Three Sigma

Code Audit

Blast Off Blast Ido Pools

Disclaimer

Code Audit

Blast Off Blast Based Yield Aggregator and Launchpad

Disclaimer

The ensuing audit offers no assertions or assurances about the code's security. It cannot be deemed an adequate judgment of the contract's correctness on its own. The authors of this audit present it solely as an informational exercise, reporting the thorough research involved in the secure development of the intended contracts, and make no material claims or guarantees regarding the contract's post-deployment operation. The authors of this report disclaim all liability for all kinds of potential consequences of the contract's deployment or use. Due to the possibility of human error occurring during the code's manual review process, we advise the client team to commission several independent audits in addition to a public bug bounty program.

Table of Contents

Code Audit

Blast Off Blast Based Yield Aggregator and Launchpad

Table of Contents

Summary

Code Audit

Blast Off Blast Based Yield Aggregator and Launchpad

Summary

Three Sigma audited Blast Off in a 1 week engagement. The audit was conducted from 07-08-2024 to 14-08-2024.

Protocol Description

Blast Off IDO Pools allow the purchase of IDO tokens based on a multiplier in specific rounds. The higher the multiplier, the more tokens can be purchased.

Scope

Code Audit

Blast Off Blast Based Yield Aggregator and Launchpad

Scope

Filepath	nSLOC
src/MultiplierContract.sol	120
src/StandardIDOPool.sol	14
src/core/IDOPoolAbstract.sol	443
src/core/IDOStorage.sol	19
src/core/IDOStructs.sol	53
Total	**638**

Assumptions

OpenZeppelin is safe.

Methodology

Code Audit

Blast Off Blast Based Yield Aggregator and Launchpad

Methodology

To begin, we reasoned meticulously about the contract's business logic, checking security-critical features to ensure that there were no gaps in the business logic and/or inconsistencies between the aforementioned logic and the implementation. Second, we thoroughly examined the code for known security flaws and attack vectors. Finally, we discussed the most catastrophic situations with the team and reasoned backwards to ensure they are not reachable in any unintentional form.

Taxonomy

In this audit we report our findings using as a guideline Immunefi's vulnerability taxonomy, which can be found at immunefi.com/severity-updated/. The final classification takes into account the severity, according to the previous link, and likelihood of the exploit. The following table summarizes the general expected classification according to severity and likelihood; however, each issue will be evaluated on a case-by-case basis and may not strictly follow it.

Severity / Likelihood	LOW	MEDIUM	HIGH
NONE	None		
LOW	Low		
MEDIUM	Low	Medium	Medium
HIGH	Medium	High	High
CRITICAL	High	Critical	Critical

Project Dashboard

Code Audit

Blast Off Blast Based Yield Aggregator and Launchpad

Project Dashboard

Application Summary

Name	Blast Off
Commit	c04bb2e7f06151659a5f117102e1735008949fd5
Fix Commit	fc71c06e4a10e859d1d4583c26885b71b5e5cabc
Language	Solidity
Platform	Blast

Engagement Summary

Timeline	07-08-2024 to 14-08-2024
№ of Auditors	1
Review Time	1 week

Vulnerability Summary

Issue Classification	Found	Addressed	Acknowledged
Critical	0	0	0
High	4	4	0
Medium	3	3	0

Low	3	3	0
None	3	3	0

Category Breakdown

Suggestion	4
Documentation	0
Bug	8
Optimization	0
Good Code Practices	1

Code Maturity Evaluation

Code Audit

Blast Off Blast Based Yield Aggregator and Launchpad

Code Maturity Evaluation

Code Maturity Evaluation Guidelines

Category	Evaluation
Access Controls	The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system.
Arithmetic	The proper use of mathematical operations and semantics.
Centralization	The presence of a decentralized governance structure for mitigating insider threats and managing risks posed by contract upgrades
Code Stability	The extent to which the code was altered during the audit.
Upgradeability	The presence of parameterizations of the system that allow modifications after deployment.
Function Composition	The functions are generally small and have clear purposes.
Front-Running	The system's resistance to front-running attacks.
Monitoring	All operations that change the state of the system emit events, making it simple to monitor the state of the system. These events need to be correctly emitted.
Specification	The presence of comprehensive and readable codebase documentation.
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage.

Code Maturity Evaluation Results

Category	Evaluation
Access Controls	Satisfactory. All access control was correctly implemented.
Arithmetic	Satisfactory. No arithmetic errors were found.
Centralization	Moderate. Owners may DoS users' funds but not steal them.
Code Stability	Satisfactory. The code was stable throughout the audit.
Upgradeability	Satisfactory. The contracts are upgradeable.
Function Composition	Satisfactory. The code was correctly split into helper functions.
Front-Running	Satisfactory. No front-running issues are present.
Monitoring	Satisfactory. Most events were correctly emitted.
Specification	Satisfactory. The code follows the specifications.
Testing and Verification	Satisfactory. The codebase implements unit tests and tries out several key flows.

Findings

Code Audit

Blast Off Blast Based Yield Aggregator and Launchpad

Findings

3S-BlastOff-H01

Rank and multiplier of a user can not be set if the user is registered for **`MetaIDO`** by the admin

Id	3S-BlastOff-H01
Classification	High
Severity	High
Likelihood	Medium
Category	Bug
Status	Addressed in #9085a42.

Description

`IDOPoolAbstract::adminAddRegForMetaIDO()` sets **`metaIDO.isRegistered[user] = true;`**, but not the rank and multiplier. In **`IDOPoolAbstract::registerForMetaIDO()`** it reverts if the user is already registered **`require(!metaIDO.isRegistered[msg.sender], "User already registered");`**. Thus, it will be impossible to set the rank or multiplier for users registered by the admin.

Additionally, there is unreachable code, **`if (metaIDO.isRegistered[msg.sender])`** is always **`false`** due to the check above requiring the user to not have been registered previously.

Recommendation

There are a few solutions, for example, when manually registering a user by the admin, the rank and multiplier can be fetched from the multiplier contract.

3S-BlastOff-H02

Cancelling rounds after finalizing will lead to incorrect **`globalTokenAllocPerIDORound`** tracking

Id	3S-BlastOff-H02
Classification	High
Severity	Critical
Likelihood	Low
Category	Bug
Status	Addressed in #20bc0b1.

Description

Rounds may be claimed after finalized, in which each claim reduces **`globalTokenAllocPerIDORound[idoConfig.idoToken]`**. When rounds are canceled and were enabled, **`globalTokenAllocPerIDORound[idoConfig.idoToken]`** is reduced by **`idoConfig.idoSize`**.

Thus, it's possible for the **`globalTokenAllocPerIDORound`** to be reduced by more than **`idoConfig.idoSize`** per round in case it is finalized and canceled. This means that other rounds could not be claimed as **`globalTokenAllocPerIDORound`** would underflow as it had been reduced too much and it may not be possible to cancel them as it may underflow.

Recommendation

According to specs, rounds should be allowed to be canceled after finalized, so **`globalTokenAllocPerIDORound[idoConfig.idoToken]`** must be reduced by the amount left of **`idoTokens`** that were not claimed. A new variable needs to be created to track the amount of tokens that were claimed per round, such that on cancel after finalizing, **`globalTokenAllocPerIDORound`** can be decreased by **`globalTokenAllocPerIDORound.idoTokensSold - idoRoundConfigs[idoRoundId].idoTokensClaimed`**.

Additionally, users should not be able to participate or claim if a round was canceled.

3S-BlastOff-H03

Funding cap in **`IDOPoolAbstract::_basicParticipationCheck()`** is imprecise and can lead to excessive tokens sold

Id	3S-BlastOff-H03
Classification	High
Severity	High
Likelihood	Medium
Category	Bug
Status	Addressed in #52b843e.

Description

`IdoTokens` are allocated in **`IDOPoolAbstract::enableIDORound()`** up to **`idoConfig.idoSize`**. The funding cap in **`IDOPoolAbstract::_basicParticipationCheck()`** is performed as **`newFundedUSDValue <= idoConfig.idoSize * idoConfig.idoPrice`**, where **`newFundedUSDValue == idoConfig.fundedUSDValue + amount * idoConfig.idoPrice`**.

Firstly, **`idoConfig.fundedUSDValue`** has **`fyToken`** or **`buyToken`** units (see **`IDOPoolAbstract::participateInRound()`**), but it is summed to **`amount * idoConfig.idoPrice`**, which has different units as it is multiplied by the price.

Additionally, tokens are handed out according to **`tokenAllocation = (amount * 10**idoConfig.idoTokenDecimals) / idoConfig.idoPrice`**, but **`newFundedUSDValue`** is compared against **`amount * idoConfig.idoPrice`**, not taking into account the decimals.

Recommendation

```
function participateInRound(...) {
    ...
    uint256 to8kenAllocation = (amount * 10**idoConfig.idoTokenDecimals) /
idoConfig.idoPrice;
    uint256 newTotalTokens = idoConfig.fundedUSDValue + tokenAllocation;
    require(newTotalTokens <= idoConfig.idoSize, "Funding cap exceeded");
}
```

3S-BlastOff-H04

`MultiplierContract::proposeMultipleUpdates()` updates `maxLevel` before executing the update

Id	3S-BlastOff-H04
Classification	High
Severity	High
Likelihood	Medium
Category	Bug
Status	Addressed in #00c733a.

Description

`MultiplierContract::proposeMultipleUpdates()` updates `maxLevel` without going through the `UPDATE_INTERVAL`, which will affect the multiplier `MultiplierContract::getMultiplier()` right away, without updating the actual thresholds and multipliers.

If `maxLevel >= prevMaxLevel`, it still works correctly in `MultiplierContract::getMultiplier()` as the levels after `prevMaxLevel` will not yet have been created so the loop will just continue `if (levelThresholds[i] == 0) continue;`.

However, if `maxLevel < prevMaxLevel`, some of the previous levels will be skipped as soon as `MultiplierContract::proposeMultipleUpdates()` is called. For example, if there were 5 levels, but now just 4, if users were elegible for level 5, the maximum they can get is level 4, although `MultiplierContract::executeUpdates()`.

Lastly, `MultiplierContract::cancelUpdate()` does not revert the `maxLevel` update in `MultiplierContract::proposeMultipleUpdates()`, even though the update was cancelled.

Recommendation

The `maxLevel` should be updated only when `MultiplierContract::executeUpdates()` is called.

3S-BlastOff-M01

`IDOPoolAbstract` does not deal with yield and gas accrued on Blast

Id	3S-BlastOff-M01
Classification	Medium
Severity	Medium
Likelihood	High
Category	Suggestion
Status	Addressed in #8c72e1f.

Description

Blast accumulates yield for contracts holding **`USDB`**, **`WETH`** and **`ETH`** and enables contracts to claim some of the gas fees. However, this is not currently dealt with so **`IDOPoolAbstract`** will miss out on this.

Recommendation

Implement functionality to deal with this yield and gas fees.

3S-BlastOff-M02

fyToken contribution limits are incorrect as they compare **fyToken** and **buyToken** amounts with **idoSize** in **idoToken** units

Id	3S-BlastOff-M02
Classification	Medium
Severity	Medium
Likelihood	High
Category	Bug
Status	Addressed in #52b843e.

Description

The **fyToken** contribution limit is enforced as:

```
uint256 globalTotalFunded = idoConfig.totalFunded[buyToken] +
idoConfig.totalFunded[fyToken] + amount;
...
uint256 maxFyTokenFunding = (idoConfig.idoSize *
idoConfig.fyTokenMaxBasisPoints) / 10000;
if (globalTotalFunded > maxFyTokenFunding) revert
FyTokenContributionExceedsLimit();
```

As can be seen, **maxFyTokenFunding** is in **idoToken** units (**idoSize** refers to **idoToken**) and **globalTotalFunded** in **fyToken** or **buyToken** units.

Recommendation

Make sure the limit is imposed by comparing the same units.

3S-BlastOff-M03

`IDOPoolAbstract::withdrawSpareIDO()` does not take into account that several rounds may use the same **`IdoToken`**

Id	3S-BlastOff-M03
Classification	Medium
Severity	Medium
Likelihood	Medium
Category	Bug
Status	Addressed in #52b843e.

Description

`IDOPoolAbstract::withdrawSpareIDO()` checks if **`contractBal >= ido.idoSize`**, where **`contractBal = IERC20(ido.idoToken).balanceOf(address(this))`**. When several rounds use the same **`ido.idoToken`**, this check is incomplete as **`ido.idoSize`** only tracks the tokens of one round.

Recommendation

The correct check is withdrawing **`contractBal - globalTokenAllocPerIDORound[idoConfig.idoToken]`**, correctly tracking excess tokens.

3S-BlastOff-L01

Use of **`address.transfer`** instead of the recommended **`address.call{value: amount}("")`**

Id	3S-BlastOff-L01
Classification	Low
Severity	Low
Likelihood	Low
Category	Bug
Status	Addressed in #e1c8ca3, #00c733a.

Description

`address.transfer` is unsafe and may lead to stuck tokens as it forwards 2300 gas, which may run out of gas in the fallback function of the receiver. Additionally, future opcode changes could make it stop working.

Recommendation

Use **`address.call{value: amount}("")`**. Keep in mind that state changes must be performed before sending the funds this way as it opens the door for reentrancy. In **`IDOPoolAbstract::claimRefund()`**, funds are sent before deleting the position, which would be exploitable.

3S-BlastOff-L02

Rounds in **`IDOPoolAbstract::manageRoundToMetaIDO()`** can be overriden

Id	3S-BlastOff-L02
Classification	Low
Severity	Low
Likelihood	Low
Category	Suggestion
Status	Addressed in #a6b47e6, #52b843e.

Description

`IDOPoolAbstract::manageRoundToMetaIDO()` does not check if **`idoRoundClocks[roundId].parentMetaIdoId`** has already been set, allowing it to be overriden and rounds to be linked to more than 1 **`metaIDO`** in the **`metaIDO.roundIds`** array. This has no impact as the **`parentMetaIdoId`** would be the latest one set to the **`roundId`**, but the arrays **`metaIDO.roundIds`** of several **`metaIDO`** could be polluted with the same **`roundId`**.

Recommendation

Consider checking that **`idoRoundClocks[roundId].parentMetaIdoId == 0`** when **`addRound == true`**.

3S-BlastOff-L03

Incorrect `RefundClaim` event due to deleting `idoConfig.accountPositions[msg.sender]`

Id	3S-BlastOff-L03
Classification	Low
Severity	Low
Likelihood	High
Category	Bug
Status	Addressed in #e1c8ca3.

Description

The refund event is emitted as `emit RefundClaim(idoRoundId, msg.sender, pos.amount, pos.fyAmount);`.

`pos` is defined as `IDOStructs.Position storage pos = idoConfig.accountPositions[msg.sender];`.

Before emitting the event, the mapping is deleted `delete idoConfig.accountPositions[msg.sender];`.

Thus, when the event is emitted, `pos.amount` and `pos.fyAmount` have been deleted.

Recommendation

Emit the event before deleting the mapping.

3S-BlastOff-N01

Missing `disableInitializers()` call in `StandardIDOPool::constructor()`

Id	3S-BlastOff-N01
Classification	None
Category	Suggestion
Status	Addressed in #af509d0.

Description

The implementation contract `StandardIDOPool` can be initialized by an attacker, although it can not do anything with it.

Recommendation

Call `_disableInitializers()` in the constructor of `StandardIDOPool`.

3S-BlastOff-N02

`IDORoundConfig.idoPrice` could have a comment somewhere specifying the units for better readability

Id	3S-BlastOff-N02
Classification	None
Category	Good Code Practices
Status	Addressed in #b5123d4.

Description

`IDORoundConfig.idoPrice` is used when converting from **`IdoToken`** to **`buyToken`** or **`fyToken`**, but there is no indication of the code of its units. When participating in rounds, the resulting **`IdoToken`** are calculated from the transferred **`buyToken`** or **`fyToken`** amounts by doing:

```
uint256 tokenAllocation = (amount * 10**idoConfig.idoTokenDecimals) /
idoConfig.idoPrice;
```

Where **`amount`** is the quantity of **`buyToken`** or **`fyToken`**. This means that for the units to be correct, **`idoConfig.idoPrice`** must represent the **`buyToken`** or **`fyToken`** price per **`IdoToken`**, with units of **`buyToken`** and **`fyToken`**. Additionally, **`buyToken`** and **`fyToken`** must have the same number of decimals. IdoToken has 18 decimals, assuming **`buyToken`** has 6 decimals and each **`IdoToken`** is worth 2 **`buyToken`**, so the **`idoPrice`** is **`2e6`**, if we specify an amount of **`buyToken`** of **`100e6`**, we would get:

```
uint256 tokenAllocation = (100e6 * 1e18) / 2e6; == 50e18
```

This is correct as 1 **`idoToken`** is worth 2 **`buyToken`** and the final decimals are **`1e18`**, the decimals of **`IdoToken`**.

Recommendation

Place a comment in the code indicating that the **`IdoPrice`** is the **`buy/fyToken`** price per **`IdoToken`** with **`buy/fyToken`** decimals.

3S-BlastOff-N03

Null transfers in **`IDOPoolAbstract::_depositToTreasury()`** could be skipped

Id	3S-BlastOff-N03
Classification	None
Category	Suggestion
Status	Addressed in #f31eeb2.

Description

`IDOPoolAbstract::_depositToTreasury()` transfers **`pos.fyAmount`** and **`pos.amount - pos.fyAmount`**, but one of them may be **`0`** if the account only used **`buyTokens`** or **`fyTokens`**. Thus, if one of these 2 amounts is null, the transfer can be skipped. This can be relevant depending on the token used, as some revert on null transfers.

Recommendation

```
function _depositToTreasury(uint32 idoRoundId, IDOStructs.Position memory
pos) internal {
    IDOStructs.IDORoundConfig storage ido = idoRoundConfigs[idoRoundId];
    if (pos.fyAmount > 0) TokenTransfer._transferToken(ido.fyToken,
treasury, pos.fyAmount);
    if (pos.amount - pos.fyAmount > 0)
TokenTransfer._transferToken(ido.buyToken, treasury, pos.amount -
pos.fyAmount);
}
```