Security Audit Report for LaunchPad Contracts

Date: Sep 14, 2024 **Version:** 1.0

Contact: contact@blocksec.com

Contents

Report Manifest

Item	Description
Client	Magpie
Target	LaunchPad Contracts

Version History

Version	Date	Description
1.0	Sep 14, 2024	First release

Signature

Digitally signed by: BlockSec (https://blocksec.com)

SHA1 digest of public key: BAB106D3708BC6DE0963CE05E9478C24B326AE1C

Date: 2024-09-20 08:47:55+0000

About BlockSec BlockSec focuses on the security of the blockchain ecosystem and collaborates with leading DeFi projects to secure their products. BlockSec is founded by top-notch security researchers and experienced experts from both academia and industry. They have published multiple blockchain security papers in prestigious conferences, reported several zero-day attacks of DeFi applications, and successfully protected digital assets that are worth more than 14 million dollars by blocking multiple attacks. They can be reached at Email, Twitter and Medium.

Chapter 1 Introduction

1.1 About Target Contracts

Information	Description
Type	Smart Contract
Language	Solidity
Approach	Semi-automatic and manual verification

The focus of this audit is on the LaunchPad Contracts[1] of the Magpie. The LaunchpadV2 contract is designed to facilitate token sales for new projects, offering a structured and secure process for both private and public phases. It ensures that token sales are conducted efficiently, with mechanisms in place for price discovery, vesting, and handling unsold quotas.

Please note that the audit scope is limited to the following smart contracts:

```
contracts/launchpad/LaunchPadV2.sol
contracts/launchpad/LaunchpadVestingV2.sol
```

Listing 1.1: Audit Scope for this Report

Other files are not within the scope of the audit. Additionally, all dependencies of the smart contracts within the audit scope are considered reliable in terms of both functionality and security, and are therefore not included in the audit scope.

The auditing process is iterative. Specifically, we would audit the commits that fix the discovered issues. If there are new issues, we will continue this process. The commit SHA values during the audit are shown in the following table. Our audit report is responsible for the code in the initial version (`Version 1`), as well as new code (in the following versions) to fix issues in the audit report.

Project	Version	Commit Hash
LaunchPad Contracts	`Version 1`	`80fc03adcb81501c6f826ee3fac24670e81e9c88`
	`Version 2`	`40d0ecc80107f853da9aaaf0f0910d360bba1161`

This audit report does not constitute investment advice or a personal recommendation. It does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Any entity should not rely on this report in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset.

This audit report is not an endorsement of any particular project or team, and the report does not guarantee the security of any particular project. This audit does not give any warranties on discovering all security issues of the smart contracts, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of smart contracts.

[1] https://github.com/magpiexyz/magpie_contracts

The scope of this audit is limited to the code mentioned in Section 1.1. Unless explicitly specified, the security of the language itself (e.g., the solidity language), the underlying compiling toolchain and the computing infrastructure are out of the scope.

1.2 Procedure of Auditing

We perform the audit according to the following procedure.

- **Vulnerability Detection** We first scan smart contracts with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of smart contracts and conduct further investigation on the possible vulnerabilities using an automatic fuzzing tool (developed by our research team). We also manually analyze possible attack scenarios with independent auditors to cross-check the result.
- **Recommendation** We provide some useful advice to developers from the perspective of good programming practice, including gas optimization, code style, and etc.

We show the main concrete checkpoints in the following.

1.2.1 Software Security

- Reentrancy
- DoS
- Access control
- Data handling and data flow
- Exception handling
- Untrusted external call and control flow
- Initialization consistency
- Events operation
- Error-prone randomness
- Improper use of the proxy system

1.2.2 DeFi Security

- Semantic consistency
- Functionality consistency
- Permission management
- Business logic
- Token operation
- Emergency mechanism
- Oracle security
- Whitelist and blacklist
- Economic impact
- Batch transfer

1.2.3 NFT Security

* Duplicated item
* Verification of the token receiver
* Off-chain metadata security

1.2.4 Additional Recommendation

* Gas optimization
* Code quality and style

Note *The previous checkpoints are the main ones. We may use more checkpoints during the auditing process according to the functionality of the project.*

1.3 Security Model

To evaluate the risk, we follow the standards or suggestions that are widely adopted by both industry and academy, including OWASP Risk Rating Methodology [2] and Common Weakness Enumeration [3]. The overall *severity* of the risk is determined by *likelihood* and *impact*. Specifically, likelihood is used to estimate how likely a particular vulnerability can be uncovered and exploited by an attacker, while impact is used to measure the consequences of a successful exploit.

In this report, both likelihood and impact are categorized into two ratings, i.e., *high* and *low* respectively, and their combinations are shown in Table 1.1.

Table 1.1: Vulnerability Severity Classification

		Likelihood	
		High	*Low*
Impact	*High*	High	Medium
	Low	Medium	Low

Accordingly, the severity measured in this report are classified into three categories: **High**, **Medium**, **Low**. For the sake of completeness, **Undetermined** is also used to cover circumstances when the risk cannot be well determined.

Furthermore, the status of a discovered item will fall into one of the following four categories:

- **Undetermined** No response yet.
- **Acknowledged** The item has been received by the client, but not confirmed yet.

[2] https://owasp.org/www-community/OWASP_Risk_Rating_Methodology

[3] https://cwe.mitre.org/

- **Confirmed** The item has been recognized by the client, but not fixed yet.
- **Fixed** The item has been confirmed and fixed by the client.

Chapter 2 Findings

In total, we found **ten** potential security issues. Besides, we have **two** recommendations and **one** note.

- High Risk: 0
- Medium Risk: 3
- Low Risk: 7
- Recommendation: 2
- Note: 1

ID	Severity	Description	Category	Status
1	Low	Incorrect rounding direction of fee calculation in function `cancelOrder()`	Software Security	Fixed
2	Low	Potential precision loss may prevent users from properly claiming	Software Security	Fixed
3	Medium	Lack of updating `publicPhase.tokenPerSaleToken` in function `setPublicPhaseSaleCap()`	DeFi Security	Confirmed
4	Medium	Lack of check in function `hasStarted()`	DeFi Security	Fixed
5	Low	Potential failure of cancellation due to incorrect check in function `transferFundsToTreasury()`	DeFi Security	Fixed
6	Low	Incorrect calculation in the function `quotePrice()`	DeFi Security	Fixed
7	Low	Lack of check for user's remaining `publicPhaseDeposits` in function `cancelOrder()`	DeFi Security	Fixed
8	Low	Lack of check in function `setPublicPhaseSaleCap()`	DeFi Security	Fixed
9	Medium	Lack of check on `_publicPhaseWithdrawalDuration` in function `configLaunchpad()`	DeFi Security	Comfirmed
10	Low	Lack of check on `publicPhase.endTime` in function `startClaimingPhase()`	DeFi Security	Fixed
11	-	Lack of check on `_startTime` in function `setPhase()`	Recommendation	Fixed
12	-	Incorrect check in function `getCurrentPhaseInfo()`	Recommendation	Fixed
13	-	Potential centralization risks	Note	-

The details are provided in the following sections.

2.1 Software Security

2.1.1 Incorrect rounding direction of fee calculation in function `cancelOrder()`

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the `cancelOrder()` function, a `cancellationFee` is charged, and in the formula for calculating the fee at line 320, rounding down is used. Users can bypass the fee collection with a rather small amount. The fee calculation should instead be rounded up.

```
306   function cancelOrder(uint256 _amountToRefund) external whenNotPaused isSaleActive nonReentrant
          {
307     if (_amountToRefund == 0) revert InvalidAmount();
308
309     (bool isPrivatePhase, ) = getCurrentPhaseInfo();
310     if (isPrivatePhase) revert InvalidPhase();
311
312     uint256 withdrawalPeriodEnd = publicPhase.startTime + publicPhaseWithdrawalDuration;
313
314     if (block.timestamp > withdrawalPeriodEnd) revert PublicPhaseWithdrawalPeriodOver();
315
316     UserInfo storage user = userInfo[msg.sender];
317     if (_amountToRefund > user.publicPhaseDeposits) revert WithdrawExceedsDeposit();
318
319     //charge a fee on cancellation
320     uint256 fee = (_amountToRefund * cancellationFee) / DENOMINATOR;
321     uint256 amountAfterFee = _amountToRefund - fee;
322     accumulatedFees += fee;
323
324     user.publicPhaseDeposits -= _amountToRefund;
325     totalRaised -= _amountToRefund;
326     publicPhase.saleTokenDeposits -= _amountToRefund;
327     _rebalanceAndUpdate();
328     IERC20(saleToken).safeTransfer(msg.sender, amountAfterFee);
329     emit OrderCancelled(msg.sender, _amountToRefund, fee);
330   }
```

Listing 2.1: contracts/launchpad/LaunchPadV2.sol

Impact Users can bypass the `cancellationFee` collection.

Suggestion Use rounding up in the fee calculation.

2.1.2 Potential precision loss may prevent users from properly claiming

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the `_checkValidCapAndUpdate()` function, `allocatedInPrivatePhase` is used to record the number of project tokens allocated to the user during the private phase. In the calculation of `publicPhase.tokenPerSaleToken`, `publicPhase.saleCap` is reduced by `allocatedInPrivatePhase` to calculate the tokenPerSaleToken.

If a user purchases project tokens multiple times during the private phase, `allocatedInPrivatePhase` may record a value lower than the total amount the user ultimately claims due to precision loss accumulation(e.g., 3/2 + 3/2 = 2 while (3+3)/2 = 3). This results in the calculated `publicPhase.tokenPerSaleToken` being slightly higher, which can cause the total number of tokens the user can claim during the final claim process to exceed the `publicPhase.saleCap`.

```
function _checkValidCapAndUpdate(uint256 _saleTokenAmount) internal {
   uint256 _toAllocate = _tokenAllocBySale(_saleTokenAmount, privatePhase.tokenPerSaleToken);
   if (_toAllocate == 0) revert ZeroAllocation();


   allocatedInPrivatePhase += _toAllocate;
   if (allocatedInPrivatePhase > privatePhase.saleCap) revert NotEnoughToken();


   uint256 privatePhasePurchased = getUserPurchasedProjectTokens(msg.sender, true);
   uint256 _userCap = (userInfo[msg.sender].priorityQuota * privatePhase.priorityMultiplier) /
       DENOMINATOR;
   if (privatePhasePurchased + _toAllocate > _userCap) revert ExceedsUserPriorityCap();
}
```

Listing 2.2: contracts/launchpad/LaunchPadV2.sol

```
function _tokenAllocBySale(
   uint256 _saleTokenAmount,
   uint256 _tokenPerSaleToken
) internal view returns (uint256) {
   uint256 numerator = _saleTokenAmount * _tokenPerSaleToken * 10 ** projectTokenDecimals;
   uint256 denominator = DENOMINATOR * 10 ** saleTokenDecimals;
   return numerator / denominator;
}
```

Listing 2.3: contracts/launchpad/LaunchPadV2.sol

```
function _getRebalancedTokenPerSaleToken(
   uint256 _saleTokenDeposits
) internal view returns (uint256) {
   if (_saleTokenDeposits == 0) {
       return publicPhaseMaxTokenPerSale;
   } else {
       uint256 rebalancedTokenPerSaleToken = ((publicPhase.saleCap - allocatedInPrivatePhase) *
           (10 ** saleTokenDecimals) *
           DENOMINATOR) / (_saleTokenDeposits * (10 ** projectTokenDecimals));
       return
           rebalancedTokenPerSaleToken < publicPhaseMaxTokenPerSale
               ? rebalancedTokenPerSaleToken
               : publicPhaseMaxTokenPerSale;
```

```
    }
}
```

Listing 2.4: contracts/launchpad/LaunchPadV2.sol

Impact The last user may fail to claim due to insufficient token balance.

Suggestion Use rounding up in the `allocatedInPrivatePhase` calculation.

2.2 DeFi Security

2.2.1 Lack of updating `publicPhase.tokenPerSaleToken` in function `setPublicPhaseSaleCap()`

Severity Medium

Status Confirmed

Introduced by `Version 1`

Description When the protocol's `owner` invokes function `setPublicPhaseSaleCap()` to reset the `publicPhase.saleCap`, the function does not invoke function `_rebalanceAndUpdate()` to update the price of the projectToken. This can result in the price becoming outdated, which may lead to losses for users.

```
function setPublicPhaseSaleCap(uint256 _saleCap) external onlyOwner {
    if (privatePhase.saleCap != 0 && _saleCap < privatePhase.saleCap) revert InvalidSaleCap();


    emit PhaseSaleCapUpdated(publicPhase.saleCap, _saleCap);
    publicPhase.saleCap = _saleCap;
}
```

Listing 2.5: contracts/launchpad/LaunchPadV2.sol

Impact The function `setPublicPhaseSaleCap()` does not update the price in a timely manner, which may result in user losses.

Suggestion Timely invoking function `_rebalanceAndUpdate()` after `publicPhase.saleCap` is set to the new value.

Feedback from the Project Correct, but we will only invoke this function only once a few minutes before the public phase starts. After the public phase has started this function will never be invoked.

2.2.2 Lack of check in function `hasStarted()`

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The protocol's `owner` can set `privatePhase` or `publicPhase` using the function `setPhase()`. Specifically, there is no required order for the `owner` to set the `privatePhase` and publicPhase, but the privatePhase must start first. If the owner sets the publicPhase without setting the privatePhase, users can invoke the function buy() to purchase projectToken as the function hasStarted() does not check whether privatePhase.startTime is 0. This allows users to skip the privatePhase and directly enter the publicPhase.

```
function setPhase(
    uint32 _startTime,
    uint32 _endTime,
    uint256 _saleCap,
    uint256 _tokenPerSaleToken,
    uint256 _priorityMultiplier,
    bool _isPrivate,
    uint256 _cliffDuration
) external onlyBeforeSale onlyOwner {
    if (_startTime == 0 || _endTime <= _startTime || _endTime <= block.timestamp)
        revert InvalidTime();
    if (
        _tokenPerSaleToken <= 0 ||
        (_isPrivate &&
            publicPhaseMaxTokenPerSale != 0 &&
            publicPhaseMaxTokenPerSale > _tokenPerSaleToken)
    ) revert InvalidPerSaleAmount();


    PhaseInfo storage phase = _isPrivate ? privatePhase : publicPhase;


    if (
        (_isPrivate && publicPhase.endTime != 0 && _endTime != publicPhase.startTime) ||
        (!_isPrivate && privatePhase.endTime != 0 && _startTime != privatePhase.endTime)
    ) revert InvalidTime();


    if (
        (_isPrivate && publicPhase.saleCap != 0 && _saleCap > publicPhase.saleCap) ||
        (!_isPrivate && privatePhase.saleCap != 0 && _saleCap < privatePhase.saleCap)
    ) revert InvalidSaleCap();


    phase.startTime = _startTime;
    phase.endTime = _endTime;
    phase.saleCap = _saleCap;
    phase.tokenPerSaleToken = _tokenPerSaleToken;
    phase.priorityMultiplier = _priorityMultiplier;
    phase.cliffDuration = _cliffDuration;


    emit PhaseUpdated(
        _startTime,
        _endTime,
        _saleCap,
        _tokenPerSaleToken,
        _priorityMultiplier,
        _cliffDuration
    );
}
```

Listing 2.6: contracts/launchpad/LaunchPadV2.sol

```
function buy(uint256 _amount) external whenNotPaused isSaleActive nonReentrant {
    if (_amount < min_sale_token_amount) {
        revert InvalidAmount();
    }


    (bool isPrivatePhase, ) = getCurrentPhaseInfo();


    PhaseInfo storage phaseInfo = isPrivatePhase ? privatePhase : publicPhase;


    totalRaised += _amount;
    phaseInfo.saleTokenDeposits += _amount;
    UserInfo storage user = userInfo[msg.sender];


    if (isPrivatePhase) {
        _checkValidCapAndUpdate(_amount);
        user.privatePhaseDeposits += _amount;
    } else if (phaseInfo.saleTokenDeposits > publicPhaseDepositCap) {
        revert PublicPhaseDepositCapExceeded();
    } else {
        user.publicPhaseDeposits += _amount;
        _rebalanceAndUpdate();
    }


    IERC20(saleToken).safeTransferFrom(msg.sender, address(this), _amount);
    emit AllocationPurchased(msg.sender, _amount);
}
```

Listing 2.7: contracts/launchpad/LaunchPadV2.sol

```
modifier isSaleActive() {
    if (!hasStarted()) revert SaleNotStarted();
    if (hasEnded()) revert SaleCompleted();
    _;
}
```

Listing 2.8: contracts/launchpad/LaunchPadV2.sol

```
function hasStarted() public view returns (bool) {
    return block.timestamp >= privatePhase.startTime;
}
```

Listing 2.9: contracts/launchpad/LaunchPadV2.sol

Impact The protocol may skip the privatePhase and directly enter the publicPhase.

Suggestion Add a check in the function `hasStarted()` to ensure that `privatePhase.startTime` is not equal to 0.

2.2.3 Potential failure of cancellation due to incorrect check in function `transferFundsToTreasury()`

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description Function `transferFundsToTreasury()` allows the privileged owner to transfer deposited sale tokens to `treasury` when the current `block.timestamp` reaches the specified `withdrawalPeriodEnd`. This includes the case where `block.timestamp` is exactly equal to `withdrawalPeriodEnd`. However, users are also allowed to cancel previous purchases at this moment through the function `cancelOrder()`. In this case, if the owner transfers the sale token out first, the user will not be able to cancel the order, which is against the design.

```
function transferFundsToTreasury(uint256 _amount) external onlyOwner {
    uint256 withdrawalPeriodEnd = publicPhase.startTime + publicPhaseWithdrawalDuration;
    if (block.timestamp >= publicPhase.startTime && block.timestamp < withdrawalPeriodEnd)
        revert TransferNotAllowed();


    if (IERC20(saleToken).balanceOf(address(this)) < _amount) revert InvalidAmount();
    IERC20(saleToken).safeTransfer(treasury, _amount);
    emit TransferredToTreasury(saleToken, _amount);
}
```

Listing 2.10: contracts/launchpad/LaunchPadV2.sol

Impact Users can not cancel at the time of `withdrawalPeriodEnd`.

Suggestion Revise the check in function `transferFundsToTreasury()` to ensure the `owner` can only withdraw after `withdrawalPeriodEnd`.

2.2.4 Incorrect calculation in the function `quotePrice()`

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The function `quotePrice()` returns the price of `projectToken` relative to `saleToken`. Specifically, the function incorrectly assumes that the decimal value of `saleToken` is `1e18` in the calculation. However, the `decimals` of `saleToken` (e.g., USDC) may not actually be `1e18`. This will ultimately result in `quoteToken` returning an incorrect price.

```
function quotePrice(
   uint256 _amount,
   bool _isBuy
) external view whenNotPaused isSaleActive returns (uint256) {
    (bool isPrivatePhase, PhaseInfo memory phaseInfo) = getCurrentPhaseInfo();


    if (_amount < min_sale_token_amount || (!_isBuy && phaseInfo.saleTokenDeposits < _amount)) {
        revert InvalidAmount();
    }


    if (!isPrivatePhase) {
        uint256 rebalancedTokenPerSaleToken = _getRebalancedTokenPerSaleToken(
            _isBuy
                ? phaseInfo.saleTokenDeposits + _amount
                : phaseInfo.saleTokenDeposits - _amount
        );
        phaseInfo.tokenPerSaleToken = rebalancedTokenPerSaleToken;
    }
    return ((DENOMINATOR * 1 ether) / phaseInfo.tokenPerSaleToken);
}
```

Listing 2.11: contracts/launchpad/LaunchPadV2.sol

Impact Function `quotePrice()` will return an incorrect price.

Suggestion Change `(DENOMINATOR * 1 ether) / phaseInfo.tokenPerSaleToken` to `(DENOMINATOR * 10**IERC20(saleToken).decimals()) / phaseInfo.tokenPerSaleToken`.

2.2.5 Lack of check for user's remaining `publicPhaseDeposits` in function `cancelOrder()`

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description Before the end of the `publicPhase`, users can cancel their orders via the function `cancelOrder()`. However, there is no check to ensure that the user's remaining `publicPhaseDeposits` is greater than or equal to `min_sale_token_amount`. Specifically, this is inconsistent with the check in the function `buy()`.

```
function buy(uint256 _amount) external whenNotPaused isSaleActive nonReentrant {
    if (_amount < min_sale_token_amount) {
        revert InvalidAmount();
    }


    (bool isPrivatePhase, ) = getCurrentPhaseInfo();


    PhaseInfo storage phaseInfo = isPrivatePhase ? privatePhase : publicPhase;


    totalRaised += _amount;
    phaseInfo.saleTokenDeposits += _amount;
    UserInfo storage user = userInfo[msg.sender];


    if (isPrivatePhase) {
        _checkValidCapAndUpdate(_amount);
        user.privatePhaseDeposits += _amount;
    } else if (phaseInfo.saleTokenDeposits > publicPhaseDepositCap) {
        revert PublicPhaseDepositCapExceeded();
    } else {
        user.publicPhaseDeposits += _amount;
        _rebalanceAndUpdate();
    }


    IERC20(saleToken).safeTransferFrom(msg.sender, address(this), _amount);
    emit AllocationPurchased(msg.sender, _amount);
}
```

Listing 2.12: contracts/launchpad/LaunchPadV2.sol

```
function cancelOrder(uint256 _amountToRefund) external whenNotPaused isSaleActive nonReentrant
    {
    if (_amountToRefund == 0) revert InvalidAmount();


    (bool isPrivatePhase, ) = getCurrentPhaseInfo();
    if (isPrivatePhase) revert InvalidPhase();


    uint256 withdrawalPeriodEnd = publicPhase.startTime + publicPhaseWithdrawalDuration;


    if (block.timestamp > withdrawalPeriodEnd) revert PublicPhaseWithdrawalPeriodOver();


    UserInfo storage user = userInfo[msg.sender];
    if (_amountToRefund > user.publicPhaseDeposits) revert WithdrawExceedsDeposit();


    //charge a fee on cancellation
    uint256 fee = (_amountToRefund * cancellationFee) / DENOMINATOR;
    uint256 amountAfterFee = _amountToRefund - fee;
    accumulatedFees += fee;


    user.publicPhaseDeposits -= _amountToRefund;
    totalRaised -= _amountToRefund;
    publicPhase.saleTokenDeposits -= _amountToRefund;
    _rebalanceAndUpdate();
    IERC20(saleToken).safeTransfer(msg.sender, amountAfterFee);
    emit OrderCancelled(msg.sender, _amountToRefund, fee);
}
```

Listing 2.13: contracts/launchpad/LaunchPadV2.sol

Impact The user's remaining `publicPhaseDeposits` may be less than `min_sale_token_amount`.

Suggestion Add a check to ensure that the user's remaining `publicPhaseDeposits` is greater than or equal to `min_sale_token_amount`.

2.2.6 Lack of check in function `setPublicPhaseSaleCap()`

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The protocol's `owner` can set the `publicPhase.saleCap` through the function `setPublicPhaseSaleCap()`. Specifically, `publicPhase.saleCap` affects the price of `projectToken` during the `publicPhase`. Thus, the function `setPublicPhaseSaleCap()` can only be invoked before `publicPhase.endTime` to ensure that the price of `projectToken` does not change once the `publicPhase` has ended.

```
  function setPublicPhaseSaleCap(uint256 _saleCap) external onlyOwner {
     if (privatePhase.saleCap != 0 && _saleCap < privatePhase.saleCap) revert InvalidSaleCap();


     emit PhaseSaleCapUpdated(publicPhase.saleCap, _saleCap);
     publicPhase.saleCap = _saleCap;
 }
```

Listing 2.14: contracts/launchpad/LaunchPadV2.sol

Impact The price of `projectToken` may still change after the `publicPhase` ends.

Suggestion Add a check to ensure that `setPublicPhaseSaleCap` can only be invoked before the end of the `publicPhase`.

2.2.7 Lack of check on `_publicPhaseWithdrawalDuration` in function `configLaunchpad()`

Severity Medium

Status Confirmed

Introduced by `Version 1`

Description Users can close their orders via the function `cancelOrder()` before `publicPhase.startTime` + `publicPhaseWithdrawalDuration`. However, after the `publicPhase` begins, a malicious user can deposit a large amount of `saleToken` to reach the `publicPhase` sale cap, preventing other users from making purchases. If the `publicPhase.endTime` is close to or the same as `publicPhase.startTime + publicPhaseWithdrawalDuration`, the malicious user can withdraw part of the `saleToken` in the last allowed time window by canceling orders.

In this case, the malicious user can arbitrarily control the price of the `projectToken`, disrupting the entire token sale process and preventing other users from purchasing normally.

```
function configLaunchpad(
    address _projectToken,
    address _saleToken,
    address _vestingContract,
    address _treasury,
    uint256 _privatePhaseVestingPart,
    uint256 _publicPhaseVestingPart,
    uint256 _minSaleTokenAmount,
    uint32 _publicPhaseWithdrawalDuration,
    uint256 _publicPhaseDepositCap,
    uint256 _publicPhaseMaxTokenPerSale,
    uint256 _cancellationFee
) public onlyBeforeSale onlyOwner {
    if (
        _treasury == address(0) ||
        _projectToken == address(0) ||
        _saleToken == address(0) ||
        _vestingContract == address(0)
    ) revert ZeroAddress();
    if (_privatePhaseVestingPart >= DENOMINATOR || _publicPhaseVestingPart >= DENOMINATOR)
        revert InvalidFDVPart();


    if (
        _publicPhaseMaxTokenPerSale <= 0 ||
        (privatePhase.tokenPerSaleToken != 0 &&
            _publicPhaseMaxTokenPerSale > privatePhase.tokenPerSaleToken)
    ) revert InvalidPerSaleAmount();


    uint8 tempProjectTokenDecimals = IERC20Metadata(_projectToken).decimals();
    uint8 tempSaleTokenDecimals = IERC20Metadata(_saleToken).decimals();
    if (tempSaleTokenDecimals > tempProjectTokenDecimals) revert TokenDecimalExceedsLimit();


    if (_cancellationFee > DENOMINATOR) revert InvalidFeeAmount();


    projectToken = _projectToken;
    saleToken = _saleToken;
    projectTokenDecimals = tempProjectTokenDecimals;
    saleTokenDecimals = tempSaleTokenDecimals;
    vestingContract = ILaunchpadVesting(_vestingContract);
    treasury = _treasury;
    PRIVATE_PHASE_VESTING_PART = _privatePhaseVestingPart;
    PUBLIC_PHASE_VESTING_PART = _publicPhaseVestingPart;
    min_sale_token_amount = _minSaleTokenAmount;
    publicPhaseWithdrawalDuration = _publicPhaseWithdrawalDuration;
    publicPhaseDepositCap = _publicPhaseDepositCap;
    publicPhaseMaxTokenPerSale = _publicPhaseMaxTokenPerSale;
    cancellationFee = _cancellationFee;


    emit LaunchpadConfigured(
        _projectToken,
        _saleToken,
        _vestingContract,
        _treasury,
        _privatePhaseVestingPart,
        _publicPhaseVestingPart,
        _minSaleTokenAmount,
        _publicPhaseWithdrawalDuration,
        _publicPhaseDepositCap,
        _publicPhaseMaxTokenPerSale,
        _cancellationFee
    );
}
```

Listing 2.15: contracts/launchpad/LaunchPadV2.sol

```
function cancelOrder(uint256 _amountToRefund) external whenNotPaused isSaleActive nonReentrant
    {
    if (_amountToRefund == 0) revert InvalidAmount();


    (bool isPrivatePhase, ) = getCurrentPhaseInfo();
    if (isPrivatePhase) revert InvalidPhase();


    uint256 withdrawalPeriodEnd = publicPhase.startTime + publicPhaseWithdrawalDuration;


    if (block.timestamp > withdrawalPeriodEnd) revert PublicPhaseWithdrawalPeriodOver();


    UserInfo storage user = userInfo[msg.sender];
    if (_amountToRefund > user.publicPhaseDeposits) revert WithdrawExceedsDeposit();


    //charge a fee on cancellation
    uint256 fee = (_amountToRefund * cancellationFee) / DENOMINATOR;
    uint256 amountAfterFee = _amountToRefund - fee;
    accumulatedFees += fee;


    user.publicPhaseDeposits -= _amountToRefund;
    totalRaised -= _amountToRefund;
    publicPhase.saleTokenDeposits -= _amountToRefund;
    _rebalanceAndUpdate();
    IERC20(saleToken).safeTransfer(msg.sender, amountAfterFee);
    emit OrderCancelled(msg.sender, _amountToRefund, fee);
}
```

Listing 2.16: contracts/launchpad/LaunchPadV2.sol

Impact A malicious user may manipulate the price of the `projectToken`, affecting the ability of other users to make purchases.

Suggestion Add a check to ensure there is sufficient time between the time at last allowable cancel order and `publicPhase.endTime`.

Feedback from the Project Noted, we will take care of this while doing configurations

2.2.8 Lack of check on `publicPhase.endTime` in function `startClaimingPhase()`

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The function `startClaimingPhase()` is used to start the claiming phase after the sales have ended. However, the check uses `!hasEnded()` to determine if the sales have ended, which is incorrect.

Specifically, the `hasEnded()` function only checks whether the current `block.timestamp` is greater than the end time of the `public phase` while ignoring the possibility that the public phase may not have been set (endTime = 0), which is incorrect.

```
/// @dev Start Tokens Claiming Phase
function startClaimingPhase() external onlyOwner {
    if (!hasEnded()) revert SaleNotCompleted();
    if (canClaimTokens) revert ClaimingPhaseAlreadyStarted();


    canClaimTokens = true;
    vestingContract.setPrivatePhaseVestingStartTime(
        block.timestamp + privatePhase.cliffDuration
    );
    vestingContract.setPublicPhaseVestingStartTime(block.timestamp + publicPhase.cliffDuration)
        ;
    emit ClaimingPhaseStarted(block.timestamp);
}
```

Listing 2.17: contracts/launchpad/LaunchPadV2.sol

```
/// @dev Returns whether the sale has already ended
function hasEnded() public view returns (bool) {
    return publicPhase.endTime <= block.timestamp;
}
```

Listing 2.18: contracts/launchpad/LaunchPadV2.sol

Impact The claiming phase may be triggered at a time that does not align with the intended design, unexpectedly setting the start time for the vesting period.

Suggestion Add a check to ensure `publicPhase.endTime` is not equal to 0.

2.3 Additional Recommendation

2.3.1 Lack of check on `_startTime` in function `setPhase()`

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the function `setPhase()`, there is no check to ensure `privatePhase.startTime > block.timestamp`.

```
function setPhase(
    uint32 _startTime,
    uint32 _endTime,
    uint256 _saleCap,
    uint256 _tokenPerSaleToken,
    uint256 _priorityMultiplier,
    bool _isPrivate,
    uint256 _cliffDuration
) external onlyBeforeSale onlyOwner {
    if (_startTime == 0 || _endTime <= _startTime || _endTime <= block.timestamp)
        revert InvalidTime();
    if (
        _tokenPerSaleToken <= 0 ||
        (_isPrivate &&
            publicPhaseMaxTokenPerSale != 0 &&
            publicPhaseMaxTokenPerSale > _tokenPerSaleToken)
    ) revert InvalidPerSaleAmount();


    PhaseInfo storage phase = _isPrivate ? privatePhase : publicPhase;


    if (
        (_isPrivate && publicPhase.endTime != 0 && _endTime != publicPhase.startTime) ||
        (!_isPrivate && privatePhase.endTime != 0 && _startTime != privatePhase.endTime)
    ) revert InvalidTime();


    if (
        (_isPrivate && publicPhase.saleCap != 0 && _saleCap > publicPhase.saleCap) ||
        (!_isPrivate && privatePhase.saleCap != 0 && _saleCap < privatePhase.saleCap)
    ) revert InvalidSaleCap();


    phase.startTime = _startTime;
    phase.endTime = _endTime;
    phase.saleCap = _saleCap;
    phase.tokenPerSaleToken = _tokenPerSaleToken;
    phase.priorityMultiplier = _priorityMultiplier;
    phase.cliffDuration = _cliffDuration;


    emit PhaseUpdated(
        _startTime,
        _endTime,
        _saleCap,
        _tokenPerSaleToken,
        _priorityMultiplier,
        _cliffDuration
    );
}
```

Listing 2.19: contracts/launchpad/LaunchPadV2.sol

Suggestion Add relevant checks to ensure `privatePhase.startTime > block.timestamp`.

2.3.2 Incorrect check in function `getCurrentPhaseInfo()`

Status Confirmed

Introduced by `Version 1`

Description The view function `getCurrentPhaseInfo()` returns information about the current phase based on the current `block.timestamp`. When `block.timestamp` is exactly equal to `publicPhase.endTime`, this function considers the public phase is ongoing. However, the `hasEnded()` in the modifier `isSaleActive()` considers both sales, including the `public sale`, are inactive, which is inconsistent.

```
/// @dev Check whether the sale is currently active
/// Will be marked as inactive if PROJECT_TOKEN has not been deposited into the contract
modifier isSaleActive() {
    if (!hasStarted()) revert SaleNotStarted();
    if (hasEnded()) revert SaleCompleted();
    _;
}
```

Listing 2.20: contracts/launchpad/LaunchPadV2.sol

```
/// @dev Returns whether the sale has already ended
function hasEnded() public view returns (bool) {
    return publicPhase.endTime <= block.timestamp;
}
```

Listing 2.21: contracts/launchpad/LaunchPadV2.sol

```
/// @dev Returns current running phase info.
function getCurrentPhaseInfo()
public
view
returns (bool isPrivatePhase, PhaseInfo memory phaseInfo)
{
    uint256 currentBlockTimestamp = block.timestamp;


    if (
    currentBlockTimestamp < privatePhase.startTime ||
    currentBlockTimestamp > publicPhase.endTime
    ) {
    return (false, phaseInfo); // not started
    }


    if (currentBlockTimestamp < privatePhase.endTime) {
    return (true, privatePhase);
    }
    return (false, publicPhase);
}
```

Listing 2.22: contracts/launchpad/LaunchPadV2.sol

Suggestion Revise the check to ensure that when `block.timestamp` is equal to `publicPhase.endTime`, the function returns the `phaseInfo` of the `public phase`.

2.4 Note

2.4.1 Potential centralization risks

Introduced by `Version 1`

Description The protocol includes several privileged functions, such as function `emergencyWithdrawFunds()`, `setCancellationFee()`. If the `owner`'s private key is lost or maliciously exploited, it could potentially cause losses to users.

BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS