RAMEN
FINANCE

Ramen Finance
Audit Report
Summary

December 26, 2024

Conducted by:

Bogo Cvetkov (b0g0), Independent Security Researcher

Table of Contents

1. About b0g0

Bogo Cvetkov (b0g0) is a smart contract security researcher with a proven track record of consistently uncovering vulnerabilities in a wide spectrum of DeFi protocols. Constantly pushing the limits of his expertise, he strives to be a superior security partner to any protocol & client he dedicates himself to!

2. About Ramen Finance

Ramen Finance is a Berachain-native token launchpad protocol powering liquidity bootstrapping and price discovery for new assets.

3. Risk Classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

3.1. Impact

- **High** - leads to a significant loss of assets in the protocol or significantly harms a group of users
- **Medium** - leads to a moderate loss of assets in the protocol or some disruption of the protocol's functionality
- **Low** - funds are not at risk

3.2. Likelihood

- **High** - almost certain to happen, easy to perform, or highly incentivized

- **Medium** - only conditionally possible, but still relatively likely
- **Low** - requires specific state or little-to-no incentive

3.3. Handling severity levels

- **Critical** - Must fix as soon as possible (if already deployed)
- **High** - Must fix (before deployment if not already deployed)
- **Medium** - Should fix
- **Low** - Could fix
- **Governance** - Could fix

4. Executive Summary

For the duration of 8 days **b0g0** has invested his expertise as a security researcher to analyze the smart contracts of **Ramen Finance** protocol and assess the state of its security. For that time a total of 24 issues have been detected, out of which **9** have been assigned a severity level of **High** and **3** a severity level of **Medium**.

5. Disclaimer

Smart contracts are an experimental technology with many known and unknown risks. A smart contract security review can never verify the complete absence of vulnerabilities. This effort is limited by time, resources, and expertise. My evaluation of the codebase aims to uncover as many vulnerabilities as possible, given the above limitations! Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended!

Bogo assumes no responsibility for any misbehavior, bugs or exploits affecting the audited code or any part of the deployment phase.

Overview

Project	Ramen Finance
URL	https://ramen.finance/
Platform	Berachain
Language	Solidity
Repo	https://github.com/ramenfinance/ramen-finance/tree/master/contracts-ramenfinance/contracts
Commit Hash	9ea1ab25ea82db306da2e9b0e193f1845e3e19fa
Mitigation	02e49a24b5ec04bacb5c1be3fb861a2459aa3183
Dates	18 December - 26 December 2024

Scope

Contract	Address
Launchpad.sol	-
LaunchpadProjectPDS.sol	-
AxisManager.sol	-
ProjectFactoryPDS.sol	-
DexManager.sol	-

Issue Statistic

Severity	Count
High	9
Medium	3
Low/Informational	12
Total	**24**

6. Findings

6.1. High Severity

6.1.1. Anyone can settle an auction externally and permanently DOS the settlement inside AxisManager and project conclusion

Context: AxisManager.sol, LaunchpadProjectPDS.sol

Description :

The **AxisManager** defines a **settle()** function that is meant to be called by projects when they get concluded.

The code assumes that the **auctionHouse.settle()** function inside Axis protocol can be called only by the auction creator (**AxisManager**) which is not correct. And this is the source of an issue.

It would be a good idea to **restrict AxisManger calls only to project launchpads** and not allow arbitrary calls from anyone.

Recommendation:

It would be a good approach here to take advantage of the callBack mechanism provided by Axis:

These are a set of callbacks invoked at a target contract defined by the creator during auction creation. Those callbacks are called by Axis during different flows - abortion,

cancellation, settlement etc. They allow the auction creator to react to them in certain ways.

In this particular case you can use the **onSettle()** callback invoked once an auction is settled, it would call a Ramen contract (probably **AxisManager)** which would execute the respective logic from **AxisManger.settle().** This way the protocol would be able to handle the settlement safely regardless if it was done through the AxisManager or externally.

It is very important to thoroughly test out the flow after the architectural changes to make sure everything works as expected and also go through the docs in detail to make sure you have a good understanding of Axis so that you would develop the code with the appropriate assumptions

Resolution:

Fixed - The team has switched to the callback mechanism of handling the auctions

6.1.2. Anyone can abort an auction and DOS AxisManager settlement

Context: AxisManager.sol

Description:

According to AxisDocs:

“

After a batch auction concludes, there is a dedicated settlement period in which the auction seller is exclusively able to settle the auction lot. Subsequent to this, anyone may abort the auction

”

Meaning that after the dedicated settlement period, anyone can abort an auction, which would change the **lotId** status to **LotStatus.Settled** and any subsequent calls to **auctionHouse.settle()** would revert and as result project conclusion would be **DOSed**.

Recommendation:

Similar to a previous issue it is recommended here to leverage the callBack mechanism provided by Axis:

Upon abort Axis invokes the **onCancel** callback - use it in AxisManager to handle the relevant state transitions and also the refunded baseTokens that are sent back to the seller.

It is very important to thoroughly test out the flow after the architectural changes to make sure everything works as expected and also go through the docs in detail to make sure you have a good understanding of Axis so that you would develop the code with the appropriate assumptions

Resolution:

Fixed - The team has switched to the callback mechanism of handling the auctions

6.1.3. Anyone can steal the proceeds and refunds of any project, that are sent back to the AxisManager after settlement

Context: AxisManager.sol

Description:

The **AxisManager.settle()** incorrectly assumes it can only get called by a project, while in fact it can get called by anyone.

All project auctions are created through **AxisManager**, which makes it the owner for the auctions(lotId) of all projects. And since there is no validation in **settle(),** anyone can call it providing the **lotId** for any project and stealing the funds

Recommendation:

Create a mapping that stores the **project** to which a **lotId** belongs and validate inside **settle()** that the caller can only be the project for the respective lotId.

And a general advice would be to validate all the calls inside **AxisManager** - this way you would encapsulate the system to be used internally by trusted contracts and reduce the attack surface that is introduced by arbitrary callers

Resolution:

Fixed - A mapping was implemented that restricts each lotId to its respective project

6.1.4. The proceeds and refunds paid back after auction settlement are not accounted properly and would DOS project settlement/conclusion

Context: AxisManager.sol

Description:

Problem is that the calculation is not based on the actual balances received in the contract, but rather it assumes that those balances have been sent, which is not correct.

For example Axis deducts a fee from **totalIn_** (the amount of paid quote tokens) before sending it to the seller. This can be verified by taking a look at the implementation of the function in their contracts.

But since **totalIn_** is used by **AxisManager**, it would either be spent from the proceeds of another project or it will just DOS if there are not enough balances.

Recommendation:

Since settlement is permissionless in Axis and anyone can call it, using a before/after pattern with balaceOf() is not an option. You can again use the callback mechanism to handle the assets once they are sent to the protocol.

Upon settlement Axis calls the **onSettle()** callback and provides to it the exact amount of quote & base tokens that have been sent to the seller (AxisManager). Use those values when sending out the assets to the appropriate project

Resolution:

Fixed - The parameters provided by Axis in the callback are used

6.1.5. During project refund wBera proceeds from auction and private sale payments are not handled and remain stuck in the contract

Context: LaunchpadProjectPDS.sol

Description:

The **concludeProject()** function is used to settle the axis auction and then transfer out all the payment and deposited base tokens from the contract.

The problem here is that even if the project should be refunded, the axis auction is still successfully concluded - meaning that the **wBera** paid and the not-sold **baseTokens** during the auction are transferred back to the **seller** (which is the **LaunchpadProjectPDS** contract**).**

All **baseTokens** are transferred out. However not all assets in the contract are handled in case of refund and without any mechanism to take them out, they remain stuck.

Recommendation:

Make sure to properly handle all assets and also warn the users that the baseTokens they paid for might get revoked and the responsibility for refunds lies in the protocol/project owner.

Resolution:

Fixed

6.1.6. Conclusion could get DOSed in case of too many bids

Context: LaunchpadProjectPDS.sol

Description:

The **concludeProject()** function settles the auction through the AxisManage.

It always takes the total number of bids and provides them to be resolved in the **settle()** function.

Currently the project launchpad does not support incremental settlement and always tries to settle all bids at once, which will revert if the bids to process are too much.

Recommendation:

Similar to previous issues the recommendation here would be to use the callback mechanism of Axis, so that wherever the settlement is done (either through Ramen or directly from Axis) the protocol would only have to handle the settlement result (**onSettle()** callback) - e.g **conclusion** should **not depend** on calling **settle() directly.**

When you implement the callbacks make sure to consider the case described above

Resolution:

Fixed

6.1.7. Native presale payments are incorrectly transferred as wrapped assets, which might DOS project conclusion

Context: LaunchpadProjectPDS.sol

Description:

The **collectFee()** function deducts a fee from **presaleSold** assets, which is paid as native Bera tokens. However they are transferred out as wrapped assets, although they never got wrapped.

As a result those fees would either get deducted from the proceeds from the auction, or if the wBera balances are insufficient it would revert.

Recommendation:

Wrap **presaleQuoteFee** to **wBera** before sending it out.

Resolution:

Fixed

6.1.8. Improper fee & runway calculation might DOS project conclusion

Context: LaunchPadProjectPDS

Description:

In **concludeProject()** the **fee** and **operationRunway** % are calculated based on the amount of tokens sold during the Axis auction. The problem is that there is no guarantee that those **baseTokens** are available in the contract - since they have been sent to Axis, while the rest that are sent to the project might have been sold at private sale.

Since actual balances are never checked, this creates a scenario where the calculated fees and amounts for liquidity are invalid and lead to DOS.

Recommendation:

Always make sure to check the actual balances in the contract and if lower than the calculated amounts, cap them down.

This also applies to the calculated **baseFee** executed in **_collectFee()** - make sure that balances are present, before sending them out. It is a general advice for all token transfers that are done

Also make sure to properly test out scenarios with different fees in order to capture potential edge cases

Resolution:

Fixed

6.1.9. Bidders can steal wBera reserves during bid submission and use it for their bids

Context: LaunchpadProjectPDS

Description:

The bid submission function is used to submit bids to buy baseTokens with quote tokens (**wBera**).

The bidders send **native Bera**, which gets wrapped and transferred to the Axis manager to create the bid.

Problem here is that **msg.value** is not properly validated, before transferring wBera from the project to AxisManager.

This strips the protocol from its **wBera** reserves or potentially in case of a follow-up contract upgrade it could allow theft of deposits.

Recommendation:

Add a check that makes sure that **msg.value == bp.amount**

Resolution:

Fixed

6.2. Medium Severity

6.2.1. Malicious tokens can be used as base tokens in deployed projects and potentially affect the Ramen protocol as well

Context: Launchpad.sol

Description :

The Launchpad acts as a factory contract that deploys project contracts, which have the goal of distributing the project specific tokens (e.g baseTokens) through various mechanisms (airdrops, pre-sales and auctions).

As such the contract is expected to be as general as possible, allowing every project to deploy its own token. This means that there is no token whitelisting and anyone can provide any contract that adheres (superficially) to the ERC20 interface.

This seems like an inherent risk to the system, that should be isolated only to the deployed project contracts - e.g a malicious baseToken should only affect/compromise the project it is deployed for.

Additionally it should be considered that a lot of the Ramen contracts are upgradeable, meaning their logic can change, so it is imperative to **NEVER assume baseTokens would work as intended** (as a standard ERC20).

Recommendation

The most straightforward approach would be to implement token whitelisting. However given that the idea of the Launchpad is to deploy any token - such safeguard might be impossible.

In that case it is recommended to explicitly warn the users (mainly buyers of project tokens) that the project might use malicious tokens and manipulate any stage (airdrops, pre-sales, bids) so that they are well aware of the risks.

Also be very cautious when you write/upgrade logic that involves such arbitrary tokens, especially when it can affect the protocol itself.

As developers **NEVER** assume such tokens would behave properly - it is a good way not to shoot yourself in the foot and write safer code

Make sure that the ramen treasury can't get DOSed in any way by malicious tokens being sent.

The recommendation applies to any arbitrary address that the creator can provide to the project config.

Resolution:

Acknowledged - Team Comments: *"Warn Administrator to not accept any upgradable token / suspicious token"*

6.2.2. Project could be created with invalid configuration

Context: Launchpad.sol

Description:

The Launchpad is the contract that is used by authorized callers to deploy new PDS & FPS projects. In order to be deployed each project includes a set of parameters that the caller must provide as structs (**ProjectFPS** / **ProjectPDS).** The fields of those structs are very important since they get used throughout the whole deployment flow and finally get passed to the initializer of the created project contract.

It is very important that all of the fields in the parameter struct get validated to make sure that projects will not be initialized with invalid or malicious state. The launchpad contract already validates the provided input parameters, but the problem is that not all important fields get checked and hence some invalid configurations could be created.

Recommendation:

Consider validating all of the fields of the provided project launch configuration structs

Resolution:

Fixed - Team Comments: *"There will be no limit on the bid price"*

6.2.3. No actual deadline enforced when adding liquidity

Context: DexManager.sol

Description:

Inside the **_addLiquidity()** function the uniswap router is being called to provide liquidity to the freshly created pool.

Currently the deadline is calculated on chain - as result **block.timestamp** will always be the timestamp of when the transaction gets executed, meaning the deadline will always be valid since 10 minutes are counted starting when the transaction is executed.

Recommendation:

Deadline should be calculated by the off-chain nodes and provided as a parameter to the **addLiquidity()** function. Never calculate it in the contract itself.

Resolution:

Acknowledged - Team Comments: *"Hard to implement on PDS: couldn't pass trusted data from offchain to project on settle. Since we're implementing a launchpad project we actually don't want to enforce the deadline and only want to bypass the deadline check"*

6.3. Low Severity

6.3.1. Potential signature replayability on another chain

Context: Launchpad.sol

Description:

Currently the launchpad uses signatures to validate that a caller can deploy an FPS or PDS project.

The only parameters included in the signature are sender & project. This might be problematic in case the team decides to deploy the contracts on another EVM chain using the same signer.

Additionally the signature lacks a deadline, meaning it never expires and so the sender can use it at any time in the future.

It is considered a best practice to always issue signatures that are specific in order to prevent any potential replayability scenarios. The most important fields to include in a signature are:

- chainId
- contractId
- deadline

Additionally consider if the signature should be provided for a particular type of project (FPS/PDS) and if so, then add that to the signature as well

Recommendation:

Consider exploring the **EIP712** signatures by OpenZeppelin ,which is the safer way of using signatures - it includes chainId & contractId when signing messages along with a few other security mechanisms. You should add the deadline parameter

It would require a bit more code changes, but OZ already took care of lots of the details. If you decide to stick with the current approach consider including deadline, chainId and contract in the signature to make sure it is used exactly as it is supposed to.

Resolution:

Acknowledged - *The team will consider this in case they deploy to another chain in the future*

6.3.2. Fee-on-transfer/rebasing tokens not supported

Context: General

Description:

The Protocol is not supposed to work with non-standard tokens, such as tokens that withhold fee upon transfer or dynamically change their balances(LIDO stETH).

Launching projects with such tokens will either fail or deploy them with invalid state

Recommendation:

Make sure to add comments in the code and the protocol docs to warn the users that suck tokens are not supported

Resolution:

Acknowledged

6.3.3. There might be unclaimable leftover amounts during private sale

Context: LaunchpadProjectPDS.sol

Description:

The **privateSale()** function checks that the amount that the caller provides cannot be bought for 0 price.

Each buyer is pre-approved by a presale commit hash and has a maximum amount (presaleCap) of tokens he/she can buy during pre-sale.

Currently there exists the scenario that he might provide such an amount, so that the leftover between (**_presaleCap** - **received[msg.sender].presale)** might not be enough to satisfy the condition in the above check and hence it won't get sold.

Such amounts would be small, hence the low severity, but this would apply for all projects and all buyers in private sale so it makes sense to fix this.

Recommendation:

Add a check at the end that verifies that after the purchase the leftover amount will not be too small to get sold.

Resolution:

Fixed

6.3.4. Private sale buyers could get charged more than they have to

Context: LaunchpadProjectPDS.sol

Description:

The **privateSale()** function allows more than the required native tokens to be sent for the purchase.

Meaning that if the user has to pay 10 native tokens, but sends 11 he won't get the 1 extra token back.

Considering that the price is hardcoded and no oracle is used here, cost prediction is very straightforward so it would be better to revert in case **msg.value** is more than it has to

Recommendation:

Consider changing the above check to use **msg.value** ==, instead of >=

You can also add a convenience function **getCost()** that can calculate the exact price for the buyer before **privateSale()** is called

Resolution:

Acknowledged - *Team Comment: "Since there are some precision loss, there are cases when buyer must provide higher amount, this is intended"*

6.3.5. Restrict the DexManager to be called only by actually deployed project

Context: DexManager.sol

Description:

The **addLiquidity()** function of the **DexManager** is used to create a Uniswap pool for a project and provide tokens to it as liquidity. The function expects that the caller is a Launchpad project since it wraps **msg.sender** in **ILaunchpadProject()** interface.

It's recommended here to also check that msg.sender is an **actually deployed** project by the **Launchpad**. This would reduce the attack surface, since it will not allow arbitrary contracts that mimic **ILaunchpadProject to** call it directly.

Recommendation:

You can add **projectId** as input to the function and validate it.

Alternatively you can create a reverse mapping in the LAUNCHPAD (address => projectId) and query it.

Resolution:

Acknowledged

6.3.6. BidId is not returned upon bid creation and makes it hard for bidders to query it

Context: AxisManager.sol

Description:

The **submitBid()** function is used to participate in an Axis auction. The bid is created by calling **auctionHouse.bid()** which returns the id of the newly created bid after execution. This id is later provided to the **claimBids()** function to withdraw the bids to their creators

The issue here is that the returned id is never captured into a variable and returned by the **submitBid()** function. The function is called by **LaunchpadProjectPDS.**

This makes it difficult to track which id belongs to which bidder and hence can hinder the claiming process

Recommendation:

Consider returning the **bidId** from **submitBid()** and also emit an event for it, so that the bidders can store/query it after the function call and use it to claim the bids later or get info about them. You can also store that in a mapping.

Resolution:

Acknowledged

6.3.7. Conclude and bid periods overlap

Context: LaunchpadProjectPDS.sol

Description:

concludeProject() start period is defined like this:

```
require(
            block.timestamp >= _auctionEnded(project.launchTime),
            'concludeProject: project is still open'
        );
```

Recommendation:

Consider updating it like this:

```
block.timestamp > _auctionEnded(project.launchTime)
```

Replace **>=** with **>**, this ensures that the project cannot be concluded while bids are still possible- e.g **inBidPhase.**

Resolution:

Fixed

6.4. Informational

6.4.1. Insufficient validation

Context: General

Description:

Validation should be more strict in multiple places throughout the contracts. Proper validation reduces the chance of accidental mistakes:

- **LaunchPad.sol:**
  - Inside the **initialize()** function validate that:
    - The provided addresses are not **address(0)**
    - Define reasonable boundaries for the periods - the **period** state variable defines the different time periods used by the projects and is set by the admin role.
    - Inside **setPenaltyThreshold** define a max limit of 1e6 (100%)
    - Inside **setDexManager()/setAxisManager()/setProjectFactoryFPS()/ setProjectFactoryPDS()/setTimelockFactory()**/ validate that the newly provided address is not **address(0)**
    - Inside **setRamenTreasury()** validate that the newly provided treasury address is not **address(0)**
  - **ProjectFactoryPDS.sol:**
    - Inside the **constructor()** validate that the provided addresses are not **address(0).**
    - Inside **setService()** check that the provided parameter is not **address(0)**
    - Inside **create()** validate that owner is not **address(0)**

- **DexManager.sol:**
  - In the constructor make sure that **_wBERA** is not **address(0)**
- **AxisManager.sol:**
  - In the constructor check that the **_owner** and **_auctionHouse** are not **address(0),** also make sure that **duration** is above 0 since Axis requires it
  - Inside **setMinFillPercent** set an upper limit of **100e2**, since this is the max allowed by Axis or else it would revert.
  - Inside **submitBid()** check that **bidder** is not **address(0)**
- **LaunchpadProjectPDS.sol:**
  - Inside the **initializer()** check that the provided **service** & **_wBERA** parameters are not **address(0)**
  - Inside **setAxisManager()** / **setDexManager()** check that parameters are not **address(0)**
  - Inside **setProjectFee()** add the same checks implemented in **Launchpad._isValidDetails()** - to make sure the ratios are respected.

Recommendation:

Consider implementing the above recommendations

Resolution:

Partially resolved- the team has implemented the bigger part of the proposed validations - in the most important functions.

6.4.2. Gas optimizations

Context: General

Description:

Recommendation:

All issues related to gas are collected here to keep the report focused and easy to read:

- **LaunchPad.sol:**
  - Inside **_checkAllowance()** if **baseSupply** is 0 you can skip the **saleSupply** calculation and just assign it to zero like this:

```
uint256 saleSupply = isAirdropOnly ? 0 : baseSupply * 2 - (baseSupply * operationRunway) / FEES_DECIMALS;
```

- **DexManager.sol:**
  - The **wBERA** variables could be defined as immutable
  - The internal **_pow()** function is not used, consider removing it to reduce contract size
- **LaunchpadProjectPDS.sol:**
  - Inside **reserveAirdrop()** check and return early in the beginning if **airdropRoot** is 0 in order to prevent unnecessary operations .
  - Inside **reserveAirdrop()** move the following check at the top in order to save some gas:

```
require(!isReserved[_tokenOwner], 'reserveAirdrop: already reserved the airdrop');
```

  -

Resolution:

Fixed

6.4.3. Emit events on important state changes

Context: General

Description:

The following state changing functions do not emit events:

Launchpad.sol:

- **setPeriod()**
- **setRamenTreasury()**
- **setPenaltyThreshold()**
- **setDexManager()**
- **setAxisManager()**
- **setProjectFactoryFPS()**
- **setProjectFactoryPDS()**
- **setTimelockFactory()**
- **setSignerAddress()**

ProjectFactoryPDS.sol:

- **setImplementation()**
- **setService()**

AxisManager.sol:

- **setAuctionDuration()**
- **setMinFillPercent()**
- **setMinBidScaler()**

LaunchPadProjectPDS.sol:

- **reserveAirdrop()**
- **privateSale()**
- **setRoot()**
- **setProjectFee()**
- **setSlippage**
- **setVester()**

It is considered a best practice to emit events that mark changes in the state of the smart contract. It provides transparency to anyone observing how the contract is configured and also allows querying by offchain listeners when that is necessary.

Recommendation:

Consider emitting relevant events in the above functions

Resolution:

Partially resolved - the team has added events to the most important functions

6.4.4. Dex/AxisManager could be fetched from launchpad instead of being managed by separate variables in LaunchPadProjectPDS

Context: LaunchpadProjectPDS.sol

Description:

The **Launchpad** has the dex/axisManager variables defined, which are also used by **LaunchPadProjectPDS** upon initialization.

Currently there are separate setter functions for those variables both in Launchpad & LaunchPadProjectPDS.

Consider if it makes sense to remove setters in LaunchPadProjectPDS to streamline the update and syncing process.

Recommendation:

Since it is expected that those be synchronized across the contracts, consider if it makes sense to only have setter function inside the **Launchpad**, so that **LaunchPadProjectPDS** only queries it and when they get updated, the change will be reflected immediately in all projects, instead of having to go through each one separately.

Also any configuration discrepancies could be prevented this way.

Resolution:

Acknowledged