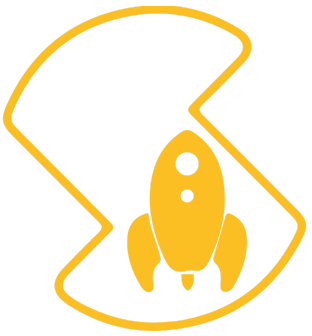
SuperLauncher V.2

Smart Contract Audit Report
Prepared for SuperLauncher



Date Issued: Dec 24, 2021
Project ID: AUDIT2021040
Version: v1.0
Confidentiality Level: Public

inspeX
CYBERSECURITY PROFESSIONAL SERVICE

Report Information

Project ID	AUDIT2021040
Version	v1.0
Client	SuperLauncher
Project	SuperLauncher V.2
Auditor(s)	Suvicha Buakhom Peeraphut Punsuwan Puttimet Thammasaeng
Author	Puttimet Thammasaeng
Reviewer	Weerawat Pawanawiwat
Confidentiality Level	Public

Version History

Version	Date	Description	Author(s)
1.0	Dec 24, 2021	Full report	Puttimet Thammasaeng

Contact Information

Company	Inspex
Phone	(+66) 90 888 7186
Telegram	t.me/inspexco
Email	audit@inspex.co

Table of Contents

1. Executive Summary

As requested by SuperLauncher, Inspex team conducted an audit to verify the security posture of the SuperLauncher V.2 smart contracts between Dec 7, 2021 and Dec 13, 2021. During the audit, Inspex team examined all smart contracts and the overall operation within the scope to understand the overview of SuperLauncher V.2 smart contracts. Static code analysis, dynamic analysis, and manual review were done in conjunction to identify smart contract vulnerabilities together with technical & business logic flaws that may be exposed to the potential risk of the platform and the ecosystem. Practical recommendations are provided according to each vulnerability found and should be followed to remediate the issue.

1.1. Audit Result

In the initial audit, Inspex found 1 critical, 3 low, 2 very low, and 2 info-severity issues. With the project team's prompt response in resolving the issues found by Inspex, all issues were resolved in the reassessment. Therefore, Inspex trusts that SuperLauncher V.2 smart contracts have high-level protections in place to be safe from most attacks.

1.2. Disclaimer

This security audit is not produced to supplant any other type of assessment and does not guarantee the discovery of all security vulnerabilities within the scope of the assessment. However, we warrant that this audit is conducted with goodwill, professional approach, and competence. Since an assessment from one single party cannot be confirmed to cover all possible issues within the smart contract(s), Inspex suggests conducting multiple independent assessments to minimize the risks. Lastly, nothing contained in this audit report should be considered as investment advice.

2. Project Overview

2.1. Project Introduction

SuperLauncher is an investment DAO that funds and collaborates with projects that are shaping the future of the blockchain. SuperLauncher facilitates mass participation in Seed/Private/IDO rounds through its smart contract architecture and offers a feature-rich platform that powers flexible and decentralized management of capital.

SuperLauncher V.2 is a new version of the core feature that includes new features e.g., flexible currency support and support for seed/private rounds. In SuperLauncher V.2, users can participate in IDO investment campaigns with various options in joining each campaign.

Scope Information:

Project Name	SuperLauncher V.2
Website	https://superlauncher.io/
Smart Contract Type	Ethereum Smart Contract
Chain	Binance Smart Chain
Programming Language	Solidity

Audit Information:

Audit Method	Whitebox
Audit Date	Dec 7, 2021 - Dec 13, 2021
Reassessment Date	Dec 23, 2021

The audit method can be categorized into two types depending on the assessment targets provided:

1. **Whitebox**: The complete source code of the smart contracts are provided for the assessment.
2. **Blackbox**: Only the bytecodes of the smart contracts are provided for the assessment.

2.2. Scope

The following smart contracts were audited and reassessed by Inspex in detail:

Initial Audit: (Commit: 610d5a017c0ecc177d5425a45d5e74d20b4e3a80)

Contract	Location (URL)
DataStore	https://github.com/SuperLauncher/v2-core/blob/610d5a017c/contracts/core/DataStore.sol
MainWorkflow	https://github.com/SuperLauncher/v2-core/blob/610d5a017c/contracts/core/MainWorkflow.sol
Campaign	https://github.com/SuperLauncher/v2-core/blob/610d5a017c/contracts/Campaign.sol
Factory	https://github.com/SuperLauncher/v2-core/blob/610d5a017c/contracts/Factory.sol
Helper	https://github.com/SuperLauncher/v2-core/blob/610d5a017c/contracts/Helper.sol
Manager	https://github.com/SuperLauncher/v2-core/blob/610d5a017c/contracts/Manager.sol
RolesRegistry	https://github.com/SuperLauncher/v2-core/blob/610d5a017c/contracts/RolesRegistry.sol

Reassessment: (Commit: ba2ce0aedbefd79d6f2154f7ccc4954bba33592a)

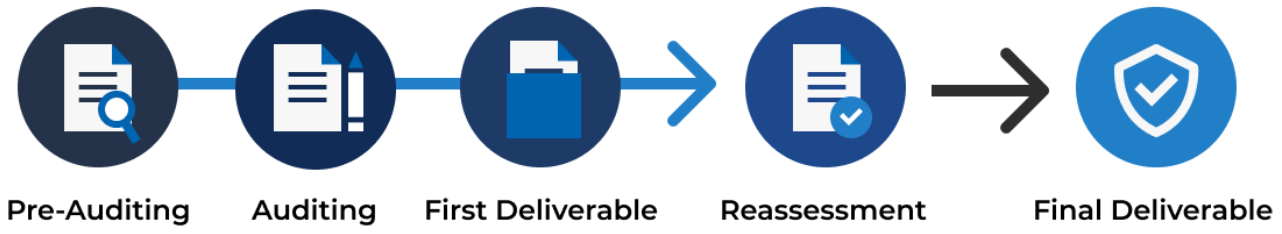
Contract	Location (URL)
DataStore	https://github.com/SuperLauncher/v2-core/blob/ba2ce0aedb/contracts/core/DataStore.sol
MainWorkflow	https://github.com/SuperLauncher/v2-core/blob/ba2ce0aedb/contracts/core/MainWorkflow.sol
Campaign	https://github.com/SuperLauncher/v2-core/blob/ba2ce0aedb/contracts/Campaign.sol
Factory	https://github.com/SuperLauncher/v2-core/blob/ba2ce0aedb/contracts/Factory.sol
Helper	https://github.com/SuperLauncher/v2-core/blob/ba2ce0aedb/contracts/Helper.sol
Manager	https://github.com/SuperLauncher/v2-core/blob/ba2ce0aedb/contracts/Manager.sol
RolesRegistry	https://github.com/SuperLauncher/v2-core/blob/ba2ce0aedb/contracts/RolesRegistry.sol

The assessment scope covers only the in-scope smart contracts and the smart contracts that they inherit from.

3. Methodology

Inspex conducts the following procedure to enhance the security level of our clients' smart contracts:

1. **Pre-Auditing**: Getting to understand the overall operations of the related smart contracts, checking for readiness, and preparing for the auditing
2. **Auditing**: Inspecting the smart contracts using automated analysis tools and manual analysis by a team of professionals
3. **First Deliverable and Consulting**: Delivering a preliminary report on the findings with suggestions on how to remediate those issues and providing consultation
4. **Reassessment**: Verifying the status of the issues and whether there are any other complications in the fixes applied
5. **Final Deliverable**: Providing a full report with the detailed status of each issue



3.1. Test Categories

Inspex smart contract auditing methodology consists of both automated testing with scanning tools and manual testing by experienced testers. We have categorized the tests into 3 categories as follows:

1. **General Smart Contract Vulnerability (General)** - Smart contracts are analyzed automatically using static code analysis tools for general smart contract coding bugs, which are then verified manually to remove all false positives generated.
2. **Advanced Smart Contract Vulnerability (Advanced)** - The workflow, logic, and the actual behavior of the smart contracts are manually analyzed in-depth to determine any flaws that can cause technical or business damage to the smart contracts or the users of the smart contracts.
3. **Smart Contract Best Practice (Best Practice)** - The code of smart contracts is then analyzed from the development perspective, providing suggestions to improve the overall code quality using standardized best practices.

3.2. Audit Items

The following audit items were checked during the auditing activity.

General
Reentrancy Attack
Integer Overflows and Underflows
Unchecked Return Values for Low-Level Calls
Bad Randomness
Transaction Ordering Dependence
Time Manipulation
Short Address Attack
Outdated Compiler Version
Use of Known Vulnerable Component
Deprecated Solidity Features
Use of Deprecated Component
Loop with High Gas Consumption
Unauthorized Self-destruct
Redundant Fallback Function
Insufficient Logging for Privileged Functions
Invoking of Unreliable Smart Contract
Use of Upgradable Contract Design
Advanced
Business Logic Flaw
Ownership Takeover
Broken Access Control
Broken Authentication
Improper Kill-Switch Mechanism

Improper Front-end Integration
Insecure Smart Contract Initiation
Denial of Service
Improper Oracle Usage
Memory Corruption
Best Practice
Use of Variadic Byte Array
Implicit Compiler Version
Implicit Visibility Level
Implicit Type Inference
Function Declaration Inconsistency
Token API Violation
Best Practices Violation

3.3. Risk Rating

OWASP Risk Rating Methodology[1] is used to determine the severity of each issue with the following criteria:

- **Likelihood**: a measure of how likely this vulnerability is to be uncovered and exploited by an attacker.
- **Impact**: a measure of the damage caused by a successful attack

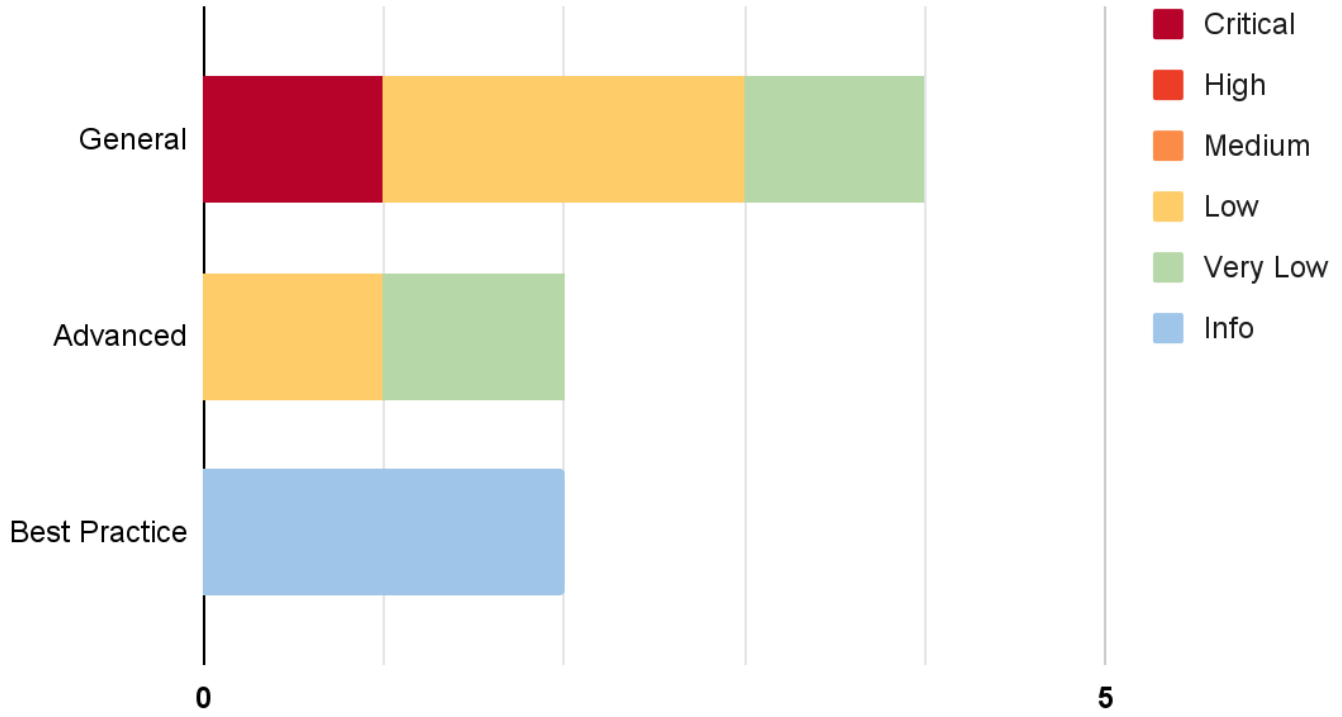
Both likelihood and impact can be categorized into three levels: **Low**, **Medium**, and **High**.

Severity is the overall risk of the issue. It can be categorized into five levels: **Very Low**, **Low**, **Medium**, **High**, and **Critical**. It is calculated from the combination of likelihood and impact factors using the matrix below. The severity of findings with no likelihood or impact would be categorized as **Info**.

Impact \ Likelihood	Low	Medium	High
Low	**Very Low**	**Low**	**Medium**
Medium	**Low**	**Medium**	**High**
High	**Medium**	**High**	**Critical**

4. Summary of Findings

From the assessments, Inspex has found 8 issues in three categories. The following chart shows the number of the issues categorized into three categories: **General**, **Advanced**, and **Best Practice**.



The statuses of the issues are defined as follows:

Status	Description
Resolved	The issue has been resolved and has no further complications.
Resolved *	The issue has been resolved with mitigations and clarifications. For the clarification or mitigation detail, please refer to Chapter 5.
Acknowledged	The issue's risk has been acknowledged and accepted.
No Security Impact	The best practice recommendation has been acknowledged.

The information and status of each issue can be found in the following table:

ID	Title	Category	Severity	Status
IDX-001	Reentrancy Attack	General	**Critical**	Resolved
IDX-002	Transaction Ordering Dependence	General	**Low**	Resolved
IDX-003	Bad Randomness	General	**Low**	Resolved
IDX-004	Missing Input Validation for feePcnt	Advanced	**Low**	Resolved
IDX-005	Missing Parameter Size Validation	Advanced	**Very Low**	Resolved
IDX-006	Insufficient Logging for Privileged Functions	General	**Very Low**	Resolved
IDX-007	Improper Function Visibility	Best Practice	**Info**	Resolved
IDX-008	Inexplicit Solidity Compiler Version	Best Practice	**Info**	Resolved

* The mitigations or clarifications by SuperLauncher can be found in Chapter 5.

5. Detailed Findings Information

5.1. Reentrancy Attack

ID	IDX-001
Target	Campaign MainWorkflow
Category	General Smart Contract Vulnerability
CWE	CWE-841: Improper Enforcement of Behavioral Workflow
Risk	**Severity: Critical** **Impact: High** The whole balance of the native token in the contract can be drained, so the users and the project owner cannot withdraw their unused funds. This results in a massive monetary loss and reputation damage. **Likelihood: High** The attacker can gain high profit from this issue, so it is very likely that this issue will happen because the motivation to attack is very high.
Status	Resolved SuperLauncher team has resolved this issue as suggested in commit `dc22e318482407ca504346f75933a463f3a3a354`.

5.1.1. Description

The `_transferOut()` function is an internal function used to transfer a token from the campaign out to a specific address.

MainWorkflow.sol

```
336 function _transferOut(address to, uint amount, DataTypes.FundType fundType)
    internal  {
337     _transferOut(to, _getAddress(fundType), amount);
338 }
339
340 function _transferOut(address to, address token, uint amount) internal  {
341
342     if (amount > 0 && to != Constant.ZERO_ADDRESS) {
343
344         if (token == Constant.ZERO_ADDRESS) {
345             (bool success, ) = to.call{ value: amount}("");
346             _require(success, Error.Code.ValidationError);
347         } else {
348              ERC20(token).safeTransfer(to, amount);
```

```
349             }
350         }
351 }
```

In the case that the `fundType` is native token, the `_transferOut()` function will transfer the native token by calling the `("")` function and send native token value in the calling to the target address, allowing a reentrant function calling from this point.

The `_transferOut()` function is called by various `public` and `external` functions. There are some functions that can be affected by this reentrancy issue.

For example, the `refundExcess()` function is used to transfer subscribers' unused funds back to them.

MainWorkflow.sol

```
174 function refundExcess() external {
175     (bool refunded, uint capital, uint egg) = getRefundable(msg.sender);
176     _require(!refunded && getState(DataTypes.Ok.Tally),
    Error.Code.CannotRefundExcess);
177
178     _transferOut(msg.sender, capital, DataTypes.FundType.Currency);
179     _transferOut(msg.sender, egg, DataTypes.FundType.Egg);
180
181     _subscriptions().items[msg.sender].refundedUnusedCapital = true;
182     _history().record(DataTypes.ActionType.RefundExcess, msg.sender, capital,
    egg, true);
183 }
```

The `refundedUnusedCapital` state is updated after the `_transferOut()` function is called. So a reentrant calling will pass the `_require()` statement that checks the `!refunded` value.

This reentrancy attack can be done by using contract address as a `to` address and make the `fallback()` in attack contract to call the target's `refundExcess()` function, for example (functions for the subscription of the campaign are omitted):

Attack.sol

```
1  pragma solidity 0.8.10;
2
3  interface ITarget {
4      function refundExcess() external;
5  }
6
7  contract Attack {
8
9      address public target;
10     uint public refundAmount;
```

```
11      constructor(address _target, uint _amount) {
12          target = _target;
13          refundAmount = _amount;
14      }
15
16      function refundExcessOnTarget() external {
17          ITarget(target).refundExcess();
18      }
19
20      function getTargetNativeBalance() public returns(uint) {
21          return address(target).balance;
22      }
23
24      function() external payable {
25          if(getTargetNativeBalance() >= refundAmount) {
26              ITarget(target).refundExcess();
27          }
28      }
29  }
```

After the attacker has subscribed to the campaign and has some unused funds left, the attacker can call the `refundExcessOnTarget()` to start the attack. The target's `_transferOut()` function calling in the `refundExcess()` function will reach the fallback function of the `Attack` contract. The fallback function then calls the target's `refundExcess()` function again repeatedly until the fund in the target contract is lower than the `refundAmount`.

Using this flaw, the attacker can drain the whole native token balance of the contract, so other users and the project owner will not be able to retrieve their funds.

5.1.2. Remediation

Inspex suggests using OpenZeppelin's `ReentrancyGuard` contract and adding the `nonReentrant` modifier in all functions that is the entry point to the `_transferOut()` function, or use the "Check-Effects-Interaction" pattern, for example:

MainWorkflow.sol

```
174 import "@openzeppelin/contracts/security/ReentrancyGuard.sol";
175
176 function refundExcess() external nonReentrant {
177     (bool refunded, uint capital, uint egg) = getRefundable(msg.sender);
178     _require(!refunded && getState(DataTypes.Ok.Tally),
    Error.Code.CannotRefundExcess);
179
180     _subscriptions().items[msg.sender].refundedUnusedCapital = true;
181
182     _history().record(DataTypes.ActionType.RefundExcess, msg.sender, capital,
```

```
egg, true);

    _transferOut(msg.sender, egg, DataTypes.FundType.Egg);
    _transferOut(msg.sender, capital, DataTypes.FundType.Currency);
}
```

5.2. Transaction Ordering Dependence

ID	IDX-002
Target	Campaign
Category	General Smart Contract Vulnerability
CWE	CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')
Risk	**Severity:** Low **Impact:** Medium The swap transaction can be front-run by the deployer or front-running bots, causing the $WBNB amount received from the swap transaction to be lower than what it should be. This results in less funds for the liquidity provision and loss of reputation for the platform. **Likelihood:** Low The `minAmountOut` can only be set by the deployer only, and it is unlikely for the deployer to set that value improperly.
Status	Resolved SuperLauncher team has resolved this issue by using the token price from Chainlink oracle to calculate the `minWBnbOut` amount with a maximum of 3% slippage in commit `ba2ce0aedbefd79d6f2154f7ccc4954bba33592a`. For the `setBnbOracle()` function in the `Manager` contract that can change the oracle address anytime, the SuperLauncher team has clarified that the `setBnbOracle()` can be called by `onlyAdmin` role only. Only the MultiSig wallet has the `onlyAdmin` role, and the wallet will be used to call the `setBnbOracle()` function only when new currency is needed to be added.

5.2.1. Description

The `swapCurrencyToWBnb()` function in the `LpProvision` library used to swap the raised currency to $WBNB before provide the liquidity.

LpProvision.sol

```
189 function swapCurrencyToWBnb(DataTypes.Lp storage param, uint fundAmt, uint
    minWBnbAmountOut, ILpProvider provider) external returns (bool) {
190
191     // Can only swap 1 time successfully
192     _require(param.swap.needSwap && !param.swap.swapped,
    Error.Code.ValidationError);
193
194     // Use pancakeswap to swap
195     (address router, ) =
```

```
provider.getLpProvider(DataTypes.LpProvider.PancakeSwap);
196
197     address wbnb = IUniswapV2Router02(router).WETH();
198     if (param.data.currency == address(0) || param.data.currency == wbnb) {
199         return false;
200     }
201
202     address[] memory path = new address[](2);
203     path[0] = param.data.currency;
204     path[1] = wbnb;
205
206     if (!ERC20(param.data.currency).approve(router, fundAmt)) { return false; }
207
208     (uint[] memory amounts) =
IUniswapV2Router02(router).swapExactTokensForTokens(
209         fundAmt,
210         minWBnbAmountOut,
211         path,
212         address(this),
213         block.timestamp + 100000000);
214
215     _require(amounts.length == 2, Error.Code.InvalidArray);
216
217     // Update
218     param.swap.swapped = true;
219     param.swap.newCurrencyAmount = amounts[1];
220     return true;
221 }
```

The `swapCurrencyToWBnb()` function in the library can be called in the `swapToWBnbBase()` function on the campaign by the deployer only.

Campaign.sol

```
128 function swapToWBnbBase(uint minAmountOut) external onlyDeployer {
129     _lp().swapCurrencyToWBnb(getLpFund(), minAmountOut, _getLpInterface());
130 }
```

The deployer can set the `minAmountOut` parameter to a low value, allowing the swap transaction to be front-run by the deployer or other front-running bots. The front-running can make the `amountOut` lower, resulting in the loss of $WBNB for the liquidity provision.

5.2.2. Remediation

Inspex suggests setting the acceptable price impact percentage and calculating the `minAmountOut` by using the price oracle, for example:

Campaign.sol

```
128 import "./interfaces/IOracle";
129
130 uint public priceImpactPctn = 2e6; // Set to an acceptable price impact
    percentage
131 address public oracle;
132
133 function swapToWBnbBase() external onlyDeployer {
134     uint minAmountOut = IOracle(oracle).consult(getLpFund()) * (100e6 -
    priceImpactPctn) / 100e6;
135     _lp().swapCurrencyToWBnb(getLpFund(), minAmountOut, _getLpInterface());
136 }
```

5.3. Bad Randomness

ID	IDX-003
Target	Campaign
Category	General Smart Contract Vulnerability
CWE	CWE-330: Use of Insufficiently Random Values
Risk	**Severity:** Low **Impact:** Medium The owner can control the `winnerStartIndex` number, which can be unfair to other users. This results in loss of reputation for the platform. **Likelihood:** Low Only the deployer can call the `tallySubscriptionAuto()` and the `tallySubscriptionManual()` functions. It is unlikely that the deployer will manipulate the `winnerStartIndex` for a small amount of the allocation with the risk of reputation loss.
Status	Resolved SuperLauncher team has resolved this issue by using Chainlink VRF as the randomness source of the `randomIndex` in commit `dc22e318482407ca504346f75933a463f3a3a354`.

5.3.1. Description

In the `Campaign` contract, the deployer role can call the `tallySubscriptionAuto()` or `tallySubscriptionManual()` functions to summarize the allocation data in the lottery phase.

The `tallySubscriptionAuto()` function calls the internal `_tallySubscription()` function.

Campaign.sol

```
113 function tallySubscriptionAuto() external onlyDeployer notAborted {
114     (, uint a, uint b) = peekTally();
115     uint ratio = (a==0 && b==0) ? Constant.PCNT_50 : (a * Constant.PCNT_100) /
    (a + b);
116     _tallySubscription(ratio);
117 }
118
119 function tallySubscriptionManual(uint splitRatio) external onlyDeployer
    notAborted {
120     _tallySubscription(splitRatio);
121 }
```

The `_tallySubscription()` function calls the `tally()` function of the `Lottery` library.

MainWorkflow.sol

```
303 function _tallySubscription(uint splitRatio) internal {
304     _require(_canTally() && splitRatio<=Constant.PCNT_100,
    Error.Code.ValidationError);
305
306     // Amount left after Guranteed has gotten their allocation
307     uint amtLeft = _data().hardCap - _guaranteed().totalSubscribed;
308
309     // Split the amount according to the splitRatio sent in by FE/Deployer
310     uint amtForLottery = (amtLeft * splitRatio) / Constant.PCNT_100;
311
312     // allocate to lottery & over-subcription
313     _lottery().tally(amtForLottery);
314     _overSubscriptions().tally(amtLeft - amtForLottery);
315
316     // Final left over
317     _live().allocLeftAtOpen = _lottery().getFinalLeftOver() +
    _overSubscriptions().getFinalLeftOver();
318     _setState(DataTypes.Ok.Tally, true);
319 }
```

The `tally()` function uses the result of the `randomIndex()` function to calculate `winnerStartIndex` value.

Lottery.sol

```
23  function tally(DataTypes.Lottery storage param, uint allocAmt) external {
24      _require(!param.data.tallyCompleted, Error.Code.NotReady);
25      param.data.tallyCompleted = true;
26
27      param.result.leftOverAmount = allocAmt; // default value in case no one in
    lottery //
28      param.data.totalAllocatedAmount = allocAmt;
29
30      uint numWinners = allocAmt / param.data.eachAllocationAmount;
31      if (numWinners==0 || param.count==0) {
32          return;
33      }
34
35      if (numWinners > param.count) {
36          numWinners = param.count;
37      }
38
39      param.result.numWinners = numWinners;
40      param.result.leftOverAmount = allocAmt - (numWinners *
    param.data.eachAllocationAmount);
41
```

```
42      // pick random index if needed //
43      if (numWinners < param.count) {
44          param.result.winnerStartIndex = randomIndex(param.count);
45      }
46  }
```

The randomIndex uses `block.difficulty` and `block.timestamp` to generate a random number.

Lottery.sol

```
87  function randomIndex(uint range) public view returns (uint) {
88      return  uint(keccak256(abi.encodePacked(range, msg.sender,
    block.difficulty, block.timestamp))) % range;
89  }
```

The result of the random can easily be predicted from the `block.difficulty` and `block.timestamp` values. Therefore, the deployer can calculate the result in each block and wait until the deployer's addresses have the allocations for lottery phase before calling the `tallySubscriptionAuto()` or the `tallySubscriptionManual()` functions.

5.3.2. Remediation

Inspex suggests using a provably-fair and verifiable source of randomness to calculate the `randomIndex`.

In this case, Inspex recommends using Chainlink VRF[2] as the randomness source of the `randomIndex`.

5.4. Missing Input Validation for feePcnt

ID	IDX-004
Target	Campaign
Category	Advanced Smart Contract Vulnerability
CWE	CWE-20: Improper Input Validation
Risk	**Severity:** Low **Impact:** Medium The `finishUp()` function that calls the `_getFinishUpStats()` function will be unusable, so the finish up process cannot be completed. As a result, the processes that require the completion of the finish up process cannot start, and the raised funds will stuck in the contract. **Likelihood:** Low It is very unlikely that the configurator role will set the fee to be over 100%, as there is no benefit in doing that, resulting in low motivation for the action.
Status	Resolved SuperLauncher team has resolved this issue by validating that the `feePcnt` must not be greater than `Constant.PCNT_100` on the `Generic` library in commit `dc22e318482407ca504346f75933a463f3a3a354`.

5.4.1. Description

The `_getFeeAmount()` function is used for calculating fees using the `Constant.PCNT_100` as the denominator.

DataStore.sol

```
226 function _getFeeAmount(uint totalAmount) internal view returns (uint) {
227     return (totalAmount * _dataStore.data.feePcnt) / Constant.PCNT_100;
228 }
```

The `Constant.PCNT_100` value is set to 1e6 in line 15.

Constant.sol

```
 6 library Constant {
 7
 8     uint    public constant FACTORY_VERSION = 1;
 9     address public constant ZERO_ADDRESS    = address(0);
10
11     string public constant  BNB_NAME        = "BNB";
12     uint    public constant VALUE_E18       = 1e18;
```

```
13      uint    public constant VALUE_10K       = 10_000e18;
14      uint    public constant VALUE_100       = 100e18;
15      uint    public constant PCNT_100        = 1e6;
16      uint    public constant PCNT_10         = 1e5;
17      uint    public constant PCNT_50         = 5e5;
18      uint    public constant MAX_PCNT_FEE    = 3e5; // 30% fee is max we can set
    //
19      uint    public constant PRIORITY_MAX    = 100;
20  }
```

The `feePcnt` can be defined by the configurator role (`init` modifier) on the contract initialization.

Campaign.sol

```
51  function initialize(
52      address token,
53      uint[4] calldata dates, // subStart, subEnd, idoStart, idoEnd //
54      uint[2] calldata caps, //softCap, hardCap
55      uint tokenSalesQty,
56      uint[4] calldata subInfo, // snapShotId, lotteryAmt, stdOverSubscribeAmt,
    eggBurnForOverSubscribe
57      uint[2] calldata buyLimitsPublic, // the min and max buy limit for public
    round
58      address currency,
59      uint feePcnt
60  ) external init {
61      _store().initialize(token, dates, caps, tokenSalesQty, subInfo,
    buyLimitsPublic, currency, feePcnt,
62          _manager.getSvLaunchAddress(), _manager.getEggAddress());
63  }
```

When the `feePcnt` that is set in the initializing is more than `Constant.PCNT_100`, the fee amount will be more than the total raised amount, resulting in the `finishUp()` function that calls the `_getFinishUpStats()` function to be unusable.

MainWorkflow.sol

```
403 function _getFinishUpStats() private view returns(bool softCapMet, uint feeAmt,
    uint totalAfterFeeLp, uint unusedLpTokensQty, uint unsoldTokensQty) {
404
405     uint amtReturn = _data().hardCap;
406     uint total = getTotalAllocSold();
407
408     // Has met SoftCap ?
409     softCapMet = total >= _data().softCap;
410     if (softCapMet) {
411         if (_data().feePcnt > 0 ) {
412             feeAmt = _getFeeAmount(total);
```

```
413             total -= feeAmt;
414         }
415
416         // Deduct for LP ?
417         DataTypes.Lp storage lp = _lp();
418         if (lp.enabled) {
419
420             (uint totalLpTokens, ) = _lp().getMaxRequiredTokensQty();
421
422             // LP creation is based on the totalRaisedAmount after deducting
    for fee first
423             (uint lpTokensUsed, uint lpFundUsed) = lp
    .getRequiredTokensQty(total);
424             total -= lpFundUsed;
425             unusedLpTokensQty = totalLpTokens - lpTokensUsed;
426         }
427         amtReturn = getAllocLeftForLive();
428     }
429     totalAfterFeeLp = total;
430     unsoldTokensQty = getTokensForCapital(amtReturn);
431 }
```

5.4.2. Remediation

Inspex suggests validating that the `feePcnt` must not be greater than `Constant.PCNT_100` on the contract initialization, for example:

Campaign.sol

```
1  function initialize(
2      address token,
3      uint[4] calldata dates, // subStart, subEnd, idoStart, idoEnd //
4      uint[2] calldata caps, //softCap, hardCap
5      uint tokenSalesQty,
6      uint[4] calldata subInfo, // snapShotId, lotteryAmt, stdOverSubscribeAmt,
    eggBurnForOverSubscribe
7      uint[2] calldata buyLimitsPublic, // the min and max buy limit for public
    round
8      address currency,
9      uint feePcnt
10 ) external init {
11     _require(feePcnt <= Constant.PCNT_100, Error.Code.ValidationError);
12     _store().initialize(token, dates, caps, tokenSalesQty, subInfo,
    buyLimitsPublic, currency, feePcnt,
13         _manager.getSvLaunchAddress(), _manager.getEggAddress());
14 }
```

5.5. Missing Parameter Size Validation

ID	IDX-005
Target	Campaign Helper
Category	Advanced Smart Contract Vulnerability
CWE	CWE-20: Improper Input Validation
Risk	**Severity:** Very Low **Impact:** Low The pack algorithm used by `pack()` and `subscribe()` functions pack data from multiple parameters to two parameters. The algorithm is missing of parameter size checking, leading to improper data logging, and can affect the platform reputation. **Likelihood:** Low The `pack()` and `subscribe()` functions' visibilities are `external`, so they can be called by anyone. This issue occurs when the values of `amtOverSub`, `eggTotalQty`, or `amtBasic` are larger than the maximum sizes of their data types. These values are from the user's subscription, so it is unlikely for them to be improperly large.
Status	Resolved The SuperLauncher team has resolved this issue as suggested in commit `dc22e318482407ca504346f75933a463f3a3a354`.

5.5.1. Description

The `Campaign` (inherit from `MainWorkflow`) and `Helper` contracts pack data into `pack1` and `pack2` variables by shifting bits.

MainWorkflow.sol

```
60  function subscribe(uint amtBasic, uint amtOverSub, uint priority, uint
    eggTotalQty) external payable {
61      DataTypes.Subscriptions storage subs = _subscriptions();
62      bool subscribed = subs.items[msg.sender].paidCapital != 0;
63      uint capital = amtBasic + amtOverSub;
64
65      _require(!_isAborted() && !subscribed && capital > 0 &&
    _isPeriod(DataTypes.Period.Subscription), Error.Code.ValidationError);
66      _transferIn(capital, DataTypes.FundType.Currency);
67
68      // Try to subscribe with checks on amounts
69      (uint maxAlloc, bool isGuaranteed) = _store().getGuaranteedAmt(msg.sender);
70
71      if (isGuaranteed) {
```

```
72          _guaranteed().subscribe(amtBasic, maxAlloc);
73      } else {
74          _lottery().subscribe(amtBasic);
75      }
76
77      // Over-subscribe & Egg transfer in.
78      if (amtOverSub > 0) {
79          _overSubscriptions().subscribe(amtOverSub, priority, eggTotalQty);
80          _transferIn(eggTotalQty, DataTypes.FundType.Egg);
81      }
82
83      // Record
84      _recordSubscription(subs, capital);
85
86      // History
87      // Data1 : Bit 0: Guaranteed | Bit 1 onwards: Amount in Currency
88      // Data2 : 120 bits: OverSub Amt, 120 bits: EggBurnAmt, 8 bits: Priority
89       uint pack1 = (amtBasic << 1) | (isGuaranteed ? 1 : 0);
90       uint pack2 = (amtOverSub) | (eggTotalQty << 120) | (priority << 240);
91      _history().record(DataTypes.ActionType.Subscribe, msg.sender, pack1, pack2,
    true);
92  }
```

Helper.sol

```
13  function pack(bool isGuaranteed, uint amtBasic, uint amtOverSub, uint
    eggTotalQty, uint priority) external pure returns (uint, uint) {
14      uint pack1 = (amtBasic << 1) | (isGuaranteed ? 1 : 0);
15      uint pack2 = (amtOverSub) | (eggTotalQty << 120) | (priority << 240);
16      return (pack1, pack2);
17  }
18
19  function unPack(uint data1, uint data2) external pure returns(bool, uint, uint,
    uint, uint) {
20
21      bool isGuaranteed = (data1 & 1) > 0 ? true : false;
22      uint amtBasic = (data1 >> 1);
23      uint amtOverSub = uint120(data2);
24      uint eggTotalQty = uint120(data2 >> 120);
25      uint priority = uint16(data2 >> 240);
26      return (isGuaranteed, amtBasic, amtOverSub, eggTotalQty, priority);
27  }
```

The pack algorithm shifts left bits data before storing them in 256 bit parameters (uint256). When the shifted bits are non 0 values, the bits data will be lost.

For example, if the priority value is 65536, which is more than the max value of uint16. The bit data of 65536 is 10000000000000000 (17 bits). The result after shifting left for 240 bits will be 0, making the data that is recorded in the history and the unpacked data to be incorrect.

5.5.2. Remediation

Inspex suggests validating that the size of input parameter value must not be higher than the max value after being packed, for example:

MainWorkflow.sol

```
60  function subscribe(uint amtBasic, uint amtOverSub, uint priority, uint
    eggTotalQty) external payable {
61      DataTypes.Subscriptions storage subs = _subscriptions();
62      bool subscribed = subs.items[msg.sender].paidCapital != 0;
63      uint capital = amtBasic + amtOverSub;
64
65      _require(!_isAborted() && !subscribed && capital > 0 &&
    _isPeriod(DataTypes.Period.Subscription), Error.Code.ValidationError);
66      _transferIn(capital, DataTypes.FundType.Currency);
67
68      // Try to subscribe with checks on amounts
69      (uint maxAlloc, bool isGuaranteed) = _store().getGuaranteedAmt(msg.sender);
70
71      if (isGuaranteed) {
72          _guaranteed().subscribe(amtBasic, maxAlloc);
73      } else {
74          _lottery().subscribe(amtBasic);
75      }
76
77      // Over-subscribe & Egg transfer in.
78      if (amtOverSub > 0) {
79          _overSubscriptions().subscribe(amtOverSub, priority, eggTotalQty);
80          _transferIn(eggTotalQty, DataTypes.FundType.Egg);
81      }
82
83      // Record
84      _recordSubscription(subs, capital);
85
86      // History
87      // Data1 : Bit 0: Guaranteed | Bit 1 onwards: Amount in Currency
88      // Data2 : 120 bits: OverSub Amt, 120 bits: EggBurnAmt, 8 bits: Priority
89      _require(amtBasic <= type(uint256).max >> 1 && amtOverSub <=
    type(uint120).max && priority <= type(uint16).max && eggTotalQty <=
    type(uint120).max, Error.Code.ValidationError);
90       uint pack1 = (amtBasic << 1) | (isGuaranteed ? 1 : 0);
91       uint pack2 = (amtOverSub) | (eggTotalQty << 120) | (priority << 240);
92      _history().record(DataTypes.ActionType.Subscribe, msg.sender, pack1, pack2,
```

```
        true);
93  }
```

Helper.sol

```
13  function pack(bool isGuaranteed, uint amtBasic, uint amtOverSub, uint
    eggTotalQty, uint priority) external pure returns (uint, uint) {
14      _require(amtBasic <= type(uint256).max >> 1 && amtOverSub <=
    type(uint120).max && priority <= type(uint16).max && eggTotalQty <=
    type(uint120).max, Error.Code.ValidationError);
15      uint pack1 = (amtBasic << 1) | (isGuaranteed ? 1 : 0);
16      uint pack2 = (amtOverSub) | (eggTotalQty << 120) | (priority << 240);
17      return (pack1, pack2);
18  }
19
20  function unPack(uint data1, uint data2) external pure returns(bool, uint, uint,
    uint, uint) {
21
22      bool isGuaranteed = (data1 & 1) > 0 ? true : false;
23      uint amtBasic = (data1 >> 1);
24      uint amtOverSub = uint120(data2);
25      uint eggTotalQty = uint120(data2 >> 120);
26      uint priority = uint16(data2 >> 240);
27      return (isGuaranteed, amtBasic, amtOverSub, eggTotalQty, priority);
28  }
```

5.6. Insufficient Logging for Privileged Functions

ID	IDX-006
Target	RolesRegistry Campaign Factory Manager
Category	Advanced Smart Contract Vulnerability
CWE	CWE-778: Insufficient Logging
Risk	**Severity:** Very Low **Impact:** Low Privileged functions' executions cannot be monitored easily by the users. **Likelihood:** Low It is not likely that the execution of the privileged functions will be a malicious action.
Status	Resolved SuperLauncher team has resolved this issue as suggested in commit `dc22e318482407ca504346f75933a463f3a3a354`.

5.6.1. Description

Privileged functions that are executable by the controlling parties are not logged properly by emitting events. Without events, it is not easy for the public to monitor the execution of those privileged functions, allowing the controlling parties to perform actions that cause big impacts to the platform.

For example, the owner can modify the role by executing `_setRole()` function in the `RolesRegistry` contract, and no event is emitted.

The privileged functions without sufficient logging are as follows:

File	Contract	Function	Modifier/Role
RolesRegistry.sol (L:38)	RolesRegistry	_setRole()	DEFAULT_ADMIN_ROLE
Campaign.sol (L:65)	Campaign	setupLp()	configure
Campaign.sol (L:79)	Campaign	setupVestingPeriods()	configure
Campaign.sol (L:91)	Campaign	setupWhitelistFcfs()	configure
Campaign.sol (L:95)	Campaign	approveConfig()	IDO_APPROVER_ROLE
Campaign.sol (L:100)	Campaign	finalize()	onlyDeployer, notAborted

Campaign.sol (L:105)	Campaign	fundIn()	onlyCampaignOwner
Campaign.sol (L:109)	Campaign	fundOut()	onlyCampaignOwner
Campaign.sol (L:113)	Campaign	tallySubscriptionAuto()	onlyDeployer, notAborted
Campaign.sol (L:119)	Campaign	tallySubscriptionManual()	onlyDeployer, notAborted
Campaign.sol (L:123)	Campaign	addRemovePrivateWhitelist()	campaignOwnerOrConfigurator
Campaign.sol (L:128)	Campaign	swapToWBnbBase()	onlyDeployer
Campaign.sol (L:133)	Campaign	addAndLockLP()	onlyDeployer
Campaign.sol (L:154)	Campaign	claimFunds()	onlyCampaignOwner
Campaign.sol (L:158)	Campaign	claimUnlockedLp()	onlyCampaignOwner, notAborted
Factory.sol (L:15)	Factory	createCampaign()	IDO_DEPLOYER_ROLE
Manager.sol (L:117)	Manager	addCurrency()	onlyAdmin
Manager.sol (L:128)	Manager	enableCurrency()	onlyAdmin
Manager.sol (L:205)	Manager	setLpProvider()	onlyAdmin

5.6.2. Remediation

Inspex suggests emitting events for the execution of privileged functions, for example:

RolesRegistry.sol

```
73  event setRolesRegistry(bytes32 role,address user,string state);
74  function _setRole(bytes32 role, address user, bool on) private {
75      if (on != hasRole(role, user)) {
76          if (on) {
77              grantRole(role, user);
78              emit setRolesRegistry(role,user,'grantRole');
79          } else {
80              revokeRole(role, user);
81              emit setRolesRegistry(role,user,'revokeRole');
82          }
83      }
84  }
```

5.7. Improper Function Visibility

ID	IDX-007
Target	Campaign
Category	Smart Contract Best Practice
CWE	CWE-710: Improper Adherence to Coding Standards
Risk	**Severity:** Info **Impact: None** **Likelihood: None**
Status	Resolved The SuperLauncher team has resolved this issue as suggested in commit `dc22e318482407ca504346f75933a463f3a3a354`.

5.7.1. Description

Functions with public visibility copy calldata to memory when being executed, while external functions can read directly from calldata. Memory allocation uses more resources (gas) than reading directly from calldata.

The following source code shows that the `addAndLockLP()` function of the `Campaign` contract is set to `public` and it is never called from any internal function.

Campaign.sol

```
133 function addAndLockLP(bool overrideStartVest, bool bypassSwap) public
    onlyDeployer {
134     _require(getState(DataTypes.Ok.FinishedUp), Error.Code.CannotCreateLp);
135
136     // Note: getLpFund() will return the raisedAmount after deducting for fee
137     (uint tokenUnsed, uint fundUnused) = _lp().create(getLpFund(), bypassSwap);
138     _setState(DataTypes.Ok.LpCreated, true);
139
140     if (overrideStartVest) {
141         _vesting().setVestingTimeNow();
142     }
143
144     // In the event that after LP provision, there is some amount left, we need
    to return this amount to campaign owner
145      _transferOut(_campaignOwner, tokenUnsed, DataTypes.FundType.Token);
146     // return un-used fund in either WBNB (if swapped) or in currency
147     _transferOut(_campaignOwner, fundUnused, _lp().swap.swapped ?
    DataTypes.FundType.WBnb : DataTypes.FundType.Currency);
148 }
```

5.7.2. Remediation

Inspex suggests changing the function's visibility to `external` if they are not called from any `internal` function as shown in the following example:

Campaign.sol

```
133 function addAndLockLP(bool overrideStartVest, bool bypassSwap) external
    onlyDeployer {
134     _require(getState(DataTypes.Ok.FinishedUp), Error.Code.CannotCreateLp);
135
136     // Note: getLpFund() will return the raisedAmount after deducting for fee
137     (uint tokenUnsed, uint fundUnused) = _lp().create(getLpFund(), bypassSwap);
138     _setState(DataTypes.Ok.LpCreated, true);
139
140     if (overrideStartVest) {
141         _vesting().setVestingTimeNow();
142     }
143
144     // In the event that after LP provision, there is some amount left, we need
    to return this amount to campaign owner
145      _transferOut(_campaignOwner, tokenUnsed, DataTypes.FundType.Token);
146     // return un-used fund in either WBNB (if swapped) or in currency
147     _transferOut(_campaignOwner, fundUnused, _lp().swap.swapped ?
    DataTypes.FundType.WBnb : DataTypes.FundType.Currency);
148 }
```

5.8. Inexplicit Solidity Compiler Version

ID	IDX-008
Target	DataStore Helper Manager RolesRegistry Campaign
Category	Smart Contract Best Practice
CWE	CWE-1104: Use of Unmaintained Third Party Components
Risk	**Severity:** Info **Impact: None** **Likelihood: None**
Status	Resolved The SuperLauncher team has resolved this issue as suggested in commit `dc22e318482407ca504346f75933a463f3a3a354`.

5.8.1. Description

The Solidity compiler versions declared in the smart contracts were not explicit. Each compilation may be done using different compiler versions, which may potentially result in compatibility issues, for example:

Campaign.sol

```
1 // SPDX-License-Identifier: BUSL-1.1
2
3 pragma solidity ^0.8.0;
```

The following table contains all targets which the inexplicit compiler version is declared.

Contract	Version
DataStore	^0.8.0
Helper	^0.8.0
Manager	^0.8.0
RolesRegistry	^0.8.0
Campaign	^0.8.0

5.8.2. Remediation

Inspex suggests fixing the Solidity compiler to the latest stable version. At the time of the audit, the latest stable version of Solidity compiler in major 0.8 is 0.8.10 [3]. For example:

Campaign.sol

```
1 // SPDX-License-Identifier: BUSL-1.1
2
3 pragma solidity 0.8.10;
```

6. Appendix

6.1. About Inspex

CYBERSECURITY PROFESSIONAL SERVICE

Inspex is formed by a team of cybersecurity experts highly experienced in various fields of cybersecurity. We provide blockchain and smart contract professional services at the highest quality to enhance the security of our clients and the overall blockchain ecosystem.

Follow Us On:

Website	https://inspex.co
Twitter	@InspexCo
Facebook	https://www.facebook.com/InspexCo
Telegram	@inspex_announcement

6.2. References

[1] "OWASP Risk Rating Methodology." [Online]. Available: https://owasp.org/www-community/OWASP_Risk_Rating_Methodology. [Accessed: 08-May-2021]

[2] "Get a Random Number" [Online]. Available: https://docs.chain.link/docs/get-a-random-number/. [Accessed: 22-Dec-2021]

[3] "Version 0.8.10" [Online]. Available: https://github.com/ethereum/solidity/releases/tag/v0.8.10. [Accessed: 22-December-2021]

inspeX

CYBERSECURITY PROFESSIONAL SERVICE