the code, there's not check for above invariants, hence owner can freely set the phase's `priorityMultiplier` to 0, `isLofFDV` to `true` or `priorityMultiplier` to greater than 0, `isLofFDV` to `false`

The impact of this:

- Public users can't buy token in the public phase because DOS in this line in `Launchpad#L360`: `if (userPurchased[identifier] > _userCap) revert ExceedsUserPriorityCap();`
- Priority users who buy token in priority sale have the vesting duration like public vesting
- Priority users who buy token in public sale have the vesting duration like priority vesting

Recommendation

0xhuy0512 : Consider doing this to avoid unnecessary mistake:

In `Launchpad.addPhase()`:

```
    function addPhase(
        ...
-       bool isLofFDV
    ) external onlyOwner {
+       bool isLofFDV;
+       if (priorityMultiplier > 0) { phase.isLofFDV = true;}
+       else { phase.isLofFDV = false; }
        ...

        phaseInfos.push(newPhase);
    }
```

In `Launchpad.setPhase()`:

```
    function setPhase(
        uint256 index,
        uint256 endTime,
        uint256 saleCap,
        uint256 tokenPerSaleToken,
        uint256 priorityMultiplier,
-       bool isLofFDV
    ) external onlyOwner {
        if (index > phaseInfos.length) revert InvalidPhase();

        PhaseInfo storage phase = phaseInfos[index];

        phase.endTime = endTime;
        phase.saleCap = saleCap;
        phase.tokenPerSaleToken = tokenPerSaleToken;
        phase.priorityMultiplier = priorityMultiplier;
-       phase.isLofFDV = isLofFDV;

+       if (priorityMultiplier > 0) { phase.isLofFDV = true;}
+       else { phase.isLofFDV = false; }
    }
```

Client Response

Fixed. now value of isLofFDV variable is getting set by code as per priorityMultiplier value.

MGP-7:Missing input validation

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	biakia

Code Reference

- code/contracts/launchpad/Launchpad.sol#L284-L313

```
284:function configLaunchpad(
285:        address _projectToken,
286:        address _saleToken,
287:        address _vestingContract,
288:        address _treasury,
289:        uint256 _startTime,
290:        uint256 _maxToDistribute,
291:        uint256 _maxToRaise,
292:        uint256 _lowFDVVestingPart,
293:        uint256 _highFDVVestingPart,
294:        uint256 _projectTokenDecimal,
295:        uint256 _saleokenDecimal
296:    ) public onlyOwner {
297:        if (_treasury == address(0)) revert ZeroAddress();
298:        if (_maxToDistribute == 0) revert InvalidAmount();
299:        if (_maxToRaise == 0) revert InvalidAmount();
300:
301:        projectToken = IERC20(_projectToken);
302:        saleToken = IERC20(_saleToken);
303:        vestingContract = ILaunchpadVesting(_vestingContract);
304:        startTime = _startTime;
305:        treasury = _treasury;
306:        max_launch_tokens_to_distribute = _maxToDistribute;
307:        maxRaiseAmount = _maxToRaise;
308:        LOW_FDV_VESTING_PART = _lowFDVVestingPart;
309:        HIGH_FDV_VESTING_PART = _highFDVVestingPart;
310:
311:        projectTokenDecimal = _projectTokenDecimal;
312:        saleTokenDecimal = _saleokenDecimal;
313:    }
```

Description

biakia : The function `configLaunchpad` does not check if `_saleToken`, `_vestingContract` and `_projectToken` are zero address. The function `configLaunchpad` does not check if `_lowFDVVestingPart` and `_highFDVVestingPart` are lower than `DENOMINATOR`.

Recommendation

biakia : Consider adding checks for these variables:

```
        if (_projectToken == address(0)) revert ZeroAddress();
        if (_saleToken == address(0)) revert ZeroAddress();
        if (_vestingContract == address(0)) revert ZeroAddress();
        if (_lowFDVVestingPart >= DENOMINATOR) revert InvalidAmount();
        if (_highFDVVestingPart >= DENOMINATOR) revert InvalidAmount();
```

Client Response

Fixed. Added required input validations in configLaunchpad function.

MGP-8:Centralization Risks

Category	Severity	Client Response	Contributor
Privilege Related	Low	Mitigated	Secure3

Code Reference

- code/contracts/launchpad/Launchpad.sol#L265-L271
- code/contracts/launchpad/Launchpad.sol#L284-L295
- code/contracts/launchpad/Launchpad.sol#L318-L321

```
265:function setPhase(
266:        uint256 index,
267:        uint256 endTime,
268:        uint256 saleCap,
269:        uint256 tokenPerSaleToken,
270:        uint256 priorityMultiplier,
271:        bool isLofFDV

284:function configLaunchpad(
285:        address _projectToken,
286:        address _saleToken,
287:        address _vestingContract,
288:        address _treasury,
289:        uint256 _startTime,
290:        uint256 _maxToDistribute,
291:        uint256 _maxToRaise,
292:        uint256 _lowFDVVestingPart,
293:        uint256 _highFDVVestingPart,
294:        uint256 _projectTokenDecimal,
295:        uint256 _saleokenDecimal

318:function emergencyWithdrawFunds(address token, uint256 amount) external whenPaused onlyOwner {
319:        IERC20(token).safeTransfer(owner(), amount);
320:
321:        emit EmergencyWithdraw(token, amount);
```

Description

Secure3 : The owner role in Launchpad has high privileges which poses security risks if the private key is compromised.

- The owner can call `emergencyWithdrawFunds` in Launchpad without any constraints to withdraw funds to its own address from the contract.
- The owner can modify the endTime of phases in `setPhase`. Modify the endTime can call `withdrawUnsoldTokens` bypass hasEnded validation.
- The owner has no limits set when calling `configLaunchpad` to set critical parameters.

Recommendation

Secure3 : It is recommended to use multi-signature wallet or DAO governance to manage for Launchpad contract.

Client Response

Mitigated.For every project we are already using multisig as Owner so this one is already resolved.

MGP-9:Incorrect index check

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	0xhuy0512, biakia, Kong7ych3

Code Reference

- code/contracts/launchpad/Launchpad.sol#L265-L282
- code/contracts/launchpad/Launchpad.sol#L273

```
265:function setPhase(
266:        uint256 index,
267:        uint256 endTime,
268:        uint256 saleCap,
269:        uint256 tokenPerSaleToken,
270:        uint256 priorityMultiplier,
271:        bool isLofFDV
272:    ) external onlyOwner {
273:        if (index > phaseInfos.length) revert InvalidPhase();
274:
275:        PhaseInfo storage phase = phaseInfos[index];
276:
277:        phase.endTime = endTime;
278:        phase.saleCap = saleCap;
279:        phase.tokenPerSaleToken = tokenPerSaleToken;
280:        phase.priorityMultiplier = priorityMultiplier;
281:        phase.isLofFDV = isLofFDV;
282:    }

273:if (index > phaseInfos.length) revert InvalidPhase();
```

Description

0xhuy0512 : In `Launchpad.setPhase()` there's this line:

- `if (index > phaseInfos.length) revert InvalidPhase();`

Notice that this requirement will not reverted if `index = phaseInfos.length`.
biakia : In contract `Launchpad`, the function `setPhase` is used to reset the phase info:

```
function setPhase(
        uint256 index,
        uint256 endTime,
        uint256 saleCap,
        uint256 tokenPerSaleToken,
        uint256 priorityMultiplier,
        bool isLofFDV
    ) external onlyOwner {
        if (index > phaseInfos.length) revert InvalidPhase();

        PhaseInfo storage phase = phaseInfos[index];

        phase.endTime = endTime;
        phase.saleCap = saleCap;
        phase.tokenPerSaleToken = tokenPerSaleToken;
        phase.priorityMultiplier = priorityMultiplier;
        phase.isLofFDV = isLofFDV;
    }
```

The `if` condition is incorrect here. Consider the `phaseInfos` has 2 phase, so the `phaseInfos.length` is 2. The `index` can be 2 because `2>2` is false. Then we will try to read `phaseInfos[2]`. Since the `phaseInfos` only has 2 phase. This read will encounter an array-out-of-bounds error.

Kong7ych3 : In the Launchpad contract, the owner role can modify the phase through the setPhase function. It will first check whether the incoming phase index is greater than the length of phaseInfos to prevent the setPhase operation from failing due to an incoming index that is too large. But the maximum index of phaseInfos is `phaseInfos.length - 1`, so `index > phaseInfos.length` misses the check when the index is `phaseInfos.length`. The check will pass when the index passed in by the user is `phaseInfos.length`, but will fail when getting phaseInfos[phaseInfos.length].

Recommendation

0xhuy0512 :

```
    function setPhase(
        uint256 index,
        uint256 endTime,
        uint256 saleCap,
        uint256 tokenPerSaleToken,
        uint256 priorityMultiplier,
        bool isLofFDV
    ) external onlyOwner {
-       if (index > phaseInfos.length) revert InvalidPhase();
+       if (index >= phaseInfos.length) revert InvalidPhase();
        PhaseInfo storage phase = phaseInfos[index];

        phase.endTime = endTime;
        phase.saleCap = saleCap;
        phase.tokenPerSaleToken = tokenPerSaleToken;
        phase.priorityMultiplier = priorityMultiplier;
        phase.isLofFDV = isLofFDV;
    }
```

biakia : Consider following fix:

```
  if (index >= phaseInfos.length) revert InvalidPhase();
```

Kong7ych3 : It is recommended to use the `>=` symbol instead of the `>` symbol for checking. Consider the following fixes:

```
    function setPhase(
        uint256 index,
        ...
    ) external onlyOwner {
        if (index >= phaseInfos.length) revert InvalidPhase();
        ...
    }
```

Client Response

Fixed.Already fixed in audit 2.0

MGP-10:Owner can still call to `Launchpad.addPhase()`, `Launchpad.setPhase()` and `Launchpad.configLaunchpad()` after sale active time

Category	Severity	Client Response	Contributor
Logical	Low	Acknowledged	Yaodao, 0xhuy0512, Kong7ych3, infinityhacker

Code Reference

- code/contracts/launchpad/Launchpad.sol#L128-L133
- code/contracts/launchpad/Launchpad.sol#L246-L263
- code/contracts/launchpad/Launchpad.sol#L265-L283
- code/contracts/launchpad/Launchpad.sol#L284-L313

```
128:function hasEnded() public view returns (bool) {
129:        uint256 length = phaseInfos.length;
130:        if (length == 0) return true;
131:
132:        return phaseInfos[length - 1].endTime <= _currentBlockTimestamp();
133:    }

246:function addPhase(
247:        uint256 endTime,
248:        uint256 saleCap,
249:        uint256 tokenPerSaleToken,
250:        uint256 priorityMultiplier,
251:        bool isLofFDV
252:    ) external onlyOwner {
253:        PhaseInfo memory newPhase = PhaseInfo({
254:            endTime: endTime,
255:            saleCap: saleCap,
256:            allocatedAmount: 0,
257:            tokenPerSaleToken: tokenPerSaleToken,
258:            priorityMultiplier: priorityMultiplier,
259:            isLofFDV: isLofFDV
260:        });
261:
262:        phaseInfos.push(newPhase);
263:    }

265:function setPhase(
266:        uint256 index,
267:        uint256 endTime,
268:        uint256 saleCap,
269:        uint256 tokenPerSaleToken,
270:        uint256 priorityMultiplier,
271:        bool isLofFDV
272:    ) external onlyOwner {
273:        if (index > phaseInfos.length) revert InvalidPhase();
274:
275:        PhaseInfo storage phase = phaseInfos[index];
276:
277:        phase.endTime = endTime;
278:        phase.saleCap = saleCap;
279:        phase.tokenPerSaleToken = tokenPerSaleToken;
280:        phase.priorityMultiplier = priorityMultiplier;
```

```
281:         phase.isLofFDV = isLofFDV;
282:     }

284:function configLaunchpad(
285:         address _projectToken,
286:         address _saleToken,
287:         address _vestingContract,
288:         address _treasury,
289:         uint256 _startTime,
290:         uint256 _maxToDistribute,
291:         uint256 _maxToRaise,
292:         uint256 _lowFDVVestingPart,
293:         uint256 _highFDVVestingPart,
294:         uint256 _projectTokenDecimal,
295:         uint256 _saleokenDecimal
296:     ) public onlyOwner {
297:         if (_treasury == address(0)) revert ZeroAddress();
298:         if (_maxToDistribute == 0) revert InvalidAmount();
299:         if (_maxToRaise == 0) revert InvalidAmount();
300:
301:         projectToken = IERC20(_projectToken);
302:         saleToken = IERC20(_saleToken);
303:         vestingContract = ILaunchpadVesting(_vestingContract);
304:         startTime = _startTime;
305:         treasury = _treasury;
306:         max_launch_tokens_to_distribute = _maxToDistribute;
307:         maxRaiseAmount = _maxToRaise;
308:         LOW_FDV_VESTING_PART = _lowFDVVestingPart;
309:         HIGH_FDV_VESTING_PART = _highFDVVestingPart;
310:
311:         projectTokenDecimal = _projectTokenDecimal;
312:         saleTokenDecimal = _saleokenDecimal;
313:     }
```

Description

Yaodao : The function of `hasEnded()` is used to check whether the phase has end. The `endTime` of last `phaseInfo` is used to compared with the current block timestamp.

```
    function hasEnded() public view returns (bool) {
        uint256 length = phaseInfos.length;
        if (length == 0) return true;

        return phaseInfos[length - 1].endTime <= _currentBlockTimestamp();
    }
```

The functions `addPhase()` and `setPhase()` are used to add new `phaseInfo` or update the current `phaseInfo`. The `endTime` of the new `phaseInfo` should larger than the front `phaseInfo` and smaller than the after `phaseInfo`, but this is not checked.

As a result, assuming that the current `phaseInfos` has 10 `phaseInfo` and the `endTime` of the last `phaseInfo` is 10000 and the current block timestamp is 9000. The result of `hasEnded` now is false. Then add a new `phaseInfo` with `endTime` is 8000 and the result of `hasEnded` will be true.

0xhuy0512 : In the code, there's no restriction about time duration when owner is allowed to call `Launchpad.addPhase()`, `Launchpad.setPhase()` and `Launchpad.configLaunchpad()`.

If owner can change configurations of phases or add new phases after allow users to buy token, then users can't trust the project, because they will scared that the vesting roadmap can be changed anytime.

Kong7ych3 : In the Launchpad contract, the configLaunchpad function is used to configure the Launchpad start time and other necessary parameters. However, whether Launchpad has been configured is not checked in the hasStarted and addPhase functions. Therefore, when configLaunchpad has not been configured, startTime is 0, and the owner can perform the addPhase operation. If the startTime set in the future is greater than the phase endTime, it will cause a phase conflict.

infinityhacker : According to the claim flow in `Launchpad` contract, when all phase is ended, owner can call `startClaimingPhase` to start the claiming process. And user who has buy tokens can now claim his token. But after reviewing, I notice that the `addPhase` function didn't check if all phases has ended before adding a new phase, which may break the whole flow of the claim process

Recommendation

Yaodao : Recommend adding the logic to check the new `endTime` to ensure that the `endTime` of the new `phaseInfo` is larger than the front `phaseInfo` and smaller than the after `phaseInfo`.

0xhuy0512 : In `addPhase()`:

```
+       if (block.timestamp >= startTime) reverted;
```

In `setPhase()`:

```
+       if (block.timestamp >= startTime) reverted;
```

In `configLaunchpad()`:

```
+       if (block.timestamp >= startTime && startTime != 0) reverted;
```

Kong7ych3 : It is recommended to check whether startTime is greater than 0 in the hasStarted function, and perform a hasStarted check when performing the addPhase operation.

```
function hasStarted() public view returns (bool) {
    if (startTime > 0) {
        return _currentBlockTimestamp() >= startTime;
    }
}

function addPhase(
    ...
) external onlyOwner {
    if (!hasStarted()) revert SaleNotStarted();
    ...
}
```

infinityhacker : Add `hasEnded` check in `addPhase` function

Client Response

Acknowledged.Mostly Fixed by fixing other vulnerabilities of this report.

MGP-11:No guarantee about sending project token to Launchpad contract

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	0xhuy0512

Code Reference

- code/contracts/launchpad/Launchpad.sol#L284

```
284:function configLaunchpad(
```

Description

0xhuy0512 : In the Launchpad contract, there's no guarantee that owner will send project token that match or greater than `max_launch_tokens_to_distribute`, hence investigators will doubt about the legitimate of the launchpad

Recommendation

0xhuy0512 :

```
    function configLlaunchpad(
        address _projectToken,
        address _saleToken,
        address _vestingContract,
        address _treasury,
        uint256 _startTime,
        uint256 _maxToDistribute,
        uint256 _maxToRaise,
        uint256 _lowFDVVestingPart,
        uint256 _highFDVVestingPart,
        uint256 _projectTokenDecimal,
        uint256 _saleokenDecimal
    ) public onlyOwner {
        if (_treasury == address(0)) revert ZeroAddress();
        if (_maxToDistribute == 0) revert InvalidAmount();
        if (_maxToRaise == 0) revert InvalidAmount();

        projectToken = IERC20(_projectToken);
        saleToken = IERC20(_saleToken);
        vestingContract = ILaunchpadVesting(_vestingContract);
        startTime = _startTime;
        treasury = _treasury;
-       max_launch_tokens_to_distribute = _maxToDistribute;
        maxRaiseAmount = _maxToRaise;
        LOW_FDV_VESTING_PART = _lowFDVVestingPart;
        HIGH_FDV_VESTING_PART = _highFDVVestingPart;

        projectTokenDecimal = _projectTokenDecimal;
        saleTokenDecimal = _saleokenDecimal;

+       if (_maxToDistribute > max_launch_tokens_to_distribute) {
+           projectToken.safeTransferFrom(
+               msg.sender,
+               address(this),
+               _maxToDistribute - max_launch_tokens_to_distribute
+           );
+       }
+       max_launch_tokens_to_distribute = _maxToDistribute;
    }
```

Client Response

Fixed. Added push strategy instead of pull. So now when the user will do configLaunchpad at that time the contract will pull required project tokens, which should be approved by the owner before calling configLaunchpad.

MGP-12:Unnecessary decimals input in `Launchpad.configLaunchpad()` function can lead to very bad result

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	0xhuy0512

Code Reference

- code/contracts/launchpad/Launchpad.sol#L284

```
284:function configLaunchpad(
```

Description

0xhuy0512 : Inputs in `Launchpad.configLaunchpad()` function included `_projectTokenDecimal` and `_saleokenDecimal` which unnecessary because we can easily deduce using inputs `_projectToken` and `_saleToken`
If the input `_projectTokenDecimal` and `_saleokenDecimal` got wrong, the calculation of `_tokenAllocBySale()` will be higher/lower than expected leading to users can get more/less project token than expected

Recommendation

0xhuy0512 :

```
+   import { IERC20Metadata } from "@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.so
l";

    ...

    function configLaunchpad(
        address _projectToken,
        address _saleToken,
        address _vestingContract,
        address _treasury,
        uint256 _startTime,
        uint256 _maxToDistribute,
        uint256 _maxToRaise,
        uint256 _lowFDVVestingPart,
        uint256 _highFDVVestingPart,
        uint256 _projectTokenDecimal,
        uint256 _saleokenDecimal
    ) public onlyOwner {
        if (_treasury == address(0)) revert ZeroAddress();
        if (_maxToDistribute == 0) revert InvalidAmount();
        if (_maxToRaise == 0) revert InvalidAmount();

        projectToken = IERC20(_projectToken);
        saleToken = IERC20(_saleToken);
        vestingContract = ILaunchpadVesting(_vestingContract);
        startTime = _startTime;
        treasury = _treasury;
        max_launch_tokens_to_distribute = _maxToDistribute;
        maxRaiseAmount = _maxToRaise;
        LOW_FDV_VESTING_PART = _lowFDVVestingPart;
        HIGH_FDV_VESTING_PART = _highFDVVestingPart;

-       projectTokenDecimal = _projectTokenDecimal;
-       saleTokenDecimal = _saleokenDecimal;

+       projectTokenDecimal = IERC20Metadata(projectToken).decimals();
+       saleTokenDecimal = IERC20Metadata(saleToken).decimals();
    }
```

Client Response

Fixed. Already fixed in audit 2.0

MGP-13:Raised tokens will not be sent to the treasury

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	biakia

Code Reference

- code/contracts/launchpad/Launchpad.sol#L57
- code/contracts/launchpad/Launchpad.sol#L318-L322

```
57:address public treasury; // Address of treasury multisig, it will receive raised amount

318:function emergencyWithdrawFunds(address token, uint256 amount) external whenPaused onlyOwner {
319:         IERC20(token).safeTransfer(owner(), amount);
320:
321:         emit EmergencyWithdraw(token, amount);
322:     }
```

Description

biakia :

```
address public treasury; // Address of treasury multisig, it will receive raised amount
```

The comment of the above code says that the `treasury` is a multisig address and the raised tokens will be sent to the `treasury`. However, there is no logic to send the raised tokens to the `treasury`. The only function to withdraw the raised tokens is `emergencyWithdrawFunds`:

```
/// @dev Emergency Withdraw for Failsafe
    function emergencyWithdrawFunds(address token, uint256 amount) external whenPaused onlyOwner {
        IERC20(token).safeTransfer(owner(), amount);

        emit EmergencyWithdraw(token, amount);
    }
```

All raised tokens will be sent to the owner instead of the multisig address.

Recommendation

biakia : Consider providing a function to withdraw raised tokens to the `treasury`.

Client Response

Fixed. Added transferFundsToTreasury so raised funds can be transferred to treasury.

MGP-14:DENOMINATOR constant is too small, lead to unwanted sale price

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	0xhuy0512

Code Reference

- code/contracts/launchpad/Launchpad.sol#L34
- code/contracts/launchpad/Launchpad.sol#L38
- code/contracts/launchpad/Launchpad.sol#L389-L390

```
34:uint256 priorityMultiplier; // > 0 for priority sale, = 0 for public sale in DENOMINATOR

38:uint256 public constant DENOMINATOR = 10000;

389:uint256 numerator = _saleTokenAmount * phaseInfo.tokenPerSaleToken * projectTokenDecimal;
390:        uint256 denominator = DENOMINATOR * saleTokenDecimal;
```

Description

0xhuy0512 : `DENOMINATOR` in Launchpad contract is equals to 10000 and can't be changed. In struct `PhaseInfo`, `tokenPerSaleToken` is project token per sale token in DENOMINATOR In common launchpad project, sale token usually is USDC, ETH , BNB,etc... And project token price is vary from very small to 2$ per token, most token is in the range of ~0.1$ per token. Because `DENOMINATOR` is so small, admin can't be comfortable adding the sale price.

Let's say we want to sell project token in this phase with the price is 0.1$ and sale project token is ETH. ETH is 2000$ per token. But admin can't sell this with 0.1$ price, because the minimum that admin can sell is 0.2$ (tokenPerSaleToken = 1 -> project token price = 0.2$)

Recommendation

0xhuy0512 : Change `DENOMINATOR` to `1e18` for more flexible

Client Response

Fixed. updated DENOMINATOR value to 1e18.

MGP-15:Allow owner to call `renounceOwnership()` can lead the contracts to ownerless

Category	Severity	Client Response	Contributor
Privilege Related	Low	Mitigated	0xhuy0512

Code Reference

- code/contracts/launchpad/Launchpad.sol#L16

```
16:OwnableUpgradeable,
```

Description

0xhuy0512 : In both `Launchpad` and `LaunchpadVesting` contract, owner can still call `renounceOwnership()` which will make the contract ownerless, causing DOS all the function that have `onlyOwner` modifier

Recommendation

0xhuy0512 : Add this in both `Launchpad` and `LaunchpadVesting` contract:

```
    function renounceOwnership() public override onlyOwner  {
        revert();
    }
```

Client Response

Mitigated.For every project we are already using multisig as Owner so this one is already resolved.

MGP-16:Miss 0 amount check

Category	Severity	Client Response	Contributor
Code Style	Low	Fixed	NoodleDonn212

Code Reference

- code/contracts/launchpad/LaunchpadVesting.sol#L137
- code/contracts/launchpad/LaunchpadVesting.sol#L152

```
137:function setLaunchpad(address _newAddress) external onlyOwner {

152:function configLaunchpadVesting(
```

Description

NoodleDonn212 : The token vesting contract does not verify that the LOW_FDV_VESTING_DURATION and HIGH_FDV_VESTING_DURATION are greater than zero when they are set. These variables are used as denominators in calculations, which could lead to division by zero errors.

The configLaunchpadVesting and configpadVesting functions in the contract allow the LOW_FDV_VESTING_DURATION and HIGH_FDV_VESTING_DURATION to be set to any value, including zero. These variables are used in the getClaimable function to calculate the amount of tokens that can be claimed. If either of these variables is set to zero, it would lead to a division by zero error when getClaimable is called, which would cause the transaction to fail.

```
function configLaunchpadVesting(
        address _projectToken,
        uint256 _lowFDVVestingDuration,
        uint256 _highFDVVestingDuration
    ) external onlyOwner {
        if (address(_projectToken) == address(0)) revert AddressZero();
        projectToken = IERC20(_projectToken);

        LOW_FDV_VESTING_DURATION = _lowFDVVestingDuration;
        HIGH_FDV_VESTING_DURATION = _highFDVVestingDuration;
    }

    function configpadVesting(
    address _projectToken,
    uint256 _lowFDVVestingDuration,
    uint256 _highFDVVestingDuration
) external onlyOwner {
    require(vestingStartTime == 0, "Vesting has already started");
    if (address(_projectToken) == address(0)) revert AddressZero();
    projectToken = IERC20(_projectToken);

    LOW_FDV_VESTING_DURATION = _lowFDVVestingDuration;
    HIGH_FDV_VESTING_DURATION = _highFDVVestingDuration;
}
```

Recommendation

NoodleDonn212 : It is recommended to add checks in the configLaunchpadVesting and configpadVesting functions to ensure that LOW_FDV_VESTING_DURATION and HIGH_FDV_VESTING_DURATION are greater than zero. If you rely on Solidity's automatic check, the transaction will simply revert with a generic "division by zero" error message. If you add your own check, you can provide a more specific message like "Vesting duration must be > 0".

So, even though Solidity 0.8.0 provides automatic checks for division by zero, it's still a good idea to add your own checks

```
function configLaunchpadVesting(
        address _projectToken,
        uint256 _lowFDVVestingDuration,
        uint256 _highFDVVestingDuration
    ) external onlyOwner {
        if (address(_projectToken) == address(0)) revert AddressZero();
        require(_lowFDVVestingDuration > 0, "Low FDV vesting duration must be > 0");
        require(_highFDVVestingDuration > 0, "High FDV vesting duration must be > 0");

        projectToken = IERC20(_projectToken);

        LOW_FDV_VESTING_DURATION = _lowFDVVestingDuration;
        HIGH_FDV_VESTING_DURATION = _highFDVVestingDuration;
    }

    function configpadVesting(
    address _projectToken,
    uint256 _lowFDVVestingDuration,
    uint256 _highFDVVestingDuration
) external onlyOwner {
    require(_lowFDVVestingDuration > 0, "Low FDV vesting duration must be > 0");
    require(_highFDVVestingDuration > 0, "High FDV vesting duration must be > 0");
    if (address(_projectToken) == address(0)) revert AddressZero();
    projectToken = IERC20(_projectToken);

    LOW_FDV_VESTING_DURATION = _lowFDVVestingDuration;
    HIGH_FDV_VESTING_DURATION = _highFDVVestingDuration;
}
```

Client Response

Fixed. Added non-zero checks for FDV_VESING_DURATION's. So that Divide by Zero will not occur now.

MGP-17:Set time variable should more than `block.timestamp`

Category	Severity	Client Response	Contributor
Code Style	Low	Fixed	NoodleDonn212

Code Reference

- code/contracts/launchpad/LaunchpadVesting.sol#L152
- code/contracts/launchpad/Launchpad.sol#L246
- code/contracts/launchpad/Launchpad.sol#L265
- code/contracts/launchpad/Launchpad.sol#L284

```
152:function configLaunchpadVesting(

246:function addPhase(

265:function setPhase(

284:function configLaunchpad(
```

Description

NoodleDonn212 : The addPhase() and setPhase() functions are used to add a new phase or set an existing phase in the Launchpad contract. These functions accept an endTime parameter, which is intended to represent the timestamp when the phase ends. However, there are currently no checks in place to ensure that this timestamp is in the future. As a result, it is possible for an admin to set the endTime to a timestamp in the past.

Impact: If the endTime is set to a timestamp in the past, the phase would end immediately as it is added or set. This could disrupt the intended flow of the contract and potentially prevent users from participating in the phase. It could also lead to confusion and a loss of trust among users.

```
function addPhase(
        uint256 endTime,
        uint256 saleCap,
        uint256 tokenPerSaleToken,
        uint256 priorityMultiplier,
        bool isLofFDV
    ) external onlyOwner {
        PhaseInfo memory newPhase = PhaseInfo({
            endTime: endTime,
            saleCap: saleCap,
            allocatedAmount: 0,
            tokenPerSaleToken: tokenPerSaleToken,
            priorityMultiplier: priorityMultiplier,
            isLofFDV: isLofFDV
        });

        phaseInfos.push(newPhase);
    }

    function setPhase(
        uint256 index,
        uint256 endTime,
        uint256 saleCap,
        uint256 tokenPerSaleToken,
        uint256 priorityMultiplier,
        bool isLofFDV
    ) external onlyOwner {
        if (index > phaseInfos.length) revert InvalidPhase();

        PhaseInfo storage phase = phaseInfos[index];

        phase.endTime = endTime;
        phase.saleCap = saleCap;
        phase.tokenPerSaleToken = tokenPerSaleToken;
        phase.priorityMultiplier = priorityMultiplier;
        phase.isLofFDV = isLofFDV;
    }
```

NoodleDonn212 : The Vesting contract does not include a check to prevent the modification of the vesting start time after it has been set. This omission could potentially disrupt the vesting schedule, allow for manipulation of token release,.

configLaunchpadVesting() Method is missing check to ensure that the vesting start time, once set, cannot be changed.

```
function configLaunchpadVesting(
        address _projectToken,
        uint256 _lowFDVVestingDuration,
        uint256 _highFDVVestingDuration
    ) external onlyOwner {
        if (address(_projectToken) == address(0)) revert AddressZero();
        projectToken = IERC20(_projectToken);

        LOW_FDV_VESTING_DURATION = _lowFDVVestingDuration;
        HIGH_FDV_VESTING_DURATION = _highFDVVestingDuration;
    }
```

Impact: Without this check, the vesting start time could be modified after vesting has begun, disrupting the vesting schedule and potentially delaying or accelerating the release of tokens.

A disrupted vesting schedule could lead to participants receiving their tokens earlier or later than expected. Furthermore, a malicious contract owner could continually reset the vesting start time, effectively locking participants' tokens indefinitely.

NoodleDonn212 : The configLaunchpad() function is used to configure the Launchpad contract. This function accepts a startTime parameter, which is intended to represent the timestamp when the sale starts. However, there are currently no checks in place to ensure that this timestamp is in the future. As a result, it is possible for an admin to set the startTime to a timestamp in the past.

Impact: If the startTime is set to a timestamp in the past, the sale would start immediately or even retroactively as it is configured. This could disrupt the intended flow of the contract and potentially prevent users from participating in the sale.

```
function configLaunchpad(
        address _projectToken,
        address _saleToken,
        address _vestingContract,
        address _treasury,
        uint256 _startTime,
        uint256 _maxToDistribute,
        uint256 _maxToRaise,
        uint256 _lowFDVVestingPart,
        uint256 _highFDVVestingPart,
        uint256 _projectTokenDecimal,
        uint256 _saleokenDecimal
    ) public onlyOwner {
        if (_treasury == address(0)) revert ZeroAddress();
        if (_maxToDistribute == 0) revert InvalidAmount();
        if (_maxToRaise == 0) revert InvalidAmount();

        projectToken = IERC20(_projectToken);
        saleToken = IERC20(_saleToken);
        vestingContract = ILaunchpadVesting(_vestingContract);
        startTime = _startTime;
        treasury = _treasury;
        max_launch_tokens_to_distribute = _maxToDistribute;
        maxRaiseAmount = _maxToRaise;
        LOW_FDV_VESTING_PART = _lowFDVVestingPart;
        HIGH_FDV_VESTING_PART = _highFDVVestingPart;

        projectTokenDecimal = _projectTokenDecimal;
        saleTokenDecimal = _saleokenDecimal;
    }
```

Recommendation

NoodleDonn212 : To mitigate this issue, it is recommended to add a check in the addPhase() and setPhase() functions to ensure that the endTime is in the future. Could be done using a require() statement.

```
function addPhase(
        uint256 endTime,
        uint256 saleCap,
        uint256 tokenPerSaleToken,
        uint256 priorityMultiplier,
        bool isLofFDV
    ) external onlyOwner {
        require(endTime > block.timestamp, "endTime must be in the future");
        PhaseInfo memory newPhase = PhaseInfo({
            endTime: endTime,
            saleCap: saleCap,
            allocatedAmount: 0,
            tokenPerSaleToken: tokenPerSaleToken,
            priorityMultiplier: priorityMultiplier,
            isLofFDV: isLofFDV
        });

        phaseInfos.push(newPhase);
    }

    function setPhase(
        uint256 index,
        uint256 endTime,
        uint256 saleCap,
        uint256 tokenPerSaleToken,
        uint256 priorityMultiplier,
        bool isLofFDV
    ) external onlyOwner {
        require(endTime > block.timestamp, "endTime must be in the future");
        if (index > phaseInfos.length) revert InvalidPhase();


        PhaseInfo storage phase = phaseInfos[index];

        phase.endTime = endTime;
        phase.saleCap = saleCap;
        phase.tokenPerSaleToken = tokenPerSaleToken;
        phase.priorityMultiplier = priorityMultiplier;
        phase.isLofFDV = isLofFDV;
    }
```

NoodleDonn212 : To mitigate this vulnerability, it is recommended to implement a check to prevent the modification of the vesting start time after it has been set.

```
function configpadVesting(
    address _projectToken,
    uint256 _lowFDVVestingDuration,
    uint256 _highFDVVestingDuration
) external onlyOwner {
    require(vestingStartTime == 0, "Vesting has already started");
    if (address(_projectToken) == address(0)) revert AddressZero();
    projectToken = IERC20(_projectToken);

    LOW_FDV_VESTING_DURATION = _lowFDVVestingDuration;
    HIGH_FDV_VESTING_DURATION = _highFDVVestingDuration;
}
```

NoodleDonn212 : To mitigate this issue, it is recommended to add a check in the configLaunchpad() function to ensure that the startTime is in the future. Could be done using a require() statement.

```
function configLaunchpad(
        address _projectToken,
        address _saleToken,
        address _vestingContract,
        address _treasury,
        uint256 _startTime,
        uint256 _maxToDistribute,
        uint256 _maxToRaise,
        uint256 _lowFDVVestingPart,
        uint256 _highFDVVestingPart,
        uint256 _projectTokenDecimal,
        uint256 _saleokenDecimal
    ) public onlyOwner {
        if (_treasury == address(0)) revert ZeroAddress();
        if (_maxToDistribute == 0) revert InvalidAmount();
        if (_maxToRaise == 0) revert InvalidAmount();
        require(_startTime > block.timestamp, "startTime must be in the future");

        projectToken = IERC20(_projectToken);
        saleToken = IERC20(_saleToken);
        vestingContract = ILaunchpadVesting(_vestingContract);
        startTime = _startTime;
        treasury = _treasury;
        max_launch_tokens_to_distribute = _maxToDistribute;
        maxRaiseAmount = _maxToRaise;
        LOW_FDV_VESTING_PART = _lowFDVVestingPart;
        HIGH_FDV_VESTING_PART = _highFDVVestingPart;

        projectTokenDecimal = _projectTokenDecimal;
        saleTokenDecimal = _saleokenDecimal;
    }
```

Client Response

Fixed. Added proper checks for startTime, endTime, vestingStartTime in Launchpad & LaunchpadVesting contract.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Invoices, or the scope of services, and terms and conditions provided to you (“Customer” or the “Company”) in connection with the Invoice. This report provided in connection with the services set forth in the Invoices shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Invoice. 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 Secure3’s prior written consent in each instance.

This report is not an “endorsement” or “disapproval” of any particular project or team. This report is not an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Secure3 to perform a security assessment. This report does not provide any warranty or guarantee of free of bug of codes analyzed, nor do they provide any indication of the technologies, business model or legal compliancy.

This report should not be used in any way to make decisions around investment or involvement with any particular project. Instead, it represents an extensive assessing process intending to help our customers increase the quality of their code and high-level consistency of implementation and business model, while reducing the risk presented by cryptographic tokens and blockchain technology.

Secure3’s position on the final decisions over blockchain technologies and corresponding associated transactions is that each company and individual are responsible for their own due diligence and continuous security.

The assessment services provided by Secure3 is subject to dependencies and under continuing development. 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.