macro

Inverter A-1

Security Audit

July 19th, 2024
Version 1.0.0

Presented by 0xMacro

Table of Contents

Introduction

This document includes the results of the security audit for Inverter 's smart contract code as found in the section titled ‘Source Code’. The security audit was performed by the Macro security team from April 15th, 2024 to May 24th, 2024.

The purpose of this audit is to review the source code of certain Inverter Solidity

contracts, and provide feedback on the design, architecture, and quality of the source code with an emphasis on validating the correctness and security of the software in its entirety.

Disclaimer: While Macro's review is comprehensive and has surfaced some changes that should be made to the source code, this audit should not solely be relied upon for security, as no single audit is guaranteed to catch all possible bugs.

Overall Assessment

The following is an aggregation of issues found by the Macro Audit team:

Severity	Count	Acknowledged	Won't Do	Addressed
High	8	1	-	7
Medium	10	-	-	10
Low	11	-	-	11
Code Quality	12	-	1	11
Informational	4	1	-	3

Severity	Count	Acknowledged	Won't Do	Addressed
Gas Optimization	2	-	-	2

Inverter was quick to respond to these issues.

Specification

Our understanding of the specification was based on the following sources:

- Discussions on Telegram with the Inverter team.

Source Code

The following source code was reviewed during the audit:

- **Repository:** contracts
- **Commit Hash (initial):** `149fc638e602573217cb6037fb32dc232a9e0cba`
- **Commit Hash (final):** `8ddf548bcc75671834f95173c2c01f06de738d36`

Specifically, we audited the following contract within this repository:

Contract	SHA256
src/external/fees/FeeManager_v1.sol	`27bf857f46137c3b0ee47a9a236dc7982f6454906f3af2162db7fce4a04bbef1`
src/external/forwarder/TransactionForwarder_v1.sol	`4987cb7b3bf2930337f674baa284408ff6af9967ffedb7791dcdf84ab3b51c48`
src/external/governance/Governor_v1.sol	`0655f2e00ee00ee51bc22407315077fba1d10e2b9008ab7837d12e0cb657a03f`
src/factories/ModuleFactory_v1.sol	`0e000d86ccbd8047fd7d431624184f4346ce8be85954c941a192c8971bd69c9d`
src/factories/OrchestratorFactory_v1.soll	`35bfc1358a0b4306730bf4cb457bf8b46e00eccce1ab74910800020832d6d61b`
src/modules/authorizer/role/AUT_Roles_v1.sol	`c63be01e903c4208127e8bec6df3b5fdf2697fff55a53998a85e10cf32968555`
src/modules/authorizer/role/AUT_TokenGated_Roles_v1.sol	`47a070d7fb15e83f6e92cc835a73d400c5b70a7d8eb5f8cabf8282f835dfe413`
src/modules/base/Module_v1.sol	`b55becbd59c5b723013d3637360b64b744c36340ae4aedd99f553d5d50d1997f`

Contract	SHA256
src/modules/fundingManager/bondingCurve/FM_BC_Bancor_Redeeming_VirtualSupply_v1.sol	`c4b2eab0c6427d4ba9d125c168be7c69e652d3f806edd3f4ca1f430354d990bc`
src/modules/fundingManager/bondingCurve/abstracts/BondingCurveBase_v1.sol	`99f14a78ac35fb76daa1ac245577f95324bd57a9e3fa95bf38aebbd1a61ddfb4`
src/modules/fundingManager/bondingCurve/abstracts/RedeemingBondingCurveBase_v1.sol	`4a27092edc417d1cdf2a18e69025577c0054e8757173c2aa46b9d1623871a3df`
src/modules/fundingManager/bondingCurve/abstracts/VirtualCollateralSupplyBase_v1.sol	`db1fc70f9a6d360d43c4ae348f9b20562d10a875155bdaa087cd4b02ed9fa793`
src/modules/fundingManager/bondingCurve/abstracts/VirtualIssuanceSupplyBase_v1.sol	`807b6abb68d9fa589423f2ac7d7ef93b91162e10b8cce3498f9efb41c6953348`
src/modules/fundingManager/bondingCurve/formulas/BancorFormula.sol	`e8d9f5464a800fd2a8b131d17f87fb786ebea6c9c4bc0f31cb3fd7719b7d0ed4`
src/modules/fundingManager/bondingCurve/formulas/Utils.sol	`c7dda64262aba978fd14efc44cd0cd50b4c79876c6984546dbc73c130c4b35ec`
src/modules/fundingManager/bondingCurve/tokens/ERC20Issuance_v1.sol	`92b745d20adb479250b762a94f26f9a6fba955b1b726d2100e0b6f70469f1549`
src/modules/lib/LibMetadata.sol	`0b7ec654efffe79260c4258475f141935eeb6a5402feb64350a018e2f81fe52f`
src/modules/lib/SafeMath.sol	`1d9e4b5bf317d54a6e8631533ef4d2078383b4b8012c4004b704e54944c64d99`
src/modules/logicModule/LM_PC_KPIRewarder_v1.sol	`8bc3dc0e4a9c71a15560137bdbd26ba336d9f007acc53928f0a149da718e83ad`
src/modules/logicModule/LM_PC_Staking_v1.sol	`00f0267caed2def72fc1cd62c0729ef3ef85bd1a710f8025276186d35d69c804`
src/modules/logicModule/abstracts/ERC20PaymentClientBase_v1.sol	`923f4d38c0e26ff6924a73f0612b418e7d84f8ef39eb977dbc4b233354d0cd79`
src/modules/logicModule/abstracts/oracleIntegrations/UMA_OptimisticOracleV3/OptimisticOracleIntegrator.sol	`518718596ab0b20ff6ff4e187e8a1abf7772fc6e7393bd166961fc3a85997622`

Contract	SHA256
src/modules/paymentProcessor/PP_Simple_v1.sol	`e34967c7b7d34116b57ee78f1f4a52ed580a36e4215ddaca133345292009165a`
src/modules/paymentProcessor/PP_Streaming_v1.sol	`0373fa097e087e962f96faa6c739b430f9fe62d5b3275a3fcce0dfd69cd8edec`
src/orchestrator/Orchestrator_v1.sol	`8eaa212f78a1d0215d27dd36426c36141fdb8696064b85cfd4b8152a3fdf4db5`
src/orchestrator/abstracts/ModuleManagerBase_v1.sol	`b5aaffe2234703355ce69d7fb7b6eb9ef60866c74fde0f702fb644fbf3d4e430`
src/proxies/InverterBeaconProxy_v1.sol	`6bb530a482334855300f5709ca7f9dad1d7f0057159fec3b0cc503ef86df50f9`
src/proxies/InverterBeacon_v1.sol	`3705fa0ab2febb1afcaf61d521f1151d289f09c1e3e5851710dd78c81d17d5f5`

Note: This document contains an audit solely of the Solidity contracts listed above. Specifically, the audit pertains only to the contracts themselves, and does not pertain to any other programs or scripts, including deployment scripts.

Issue Descriptions and Recommendations

Click on an issue to jump to it, or scroll down to see them all.

~~M-1~~ RedeemingBondingCurveBase has an inflated supply that may affect pricing calculations when selling

~~M-2~~ Authorizer can set Module factory as owner, inadvertently bricking protocol ownership

~~M-3~~ Resolved assertions reported as unresolved in LM_PC_KPIRewarder_v1

~~M-4~~ Required modules can be removed

~~M-5~~ Queued stakers can receive rewards from pending assertions

~~M-6~~ Asserter can post invalid assertions with minimal penalty

~~M-7~~ Staking rounding errors

~~M-8~~ Orchestrator should not be its own owner

~~M-9~~ FM_BC_Bancor_Redeeming_VirtualSupply_v1 may get initialized with incompatible tokens

~~M-10~~ Implemented access control does not match those described in natspec

~~L-1~~ BondingCurveBase_v1 does not implement the funding manager interface

~~L-2~~ Changing modules fails if at module limit

~~L-3~~ Orchestrator's required modules interfaces are unchecked on initialization

~~L-4~~ Module manager does not check if added modules support `IModule_v1` interface

~~L-5~~ Missing validation for `assertionLiveness` in OptimisticOracleIntegrator

~~L-6~~ Metadata id hash can conflict

~~L-7~~ Interface getter functions not set as view

~~L-8~~ Removal of token gated role leaves stale data

~~L-9~~ Token interface compliance check can be easily circumvented

~~L-10~~ Events related to different orders may report same wallet ID

~~L-11~~ Default visibility of _isModule variable may break contract boundaries

~~Q-1~~ Unnecessary type casting

~~Q-2~~ Unnecessary callback function declaration in IOptimisticOracleIntegrator

~~Q-3~~ Missing function declaration in the interface

~~Q-4~~ Unused errors

~~Q-5~~ Unused events

~~Q-6~~ Events without indexed attributes

~~Q-7~~ Unnecessary Imports

Q-8 Utils.sol unused

~~Q-9~~ Unused code in BondingCurveBase_v1

~~Q-10~~ Avoid conditional expressions with bool literals

~~Q-11~~ Unnecessary duplicate function invocation

~~Q-12~~ Unnecessary variable

~~G-1~~ Remove unnecessary indexed in Event declarations

~~G-2~~ Cache cross contract reads

~~I-1~~ Fee on transfer tokens not supported

I-2 Modules can be upgraded or shutdown without warning

~~I-3~~ Tokens with callbacks on transfers are not supported

~~I-4~~ OrchestratorOwner role has control over the protocol

Security Level Reference

We quantify issues in three parts:

1. The high/medium/low/spec-breaking **impact** of the issue:
    - How bad things can get (for a vulnerability)
    - The significance of an improvement (for a code quality issue)
    - The amount of gas saved (for a gas optimization)
2. The high/medium/low **likelihood** of the issue:
    - How likely is the issue to occur (for a vulnerability)
3. The overall critical/high/medium/low **severity** of the issue.

This third part – the severity level – is a summary of how much consideration the client should give to fixing the issue. We assign severity according to the table of guidelines below:

Severity	Description
(C-x) Critical	We recommend the client **must** fix the issue, no matter what, because not fixing would mean **significant funds/assets WILL be lost.**
(H-x) High	We recommend the client **must** address the issue, no matter what, because not fixing would be very bad, *or* some funds/assets will be lost, *or* the code's behavior is against the provided spec.
(M-x) Medium	We recommend the client to **seriously consider** fixing the issue, as the implications of not fixing the issue are severe enough to impact the project significantly, albiet not in an existential manner.
(L-x) Low	The risk is small, unlikely, or may not relevant to the project in a meaningful way. Whether or not the project wants to develop a fix is up to the goals and needs of the project.
(Q-x) Code Quality	The issue identified does not pose any obvious risk, but fixing could improve overall code quality, on-chain composability, developer ergonomics, or even certain aspects of protocol design.
(I-x) Informational	Warnings and things to keep in mind when operating the protocol. No immediate action required.
(G-x) Gas Optimizations	The presented optimization suggestion would save an amount of gas significant enough, in our opinion, to be worth the development cost of implementing it.

Issue Details

~~H-1~~ Storage slot conflicts can occur when upgrading

TOPIC	STATUS	IMPACT	LIKELIHOOD
Upgradability	Fixed	Critical	Medium

Modules are intended to be upgradeable, and their implementation is determined by the `InverterBeacon_v1`, however there are extra considerations to be made, particularly to do with storage slot conflicts. These storage conflicts typically occur when an upstream contract has additional storage parameters added that weren't present in a prior implementation, which takes a storage slot, and pushes the storage slot number used in downstream contracts down, which causes the downstream contracts to no longer point to their prior slot and the new storage variable may point to storage values already set. This can lead to many unintended consequences that can be quite severe.

Currently, there are several cases of upstream contracts using regular storage slots that could conflict with inheriting contract slots on upgrade. The `FM_BC_Bancor_Redeeming_VirtualSupply_v1` funding manager contract has its own storage variables, while it inherits a chain of contracts: `RedeemingBondingCurveBase_v1`, `BondingCurveBase_v1`, and `Module_v1.sol`, each of which has its own storage variables.

It is likely that if there is an upgrade, some part of the upstream inheritance chain would require an additional storage variable, which would conflict with all storage downstream if added normally and would require a patchwork storage pattern to resolve.

Remediations to Consider

- Add storage gaps to the end of each contract that is intended to be upgradable, reserving slots for future use, or
- Implement the unstructured storage pattern to allow for the easy addition of storage variables for future module upgrades.

~~H-2~~ `BURN_ADMIN_ROLE` is not immutable and controlled by `ORCHESTRATOR_OWNER_ROLE`

TOPIC	STATUS	IMPACT	LIKELIHOOD
Authorization	Fixed	High	Medium

`AUT_Roles_v1` is a base authorizer module used throughout a protocol to determine whether modules or users are authorized for particular functions. It has four major roles: the `DEFAULT_ADMIN_ROLE`, `ORCHESTRATOR_OWNER_ROLE`, `ORCHESTRATOR_MANAGER_ROLE`, and `BURN_ADMIN_ROLE`.

Each module in the protocol also has its own module role, and thus, its role admin is the `DEFAULT_ADMIN_ROLE`, which is owned by the `ORCHESTRATOR_OWNER_ROLE`, giving it control over setting members. The `ORCHESTRATOR_OWNER_ROLE` is the main owner of the protocol and has the highest authority, and is granted the admin over the `DEFAULT_ADMIN_ROLE`, giving it control of setting members of roles for all roles that do not have an explicitly set role admin, as well as directly being set as the admin of the `ORCHESTRATOR_MANAGER_ROLE`.

The `BURN_ADMIN_ROLE` is special as it is intended to be used to remove any admin control over a module role since it is not supposed to be controlled by anyone or have any members. A module role's admin is burned when `burnAdminFromModuleRole()` is called by a module, making changes to roles immutable:

```
/// @inheritdoc IAuthorizer_v1
 function burnAdminFromModuleRole(bytes32 role)
     external
     onlyModule(_msgSender())
 {
     bytes32 roleId = generateRoleId(_msgSender(), role);
     _setRoleAdmin(roleId, BURN_ADMIN_ROLE);
 }
```

Reference: AUT_Roles_v1.sol#L225-L232

However, in order for the `BURN_ADMIN_ROLE` to work as intended, it needs not to be controlled by any other role, which is done by setting its own admin to itself as is done in the constructor:

```
constructor() {
    // make the BURN_ADMIN_ROLE immutable
    _setRoleAdmin(BURN_ADMIN_ROLE, BURN_ADMIN_ROLE);
}
```

Reference: AUT_Roles_v1.sol#L86-L89

This should work for normal contracts, but this is intended to be a proxy contract where initialization is handled in a separate initializer function, anything done to storage in the

constructor does not effect the storage of proxies that use it as an implementation contract. Since in its initializer function, there is no role admin set for the `BURN_ADMIN_ROLE`, its role admin will be the `DEFAULT_ADMIN_ROLE`, which its admin is set to be the `ORCHESTRATOR_OWNER_ROLE`, giving the ability for members of the `ORCHESTRATOR_OWNER_ROLE` to control module roles that are intended to be immutable, breaking the purpose of the `BURN_ADMIN_ROLE`.

Remediations to Consider

Set the `BURN_ADMIN_ROLE`'s admin to itself in the initializer rather than the constructor to ensure it is not controlled by any role and serves its intended purpose.

H-3 `AUT_TokenGated_Roles_v1` susceptible to flash loans

TOPIC	STATUS	IMPACT	LIKELIHOOD
Protocol Design	Acknowledged	High	High

`AUT_TokenGated_Roles_v1` is an authorizer module that allows roles to be set that are token gated, which means users have a given role if they have a sufficient amount of a set token, determined by a set `threshold`, which is checked in `_hasTokenRole()`:

```
/// @notice Internal function that checks if an account qualifies for a token-gated role.
/// @param role The role to be checked.
/// @param who The account to be checked.
function _hasTokenRole(bytes32 role, address who)
    internal
    view
    returns (bool)
{
    uint numberOfAllowedTokens = getRoleMemberCount(role);

    for (uint i; i < numberOfAllowedTokens; ++i) {
        address tokenAddr = getRoleMember(role, i);
        bytes32 thresholdId = keccak256(abi.encodePacked(role, tokenAddr));
        uint tokenThreshold = thresholdMap[thresholdId];

        //Should work with both ERC20 and ERC721
        try TokenInterface(tokenAddr).balanceOf(who) returns (
            uint tokenBalance
        ) {
            if (tokenBalance >= tokenThreshold) {
                return true;
```

```
                }
            } catch {
                // If the call fails, we continue to the next token.
                // Emitting an event here would make this function (and the functions calling
                // note we already enforce Interface implementation when granting the role.
            }
        }

        return false;
    }
```

Reference: AUT_TokenGated_Roles_v1.sol#L227-L257

However, for sufficiently liquid tokens, the balance can be manipulated within the context of a transaction to briefly exceed the required threshold via flash loans or similar interactions. This can allow users to gain access to token gated functions easily, but not actually own/hold the tokens.

Remediations to Consider

This may not be an issue if the tokens used are illiquid or non-transferrable, so users and integrators should be aware of this issue and take it into consideration when deciding eligible tokens to use to gate roles.

RESPONSE BY INVERTER

> We acknowledge that the token-gated role system in `AUT_TokenGated_Roles_v1` could potentially be manipulated for liquid tokens through flash loans or similar mechanisms. However, this module is designed with the intention of using illiquid and/or non-transferrable tokens for access control:
>
> (1) We've added explicit comments in the code to alert developers to this intended use case. (2) Our user interface will display warnings to inform users about the appropriate token types for this module.
>
> We advise users and integrators to be aware of this design consideration when implementing token-gated roles. For the most effective and secure use of this module, we recommend using illiquid or non-transferrable tokens.

~~H-4~~ PP_Streaming cannot be configured as a payment processor module

within Orchestrator

TOPIC	STATUS	IMPACT	LIKELIHOOD
Spec breaking	Fixed	High	High

`PP_Streaming_v1` is a payment processor module that performs linear payouts across time. It is meant to be used as an alternative to the `PP_Simple` module, which performs lump-sum payments. An appropriate payment processor module can be configured on the Orchestrator_v1 instance. The `Orchestrator_v1:initiateSetPaymentProcessorWithTimelock()` implementation performs a check if the provided module supports the `IPaymentProcessor_v1` interface.

However, `supportsInterface()` in `PP_Streaming_v1` is not correctly implemented and declares support only for `IPP_Streaming_v1` and `IModule_v1` interfaces (in `Module_v1:supportsInterface()`). `IPP_Streaming_v1` does inherit from the `IPaymentProcessor_v1`, but that does not affect the return result of the `supportsInterface()` when it is executed. As a result, `Orchestrator_v1:initiateSetPaymentProcessorWithTimelock()` will revert when provided module for payment processing is `PP_Streaming_v1`.

```
// add to PP_StreamingV1Test
function testSupportsPPInterface() public {
    assertTrue(
        paymentProcessor.supportsInterface(
            type(IPaymentProcessor_v1).interfaceId
        )
    );
}
```

Remediations to consider

- Update `PP_Streaming_v1:supportsInterface()` implementation and instead of `IPP_Streaming_v1` inheriting from the `IPaymentProcessor_v1` make `PP_Streaming_v1` inherit `IPaymentProcessor_v1`.

~~H-5~~ A single failed transfer may block all PP_Simple payouts

TOPIC	STATUS	IMPACT	LIKELIHOOD
Spec breaking	Fixed	High	High

In `PP_Simple_v1`, the `processPayments()` is responsible for processing all pending orders. Orders are retrieved from the associated payment client and iterated through for final asset transfers.

If a single transfer in the whole batch fails and results in a revert when the external call is made, the whole batch is reverted, and `processPayments()` does not change the state.

```
token_.safeTransferFrom(address(client), recipient, amount);
```

Asset transfers may fail for various reasons, such as the recipient being blocked in USDC. A malicious user may set a blocked recipient for pending orders and cause order batches to fail repeatedly.

Currently, the `PP_Simple_v1` module does not have a mechanism to ignore failed transfers, and the `ERC20PaymentClientBase_v1` module does not feature the capability to remove unprocessable orders from the pending queue. As a result, one single bad transfer will block all payouts.

Remediations to consider

- In the `PP_Simple_v1` implement a similar mechanism for failed transfer as in `PP_Streaming_v1`, where failed transfers become unclaimed and can be pulled by the corresponding asset recipient later.
- Also, consider having functionality for removing unprocessable orders from the pending orders in `ERC20PaymentClientBase_v1` or its child contracts.

RESPONSE BY INVERTER

> Follow-up fix: https://github.com/InverterNetwork/contracts/pull/475/commits/82052a2222196bc3c7e649ec1902686da899ba6b

~~H-6~~ Modules assume the FundingManager's token can not change

TOPIC	STATUS	IMPACT	LIKELIHOOD
Composability	Fixed	High	Medium

Throughout the protocol, modules query the Funding Managers token, interacting with it and/or storing balances denominated in this token. However, there is no guarantee that this token will not change since, in the Orchestrator, the FundingManager can be updated and does not ensure a constant token. A new funding manager could have a different value for its token, which would affect all other modules that depend on it, resulting in potential loss of funds or breaking the protocol's functionality.

Remediations to Consider

In `Orchestrator_v1.sol`, when changing the FundingManager, ensure the token is the same.

~~H-7~~ Asserted data unrelated to the `assertedValue` used in rewards

TOPIC	STATUS	IMPACT	LIKELIHOOD
Protocol Design	Fixed	High	Medium

In `LM_PC_KPIRewarder_v1.sol`, the `ASSERTER_ROLE` calls postAssertion() with an assertion of data that supposedly relates to the KPI of a relevant target. This assertion is separated into 3 main data points: the `asserted value`, `data`, and `dataId`. Of these 3 parameters, only the `data` and `dataId` parameters are actually asserted to be true via the uma optimistic oracle, while the `assertedValue` is potentially unrelated. However, `assertedValue` is the only value that actually determines the outcome of the rewards. This can lead to cases where an asserter could provide `data` and `dataId` that claims an unrelated true fact, which will not be disputed, while the provided `assertedValue` can be anything, as it is unverified.

Remediations to Consider

- Consider including the assertedValue in the data being verified by the uma optimistic

oracle.

~~H-8~~ Attacker may DoS KPI rewarder by filling out the staking queue

TOPIC	STATUS	IMPACT	LIKELIHOOD
Griefing	Fixed	High	Medium

In `LM_PC_KPIRewarder_v1.sol`, users can stake tokens to earn rewards based on KPI, which is verified by the uma optimistic oracle. However, when stake() is called, the funds are not immediately staked, but are added to a queue, so that they are ineligible for rewards on any pending KPI reward assertions. So when a new assertion is made by an address with the ASSERTER_ROLE by calling postAssertion(), all queued stake requests are processed:

```
...
// Staking Queue Management

for (uint i = 0; i < stakingQueue.length; i++) {
    address user = stakingQueue[i];
    _stake(user, stakingQueueAmounts[user]);
    totalQueuedFunds -= stakingQueueAmounts[user];
    stakingQueueAmounts[user] = 0;
}

delete stakingQueue; // reset the queue
...
```

Reference: LM_PC_KPIRewarder_v1.sol#L170-L179

However, to prevent the queue from becoming too long, there is a `MAX_QUEUE_LENGTH` when staking, preventing it from getting any larger.

```
/// @inheritdoc ILM_PC_Staking_v1
function stake(uint amount)
    external
    override
    nonReentrant
    validAmount(amount)
{
    if (stakingQueue.length >= MAX_QUEUE_LENGTH) {
        revert Module__LM_PC_KPIRewarder_v1__StakingQueueIsFull();
    }
```

```
...
```

Reference: LM_PC_KPIRewarder_v1.sol#L262-L271

The `MAX_QUEUE_LENGTH` is set to 50. Having a limit to the queue length can prevent a malicious user from staking on many addresses and expanding the queue to the point where it would exceed the gas limit to process during the `postAssertion()` call.

In adding this `MAX_QUEUE_LENGTH`, however, it does allow for a malicious user to block new stakers from staking if they stake using multiple accounts until the `MAX_QUEUE_LENGTH` is reached, preventing any other users from staking. This could be done to grief or to gain a monopoly on staking and earn all rewards themselves. In addition, it could be repeated each time a new assertion is made, which processes the queue.

Remediations to Consider

A couple of approaches could be made to resolve this issue, but in all `MAX_QUEUE_LENGTH` should be removed:

- redesign the contract not to require a queue, but rather have staked balances stored for the user but tied to the next KPICounter. Although this would take time and consideration, it would remove the issue entirely.
- Add a way for the `ASSERTER_ROLE` to process the queue in chunks, preventing the gas limit from being exceeded in a call to `postAssertion()`. Also, introduce a minimum stake amount to increase the capital requirement to grief. **Note**: this will not entirely remove the possibility of griefing from a user with enough capital, but the incentive to do so should be reduced enough to make it unlikely to occur.

~~M-1~~ RedeemingBondingCurveBase has an inflated supply that may affect pricing calculations when selling

TOPIC	STATUS	IMPACT	LIKELIHOOD
Protocol Design	Fixed	Medium	Low

In `RedeemingBondingCurveBase_v1.sol`'s `_sellOrder()`, it allows users to sell their issuance tokens for the collateral token. The amount of collateral token received for a given amount of issuance token is determined by the downstream contract implementation of

the `_redeemTokensFormulaWrapper()`.

The example current implementation, in `FM_BC_Bancor_Redeeming_VirtualSupply_v1`, uses virtual issuance and collateral supplies and a bonding curve to calculate the sell price, but there may be future implementations that use something simple like the total supply of issuance tokens compared to amount of collateral held to determine sell price. In cases where protocols use the total supply of issuance tokens in their pricing calculations it can lead to issues if there is also a protocol fee taken if the `sellFee` is set.

This is because before pricing is calculated via the implemented call to `_redeemTokensFormulaWrapper()`, sell fees are taken via minting issuance tokens to the protocol via `_processProtocolFeeViaMinting()`, which briefly increases the issuance tokens total supply:

```
...
// Process the protocol fee
 _processProtocolFeeViaMinting(issuanceTreasury, protocolFeeAmount);
 // Calculate redeem amount based on upstream formula
 uint collateralRedeemAmount = _redeemTokensFormulaWrapper(netDeposit);

 totalCollateralTokenMovedOut = collateralRedeemAmount;

 // Burn issued token from user
 _burn(_msgSender(), _depositAmount);
 ...
```

Reference: RedeemingBondingCurveBase_v1.sol#L185-L193

In simple implementations that resemble vaults, if the value of the issuance token is proportional to the amount of collateral token over the issuance tokens total supply, ie each issuance token represents a share of the collateral held in the contract, then the protocol fee being minted beforehand would inflate the total supply and decrease the value of issuance tokens in this calculation, resulting in a lower sell price than expected and fewer collateral tokens received. This effect is magnified when the total supply is low, and if the last issuance tokens are attempted to be sold, the `sellFee` effectively doubles.

Remediations to Consider

Reorder the operations to call `_redeemTokensFormulaWrapper()` before minting the protocol fees to the issuanceTreasury via `_processProtocolFeeViaMinting()`; this ensures the supply of tokens that may be used to determine prices are correct and not inflated.

~~M-2~~ Authorizer can set Module factory as owner, inadvertently bricking protocol ownership

TOPIC	STATUS	IMPACT	LIKELIHOOD
Authorization	Fixed	Medium	Low

When the base Authorizer module, AUT_Roles_v1.sol, is initialized via `__RoleAuthorizer_init()`, an `initialOwner` parameter is expected, which sets the owner of the protocol that uses it as an authorizer, the `ORCHESTRATOR_OWNER_ROLE`.

```
...
if (initialOwner != address(0)) {
    _grantRole(ORCHESTRATOR_OWNER_ROLE, initialOwner);
} else {
        ///@audit: here the message sender becomes the owner, but that is likely the modul
    _grantRole(ORCHESTRATOR_OWNER_ROLE, _msgSender());
}
...
```

Reference: AUT_Roles_v1.sol#L128-L132

Implementation allows the `initialOwner` to be set as the zero address, which in that case, sets the owner to be the `msg.sender.` However, in most cases, modules are intended to be deployed and initialized by the `ModuleFactory_v1` contract via its `createModule()` function:

```
function createModule(
    IModule_v1.Metadata memory metadata,
    IOrchestrator_v1 orchestrator,
    bytes memory configData
) external returns (address) {
    // Note that the metadata's validity is not checked because the
    // module's `init()` function does it anyway.

    IInverterBeacon_v1 beacon;
    (beacon, /*id*/ ) = getBeaconAndId(metadata);

    if (address(beacon) == address(0)) {
        revert ModuleFactory__UnregisteredMetadata();
    }

    address implementation = address(new InverterBeaconProxy_v1(beacon));

    IModule_v1(implementation).init(orchestrator, metadata, configData);
```

```
        emit ModuleCreated(
            address(orchestrator),
            implementation,
            LibMetadata.identifier(metadata)
        );

        return implementation;
    }
```

Reference: ModuleFactory_v1.sol#L115-L141

This becomes an issue because the `ModuleFactory` does not have the functionality to act as the owner and would brick the most important role of the protocol, requiring redeployment or an upgrade to resolve.

Remediations to Consider

Revert if the provided `initialOwner` is `address(0)` to prevent this invalid initialization from occurring.

~~M-3~~ Resolved assertions reported as unresolved in LM_PC_KPIRewarder_v1

TOPIC	STATUS	IMPACT	LIKELIHOOD
Spec breaking	Fixed	Medium	Low

In the `LM_PC_KPIRewarder_v1`, the implementation overrides `assertionResolvedCallback` from the parent `OptimisticOracleIntegrator` contract. Within parent implementation, `assertionData[assertionId].resolved` is set to true when it is successfully resolved, and `assertionData[assertionId]` is deleted when it is false.

However, within `LM_PC_KPIRewarder_v1's` implementation `assertionData[assertionId]` mapping is never updated. As a result, the corresponding entry will have a false value for the resolved field. In addition, `OptimisticOracleIntegrator:getData()` will return (false, 0) for already resolved assertions.

Remediations to Consider

- Update `LM_PC_KPIRewarder_v1:assertionResolvedCallback()` implementation to manage data structures related to the overridden parent functions properly.

~~M-4~~ Required modules can be removed

TOPIC	STATUS	IMPACT	LIKELIHOOD
Spec	Fixed	Medium	Low

Orchestrator_v1.sol requires that there always is an Authorizer, Funding Manager, and Payment Processor module set, with other nonessential modules able to be added or removed more freely. This is done by having specific functions to update these essential modules, like for the Authorizer:

```
/// @inheritdoc IOrchestrator_v1
function initiateSetAuthorizerWithTimelock(IAuthorizer_v1 authorizer_)
    external
    onlyOrchestratorOwner
{
    address authorizerContract = address(authorizer_);
    bytes4 moduleInterfaceId = type(IModule_v1).interfaceId;
    bytes4 authorizerInterfaceId = type(IAuthorizer_v1).interfaceId;
    if (
        ERC165Checker.supportsInterface(
            authorizerContract, moduleInterfaceId
        )
            && ERC165Checker.supportsInterface(
                authorizerContract, authorizerInterfaceId
            )
    ) {
        _initiateAddModuleWithTimelock(authorizerContract);
        _initiateRemoveModuleWithTimelock(address(authorizer));
    } else {
        revert Orchestrator__InvalidModuleType(address(authorizer_));
    }
}

/// @inheritdoc IOrchestrator_v1
function executeSetAuthorizer(IAuthorizer_v1 authorizer_)
    external
    onlyOrchestratorOwner
{
    _executeAddModule(address(authorizer_));
    _executeRemoveModule(address(authorizer));
    authorizer = authorizer_;
    emit AuthorizerUpdated(address(authorizer_));
}

/// @inheritdoc IOrchestrator_v1
```

```
function cancelAuthorizerUpdate(IAuthorizer_v1 authorizer_)
    external
    onlyOrchestratorOwner
{
    _cancelModuleUpdate(address(authorizer_));
}
```

Reference: Orchestrator_v1.sol#L208-L248

Removal of one of these modules using the normal functions is prevented by explicitly checking if the module is essential:

```
/// @inheritdoc IOrchestrator_v1
function initiateRemoveModuleWithTimelock(address module_) external {
    // Revert given module to be removed is equal to current authorizer
    if (module_ == address(authorizer)) {
        revert Orchestrator__InvalidRemovalOfAuthorizer();
    }
    // Revert given module to be removed is equal to current fundingManager
    if (module_ == address(fundingManager)) {
        revert Orchestrator__InvalidRemovalOfFundingManager();
    }
    // Revert given module to be removed is equal to current paymentProcessor
    if (module_ == address(paymentProcessor)) {
        revert Orchestrator__InvalidRemovalOfPaymentProcessor();
    }
    _initiateRemoveModuleWithTimelock(module_);
}
```

Reference: Orchestrator_v1.sol#L342-L357

However, if `initiateSetAuthorizerWithTimelock()` is called, it will call the same function `_initiateRemoveModuleWithTimelock()` without the above checks, which after a timelock period, the normal executeRemoveModule() can be called, removing the module without replacing it.

Remediations to Consider

In `executeRemoveModule()`, check if the module is an essential module, as is done in `initiateRemoveModuleWithTimelock()` to ensure that there is always one of each essential module.

~~M-5~~ Queued stakers can receive rewards from pending assertions

TOPIC	STATUS	IMPACT	LIKELIHOOD
Spec	Fixed	Medium	Low

In `LM_PC_KPIRewarder_v1.sol`, a queue of stakers is created to prevent new stakers from receiving the rewards of KPI assertions that are pending verification. Without this queue, users could frontrun the verification, stake to earn rewards then unstake in the next block. As mentioned in the contract description, the stakers are queued until the next assertion is resolved. However, this assumes that there is only ever 1 assertion pending a resolution. If an asserter were to call `postAssertion()` while there is another pending assertion, then the stakers queued would be added and eligible for rewards for both the new and prior pending assertion, which is not desired.

Remediations to Consider

Require that there is only ever 1 pending assertion to ensure fairness for stakers.

~~M-6~~ Asserter can post invalid assertions with minimal penalty

TOPIC	STATUS	IMPACT	LIKELIHOOD
Incentive Design	Fixed	Medium	Low

In `OptimisticOracleIntegrator.sol` and contracts that inherit it like `LM_PC_KPIRewarder_v1.sol`, the minimum bond is used to make an assertion:

```
        function assertDataFor(bytes32 dataId, bytes32 data, address asserter)
         public
         virtual
         onlyModuleRole(ASSERTER_ROLE)
         returns (bytes32 assertionId)
    {
         asserter = asserter == address(0) ? _msgSender() : asserter;
         uint bond = oo.getMinimumBond(address(defaultCurrency));
         ...
```

Reference: OptimisticOracleIntegrator.sol#L152-L159

This bond is returned if the assertion is not successfully disputed but is lost otherwise. As mentioned in the Uma docs: “If data received from the oracle could potentially move large amounts of value, you may want to set a higher bond to make it more costly for an attacker to attempt to steal funds through a false proposal.“ In the case where the `ASSERTER_ROLE` acts maliciously, they could post multiple proposals with invalid data, potentially with inflated KPI, with minimal loss on a dispute. Since the bond is low, and a disputer receives half the bond as an incentive, an assertion with a lower bond is less likely to be disputed, as it may not be worth disputing, so an invalid assertion may end up passing.

Remediations to Consider

Consider using a higher bond than the minimum. Since the result of the data asserted can have a lot of funds associated with it, a higher bond will increase the security and confidence the data is accurate.

~~M-7~~ Staking rounding errors

TOPIC	STATUS	IMPACT	LIKELIHOOD
Rounding	Fixed	Medium	High

In `LM_PC_Staking_v1.sol`, users stake assets and receive rewards over time based on their proportion of total staked assets and the set reward and duration.

The amount of rewards a user is owed is calculated for the most part in the functions `_calculateRewardValue()` and `_earned()`:

```
///@dev This is the heart of the algorithm
/// The reward Value is the accumulation of all the rewards a user would get for a single
/// A "single" reward value or with the lack of a better word "reward period" is the rewar
/// multiplied by the time period it was active and dividing that with the total supply
/// This "single" value is essentially what a single token would have earned in that time
function _calculateRewardValue() internal view returns (uint) {
    //In case the totalSupply is 0 the rewardValue doesnt change
    if (totalSupply == 0) {
        return rewardValue;
    }

    return (_getRewardDistributionTimestamp() - lastUpdate) //Get the time difference betv
        * rewardRate //Multiply it with the rewardrate to get the rewards distributed for
```

```
        * 1e18 // for the later division we need a value to compensate for the loss of pre
        / totalSupply //divide it by the totalSupply to get the rewards per token
        + rewardValue; //add the old rewardValue to the new "single" rewardValue
}
```

Reference: LM_PC_Staking_v1.sol#L237-L253

```
 /// @dev internal function to calculate how much a user earned for their stake up to this
/// Uses the difference between the current Reward Value and the reward value when the use
/// in combination with their current balance to calculate their earnings
function _earned(address user, uint providedRewardValue)
    internal
    view
    returns (uint)
{
    return (providedRewardValue - userRewardValue[user]) //This difference in rewardValues
        * _balances[user] // multiply by users balance of tokens to get their share of the
        / 1e18 // See comment in _calculateRewardValue();
        + rewards[user];
}
```

Reference: LM_PC_Staking_v1.sol#L262-L274

As mentioned in the comments of these functions, a value of `1e18` is multiplied in `_calculateRewardValue()` and divided out in `_earned()` in order to maintain precision.

However, this value of `1e18` is not sufficient to completely maintain precision when the total staked supply is sufficiently large relative to the current rewards rate. In the current state with reasonable values, there is a minor loss of precision. The value 1e18 has a fair bit of room to increase without overflow risk for reasonable values of `rewardRate` and `totalSupply`, and the larger this value is, the more precision that can be maintained

Remediations to Consider

Increase this precision value to e.g. `1e36` to reduce loss of precision for user rewards.

~~M-8~~ Orchestrator should not be its own owner

TOPIC	STATUS	IMPACT	LIKELIHOOD
Authorization	Fixed	High	Low

In the case where the Orchestrator itself is set to have the OrchestratorOwner role, it can end up leading to modules calling functions on other modules as if they were the OrchestratorOwner. This is because modules are able to call executeTxFromModule(), allowing calls with arbritrary data to be made by the orchestrator:

```
/// @inheritdoc IModuleManagerBase_v1
function executeTxFromModule(address to, bytes memory data)
    external
    virtual
    onlyModule
    returns (bool, bytes memory)
{
    bool ok;
    bytes memory returnData;

    (ok, returnData) = to.call(data);

    return (ok, returnData);
}
```

Reference: ModuleManagerBase_v1.sol#L258-L271

This allows for modules to be added that could override the authorization scheme, potentially allowing anyone to call permissioned functions on other modules with any data.

Remediations to Consider

Add checks in authorizers to ensure that the orchestrator cannot have the `ORCHESTRATOR_OWNER_ROLE`.

~~M-9~~ FM_BC_Bancor_Redeeming_VirtualSupply_v1 may get initialized with incompatible tokens

TOPIC	STATUS	IMPACT	LIKELIHOOD
Input validation	Fixed	High	Low

In the `FM_BC_Bancor_Redeeming_VirtualSupply_v1` init() function setup of collateral and issuance token is performed. To guarantee that these tokens have a proper number of decimals, their decimals are compared and validated within `_setIssuanceToken()`.

However, `_setIssuanceToken()` is called and validation is performed before `collateralTokenDecimals` is set. As a result, at the time of expression evaluation, `collateralTokenDecimals` is 0. Therefore, it is possible to set an issuance token with a number of decimals less than the `collateralTokenDecimals`. This is not according to the implementation intention and may cause incorrect results.

Remediations to consider:

- Change the order of operations in init(), and initialize `collateralTokenDecimals` before invoking `_setIssuanceToken()`

```
// Set issuance token. This also caches the decimals
_setIssuanceToken(address(_issuanceToken));

// Set accepted token
_token = IERC20(_acceptedToken);

// Cache token decimals for collateral
collateralTokenDecimals = IERC20Metadata(address(_token)).decimals();
```

~~M-10~~ Implemented access control does not match those described in natspec

TOPIC	STATUS	IMPACT	LIKELIHOOD
Spec	Fixed	Low	High

In the `IFM_BC_Bancor_Redeeming_VirtualSupply_v1` following functions are meant to be accessible by the Orchestrator owner and Manager:

- setReserveRatioForBuying()
- setReserveRatioForSelling()

In the `IRedeemingBoundingCurveBase_v1` both Orchestrator owner and Manager should have access to the following functions:

- openSell()

- closeSell()
- setSellFee()

In the `IBondingCurveBase_v1` Orchestratror owner and Manager should be able to invoke:

- openBuy()
- closeBuy()
- setBuyFee()

However in corresponding implementations for these interfaces none of the functions listed above is accessible to the Manager.

Remediations to consider

- Update implementation contracts to use different access control modifier which allows Manager to invoke these functions, or
- Update natspec and remove mention of Manager capabilities in reference to these functions.

~~L-1~~ BondingCurveBase_v1 does not implement the funding manager interface

TOPIC	STATUS	IMPACT	LIKELIHOOD
Protocol Design	Fixed	Low	Low

`BondingCurveBase_v1.sol` is intended to be a funding manager, and declares that it supports the `IFundingManager` interface:

```
...
abstract contract BondingCurveBase_v1 is IBondingCurveBase_v1, Module_v1 {
    function supportsInterface(bytes4 interfaceId)
        public
        view
        virtual
        override(Module_v1)
        returns (bool)
    {
```

```
        return interfaceId == type(IBondingCurveBase_v1).interfaceId || interfaceId == typ
            || super.supportsInterface(interfaceId);
    }
...
```

Reference: BondingCurveBase_v1.sol#L35-L46

However, it does not actually implement the `IFundingManager` interface, and has no implemented or abstract functions that it requires. If downstream contracts do not implement the expected functions while indicating that they do, it may cause these contracts to be added as funding managers and pass the checks in the orchestrator when they shouldn't:

```
...
bytes4 fundingManagerInterfaceId = type(IFundingManager_v1).interfaceId;
 if (
     ERC165Checker.supportsInterface(
         fundingManagerContract, moduleInterfaceId
     )
         && ERC165Checker.supportsInterface(
             fundingManagerContract, fundingManagerInterfaceId
         )
 ) {
 ...
```

Reference: Orchestrator_v1.sol#L241-L249

Remediations to Consider

- Add abstract versions of the `token()` and `transferOrchestratorToken()` functions to `BondingCurveBase_v1` to ensure any downstream contracts implement this functions, or
- Remove out support for `IFundingManager` interface in `BondingCurveBase_v1` and leave to child contracts to properly declare that support on lower level in the contract inheritance chain.

~~t-2~~ Changing modules fails if at module limit

TOPIC	STATUS	IMPACT	LIKELIHOOD
Error Recovery	Fixed	Low	Low

In `Orchestrator_v1`, there are 3 required modules, the funding manager, payment processor, and the authorizer, which are set up on initialization and can be directly queried. Although additional modules can be added and removed, these modules are special as they always need to be present, but they can be replaced by the orchestrator owner with a special but similar setter for each.

```
/// @inheritdoc IOrchestrator_v1
function setFundingManager(IFundingManager_v1 fundingManager_)
    external
    onlyOrchestratorOwner
{
    address fundingManagerContract = address(fundingManager_);
    bytes4 moduleInterfaceId = type(IModule_v1).interfaceId;
    bytes4 fundingManagerInterfaceId = type(IFundingManager_v1).interfaceId;
    if (
        ERC165Checker.supportsInterface(
            fundingManagerContract, moduleInterfaceId
        )
            && ERC165Checker.supportsInterface(
                fundingManagerContract, fundingManagerInterfaceId
            )
    ) {
            ///@audit: notice the new funding manager is added before the prior was remove
        addModule(address(fundingManager_));
        removeModule(address(fundingManager));
        fundingManager = fundingManager_;
        emit FundingManagerUpdated(address(fundingManager_));
    } else {
        revert Orchestrator__InvalidModuleType(address(fundingManager_));
    }
}
```

Reference: Orchestrator_v1.sol#L234-L257

However, when these functions are called, the new module is attempted to be added via the `addModule()` function before the prior one is removed. In most cases, this is fine, but in the case where the number of modules is at the `MAX_MODULE_AMOUNT` of 128, this causes a revert since `addModule()` calls `moduleLimitNotExceeded` before adding:

```
/// @inheritdoc IModuleManagerBase_v1
function addModule(address module)
    public
    __ModuleManager_onlyAuthorized
    moduleLimitNotExceeded
    isNotModule(module)
```

```
    validModule(module)
{
    _commitAddModule(module);
}
```

Reference: ModuleManagerBase_v1.sol#L180-L189

```
modifier moduleLimitNotExceeded() {
    if (_modules.length >= MAX_MODULE_AMOUNT) {
        revert ModuleManagerBase__ModuleAmountOverLimits();
    }
    _;
}
```

Reference: ModuleManagerBase_v1.sol#L77-L82

This prevents updating one of these special modules without first removing another module if the module limit is reached.

Remediations to Consider

Reorder the calls in `setAuthorizer()`, `setFundingManager()` and `setPaymentProcessor()` to call `removeModule()` before `addModule()` to cover this edge case.

~~L-3~~ Orchestrator's required modules interfaces are unchecked on initialization

TOPIC	STATUS	IMPACT	LIKELIHOOD
Error Recovery	Fixed	Low	Low

In `Orchestrator_v1.sol`, when it is initialized via the init() function, it adds its `fundingManager`, `paymentProcessor`, and `authorizer` module directly, calling `__ModuleManager_addModule()` for each:

```
fundingManager = fundingManager_;
authorizer = authorizer_;
paymentProcessor = paymentProcessor_;

governor = governor_;
```

```
// Add necessary modules.
// Note to not use the public addModule function as the factory
// is (most probably) not authorized.
__ModuleManager_addModule(address(fundingManager_));
__ModuleManager_addModule(address(authorizer_));
__ModuleManager_addModule(address(paymentProcessor_));
```

Reference: Orchestrator_v1.sol#L115-L126

However, if any of these modules is changed it calls it's own function that explicitly requires that the replacement module supports the required interface:

```
function initiateSetFundingManagerWithTimelock(
    IFundingManager_v1 fundingManager_
) external onlyOrchestratorOwner {
    address fundingManagerContract = address(fundingManager_);
    bytes4 moduleInterfaceId = type(IModule_v1).interfaceId;
    bytes4 fundingManagerInterfaceId = type(IFundingManager_v1).interfaceId;
    if (
        ERC165Checker.supportsInterface(
            fundingManagerContract, moduleInterfaceId
        )
            && ERC165Checker.supportsInterface(
                fundingManagerContract, fundingManagerInterfaceId
            )
    ) {
        _initiateAddModuleWithTimelock(address(fundingManager_));
        _initiateRemoveModuleWithTimelock(address(fundingManager));
    } else {
        revert Orchestrator__InvalidModuleType(address(fundingManager_));
    }
}
```

Reference: Orchestrator_v1.sol#L252-L271

Since the interface is not directly checked, it could lead to improper initial setup, using a contract that does may implement the intended functions.

Remediations to Consider

Check if the required modules support the required interfaces, as is done when updating these modules, to prevent errors and be more consistent.

~~I-4~~ Module manager does not check if added modules support `IModule_v1`

interface

TOPIC	STATUS	IMPACT	LIKELIHOOD
Error Recovery	Addressed	Low	Low

`ModuleManagerBase_v1.sol` is inherited by `Orchestrator_v1.sol`, and manages the modules it controls, ensuring they are valid and there are not duplicates among other responsibilities. However, when adding a module, it checks if the new module is valid by calling the `validModule` modifier, which calls `_ensureValidModule()`:

```
function __ModuleManager_addModule(address module)
        internal
        isNotModule(module)
        validModule(module)
    {
        _commitAddModule(module);
    }
```

Reference: ModuleManagerBase_v1.sol#L158-L164

```
modifier validModule(address module) {
    _ensureValidModule(module);
    _;
}
```

Reference: ModuleManagerBase_v1.sol#L60-L63

```
function _ensureValidModule(address module) private view {
        ///@audit: it does not check if the address supports the IModule interface which t
    if (module == address(0) || module == address(this)) {
            revert ModuleManagerBase__InvalidModuleAddress();
        }
    }
```

Reference: ModuleManagerBase_v1.sol#L316-L320

However, `_ensureValidModule()` only checks if the address is valid and does not ensure the new module supports the `IModule` interface. This could lead to errors if the protocol expects to interact with these functions on this module.

Remediations to Consider

In `_ensureValidModule()`, check if the new module supports the `IModule_v1` interface to ensure the module can be interacted with as expected.

~~L-5~~ Missing validation for `assertionLiveness` in OptimisticOracleIntegrator

TOPIC	STATUS	IMPACT	LIKELIHOOD
Input Validation	Fixed	Medium	Low

In the `OptimisticOracleIntegrator`, the `assertionLiveness` setting defines the time period within which an assertion is open for dispute (in seconds). This configuration setting can be updated on initialization in `__OptimisticOracleIntegrator_init()` or using `setDefaultAssertionLiveness()`, which is accessible only to the orchestrator owner.

Currently, there is only validation that the provided value is not 0. However, if `assertionLiveness` is set to some relatively small value, such as 1 second, that would not allow meaningful system operation.

Remediations to Consider

Consider adding validation for a valid minimum period for `assertionLiveness` config changes.

~~L-6~~ Metadata id hash can conflict

TOPIC	STATUS	IMPACT	LIKELIHOOD
Protocol Design	Fixed	Medium	Low

In `LibMetadata`, a module's metadata is hashed to create a "unique" id or `identifier()`:

```
 /// @dev Returns the identifier for given metadata.
/// @param metadata The metadata.
/// @return The metadata's identifier.
function identifier(IModule_v1.Metadata memory metadata)
```

```
    internal
    pure
    returns (bytes32)
{
    return keccak256(
        abi.encodePacked(
            metadata.majorVersion, metadata.url, metadata.title
        )
    );
}
```

Reference: LibMetadata.sol#L14-L27

However, this identifier may not be unique since the metadata values are encoded with `abi.encodePacked()` and the values url and title are strings which are dynamic values. When dynamic values are encoded with `abi.encodePacked()`, the first word that determines its length is removed, and only the content is encoded. This has the effect of when multiple dynamic values are encoded and packed next to each other; different values can encode to the same thing. For example: `abi.encodePacked("someUrl", "module title") == abi.encodePacked("some", "Urlmodule title")`

This can lead to modules being created in the ModuleFactory with unintended metadata, which could invalidate URLs and titles.

Remediations to Consider

Instead of using `abi.encodePacked()` to hash the metadata values in the identifier, use `abi.encode()` to preserve the length data and ensure all unique metadata values also have a unique identifier with no collisions.

~~L-7~~ Interface getter functions not set as view

TOPIC	STATUS	IMPACT	LIKELIHOOD
Interoperability	Fixed	Medium	Low

Throughout the protocol there are multiple external calls to other contracts that are expected to support particular interfaces. However, some interfaces describe functions that are clearly always intended to be getter view functions, but are not set as view. This could cause contracts that are setup within the protocol that implement these functions but when called have the ability to take over execution in unintended ways.

Remediations to Consider

Make sure that each interface function which is intended to be read only is explicitly set as view.

~~L-8~~ Removal of token gated role leaves stale data

TOPIC	STATUS	IMPACT	LIKELIHOOD
Interoperability	Fixed	Medium	Low

In the `AUT_TokenGated_Roles_v1`, `grantTokenRoleFromModule()` is used to configure a specific role. While parent contract `AUT_Roles_v1` features capabilities to `grantRoleFromModule()` and `revokeRoleFromModule()`, token gated child contract does not feature a specific function for removing the token gated role.

However, if parent `revokeRoleFromModule()` is utilized `thresholdMap[thresholdId]` associated with the particular role and token will remain set and potentially reused if role is granted again to the same token in the future.

Remediations to Consider

- Override `revokeRoleFromModule()` in the `AUT_TokenGated_Roles_v1` to perform necessary cleanup actions in addition to those performed in the parent contract's implementation.

~~L-9~~ Token interface compliance check can be easily circumvented

TOPIC	STATUS	IMPACT	LIKELIHOOD
Interoperability	Fixed	Medium	Low

In the `AUT_TokenGated_Roles_v1`, `_grantRole()` implementation performs check if provided address is a Token.

```
if (isTokenGated[role]) {
    try TokenInterface(account).balanceOf(address(this)) {}
    catch {
        revert Module__AUT_TokenGated_Roles__InvalidToken(account);
    }
}
```

However, the current check will pass for any EOA address or for any other contract with a `fallback()` method, which will not result in a revert/exception that will be caught in the catch block.

Remediations to Consider

Consider removing this check, which may result in a false sense of security, or making this guard condition more restrictive.

~~L-10~~ Events related to different orders may report same wallet ID

TOPIC	STATUS	IMPACT	LIKELIHOOD
Events	Fixed	Medium	Low

In the `PP_Streaming_v1` contract, when `processPayments()` is invoked, pending orders are iterated through and processed individually. In addition to their corresponding data attributes, each order's `_walletId` is initialized based on the current counter specific to the particular client recipient combination.

```
...
_walletId = numVestingWallets[address(client)][_recipient] + 1;

_addPayment(
    address(client),
    _recipient,
    _amount,
    _start,
    _dueTo,
    _walletId
);

emit PaymentOrderProcessed(
    address(client),
    _recipient,
```

```
        _amount,
        _start,
        _dueTo,
        _walletId
    );
```

When validation passes, the specific counter is incremented, and the `StreamingPaymentAdded()` event is emitted.

However, when validation fails in the `_addPayment()` function, the function does not revert but emits an `InvalidStreamingOrderDiscarded()` event. In this case, it doesn't increment a specific counter on which the `_walletId` value is based.

Because of this slightly different behavior, it is possible for multiple orders for the same recipient to fail, and they will emit multiple `PaymentOrderProcessed()` events with the same `_walletId` value. Moreover, a case exists where multiple failures and a single successfully processed order each emit a `PaymentOrderProcessed` event with the same walletId value. As a result, off-chain tracking and monitoring tools may not operate properly unless aware of this edge case.

Remediations to consider

- Update PaymentOrder data struct to maintain unique `orderId` across various stages of processing, or
- Increment `numVestingWallets[client][_paymentReceiver]` even when validation fails in `_addPayment()`.

~~L-11~~ Default visibility of _isModule variable may break contract boundaries

TOPIC	STATUS	IMPACT	LIKELIHOOD
Spec	Fixed	Low	Low

In the `ModuleManagerBase_v1` contract, the `_isModule` variable is defined without an explicit visibility specifier. The default visibility for this variable, in that case, would be `internal`.

```
mapping(address => bool) _isModule;
```

As a result, contracts inheriting from the `ModuleManagerBase_v1` would be able to directly manipulate `_isModule` mapping instead of using or overriding corresponding functions from the `ModuleManagerBase_v1`.

Remediations to consider

Update the variable definition to include a `private` visibility specifier so it cannot be directly updated without the associated functionality present in `ModuleManagerBase_v1` being executed.

~~Q-1~~ Unnecessary type casting

TOPIC	STATUS	QUALITY IMPACT
Code Quality	Fixed	Low

In some areas there is type casting from one type to another, that isn't necessarily. For example in Orchestrator_v1.sol's `initiateSetFundingManagerWithTimelock()` function, there is a local fundingManagerContract address variable created that is casted from IFundingManager_v1 to an address, but it is never used, and it is continually directly casted to an address where required:

```
function initiateSetFundingManagerWithTimelock(
    IFundingManager_v1 fundingManager_
) external onlyOrchestratorOwner {
        ///@audit: local value is created, but never used
    address fundingManagerContract = address(fundingManager_);
    bytes4 moduleInterfaceId = type(IModule_v1).interfaceId;
    bytes4 fundingManagerInterfaceId = type(IFundingManager_v1).interfaceId;
    if (
        ERC165Checker.supportsInterface(
            fundingManagerContract, moduleInterfaceId
        )
            && ERC165Checker.supportsInterface(
                fundingManagerContract, fundingManagerInterfaceId
            )
    ) {
            ///@audit: could be used here instead of recasting
        _initiateAddModuleWithTimelock(address(fundingManager_));
        _initiateRemoveModuleWithTimelock(address(fundingManager));
    } else {
            ///@audit: as well as here
```

```
            revert Orchestrator__InvalidModuleType(address(fundingManager_));
        }
    }
```

Reference: Orchestrator_v1.sol#L252-L271

A Similar issue is found in `initiateSetAuthorizerWithTimelock()` and `initiateSetPaymentProcessorWithTimelock()`.

Remediations to Consider

Use the local variable setup in these functions to prevent recasting and duplicating code.

~~Q-2~~ Unnecessary callback function declaration in IOptimisticOracleIntegrator

TOPIC	STATUS	QUALITY IMPACT
Best practices	Fixed	Low

In the `IOptimisticOracleIntegrator`, `assertionResolvedCallback()` and `assertionDisputedCallback()` are declared explicitly, which is unnecessary since `IOptimisticOracleIntegrator` inherits from the `OptimisticOracleV3CallbackRecipientInterface` which declares these same callback functions.

Consider removing the explicit declaration of `assertionResolvedCallback()` and `assertionDisputedCallback()` from the `IOptimisticOracleIntegrator`.

~~Q-3~~ Missing function declaration in the interface

TOPIC	STATUS	QUALITY IMPACT
Best practices	Fixed	Low

The `claimRewards()` function implementation is present in the LM_PC_Staking_v1 contract. However, the corresponding interface, `ILM_PC_Staking_v1`, lacks this function declaration.

~~Q-4~~ Unused errors

TOPIC	STATUS	QUALITY IMPACT
Unnecessary code	Fixed	Low

- In `ILM_PC_KPIRewarder_v1`

```
error Module__LM_PC_KPIRewarder_v1__InvalidTargetValue();
```

- In `IModuleManagerBase_v1`

```
error ModuleManagerBase__ModulesNotConsecutive();
```

- In `IFM_BC_Bancor_Redeeming_VirtualSupply_v1`

```
error Module__FM_BC_Bancor_Redeeming_VirtualSupply__InvalidDepositAmount();
```

RESPONSE BY INVERTER

> Follow-up fix:
> https://github.com/InverterNetwork/contracts/pull/475/commits/68507a2f91c9998b6a0cc5c5187a7514a2bcd8ee

~~Q-5~~ Unused events

TOPIC	STATUS	QUALITY IMPACT
Events	Fixed	Low

- In `ILM_PC_Staking_v1` the following event is defined but never used

```
 /// @notice Event emitted when the reward duration is updated.
event RewardsDurationUpdated(uint newDuration);
```

~~Q-6~~ Events without indexed attributes

TOPIC	STATUS	QUALITY IMPACT
Events	Fixed	Low

- In `ILM_PC_KPIRewarder_v1`, the following events may use indexed attributes
  - `FeeFundsDeposited` (token)
  - `StakeEnqueued` (user)
  - `StakeDequeued` (user)
  - `KPICreated` (KPI_Id)

~~Q-7~~ Unnecessary Imports

TOPIC	STATUS	QUALITY IMPACT
Unnecessary code	Fixed	Low

- In `IInverterBeacon_v1.sol`, it imports IModule_v1 and IOrchestrator_v1
- In `LM_PC_KPIRewarder_v1`

```
 - import "forge-std/console.sol";

    import {
        ILM_PC_Staking_v1,
        LM_PC_Staking_v1,
        SafeERC20,
        IERC20,
        ERC20PaymentClientBase_v1,
-       IERC20PaymentClientBase_v1,
-       ReentrancyGuard
    } from "./LM_PC_Staking_v1.sol";

  import {
```

```
     IOptimisticOracleIntegrator,
     OptimisticOracleIntegrator,
-    OptimisticOracleV3CallbackRecipientInterface,
-    OptimisticOracleV3Interface,
-    ClaimData
  } from
    "src/modules/logicModule/abstracts/oracleIntegrations/UMA_OptimisticOracleV3/Opti
```

- In `ILM_PC_KPIRewarder_v1`

```
import {
    ILM_PC_Staking_v1,
    LM_PC_Staking_v1,
    SafeERC20,
    IERC20,
    IERC20PaymentClientBase_v1,
    ReentrancyGuard
} from "src/modules/logicModule/LM_PC_Staking_v1.sol";

import {
    IOptimisticOracleIntegrator,
    OptimisticOracleIntegrator,
    OptimisticOracleV3CallbackRecipientInterface,
    OptimisticOracleV3Interface,
    ClaimData
} from
    "@lm/abstracts/oracleIntegrations/UMA_OptimisticOracleV3/OptimisticOracleIntegrat
```

- In `OptimisticOracleIntegrator`

```
 - import {OptimisticOracleV3CallbackRecipientInterface} from "@lm/abstracts/oracleIn
- import {ERC165Checker} from "@oz/utils/introspection/ERC165Checker.sol";
```

- In `ERC20PaymentClientBase_v1`

```
import {
        Module_v1,
-    ContextUpgradeable
} from "src/modules/base/Module_v1.sol";
```

- In `BondingCurveBase_v1`

```
import {IOrchestrator_v1} from
    "src/orchestrator/interfaces/IOrchestrator_v1.sol";
```

- In `FM_BC_Bancor_Redeeming_VirtualSupply_v1`

```
import {IBondingCurveBase_v1} from
    "@fm/bondingCurve/interfaces/IBondingCurveBase_v1.sol";
import {IRedeemingBondingCurveBase_v1} from
    "@fm/bondingCurve/interfaces/IRedeemingBondingCurveBase_v1.sol";
```

Q-8 Utils.sol unused

TOPIC	STATUS	QUALITY IMPACT
Unnecessary code	Wont Do	Low

None of the functions from the `src/modules/fundingManager/bondingCurve/formulas/Utils.sol` are being used even though `BancorFormula` contract inherits from Utils.

Consider removing `Utils.sol` and reference to it in the `BancorFormula` contract.

RESPONSE BY INVERTER

> The unused `Utils.sol` contract is intentionally retained as part of the unmodified, battle-tested Aragon BancorFormula implementation. We did not want to modify the original implementation at all to be as safe as possible. This approach ensures we maintain the full integrity of the audited code.

~~Q-9~~ Unused code in BondingCurveBase_v1

TOPIC	STATUS	QUALITY IMPACT
Unnecessary code	Addressed	Low

Function `_calculateNetAmountAndFee()` is not used. Consider removing it.

RESPONSE BY INVERTER

> The unused function has been removed in a separate pull request (PR #467:

> https://github.com/InverterNetwork/contracts/pull/467). During this PR, we also noticed that the function wasn't being used and subsequently removed it. The PR addresses both the unused code issue and optimizes contract size.

~~Q-10~~ Avoid conditional expressions with bool literals

TOPIC	STATUS	QUALITY IMPACT
Unnecessary code	Fixed	Low

Bool variables do not need to be compared with literal true/false values.

For example:

```
// replace
if (buyIsOpen == false) {
// with
if (!buyIsOpen) {
```

Instances of this present in multiple contracts:

- BondingCurveBase_v1 - `buyingIsEnabled()` , or `_openBuy()` , `_closeBuy()`
- RedeemingBondingCurveBase_v1 - `sellingIsEnabled()` , or `_openSell()` , `_closeSell()`
- FM_BC_Bancor_Redeeming_VirtualSupply_v1 - `init()`

~~Q-11~~ Unnecessary duplicate function invocation

TOPIC	STATUS	QUALITY IMPACT
Unnecessary code	Fixed	Low

In `Module_v1` and `OrchestratorFactory_v1` , `_dependencyInjectionRequired()` function has following implementation:

```
function _dependencyInjectionRequired(bytes memory dependencyData)
    internal
    view
    returns (bool)
{
    try this.decoder(dependencyData) returns (bool) {
        return this.decoder(dependencyData);
    } catch {
        return false;
    }
}
```

Consider updating to the following to avoid duplicate `decoder()` function calls

```
function _dependencyInjectionRequired(bytes memory dependencyData)
    internal
    view
    returns (bool)
{
    try this.decoder(dependencyData) returns (bool requirement) {
        return requirement;
    } catch {
        return false;
    }
}
```

RESPONSE BY INVERTER

> We have addressed this issue in PR #467 (https://github.com/InverterNetwork/contracts/pull/467). During our review, we noticed that the underlying functionality of init2 / dependency injection was no longer being used. As a result, we removed this entire feature, including the function in question, thus resolving the duplicate function invocation issue and simplifying our codebase.

~~Q-12~~ Unnecessary variable

TOPIC	STATUS	QUALITY IMPACT
Unnecessary code	Fixed	Low

In `ModuleManagerBase_v1`, the following variable is defined but never used. Consider removing it.

```
/// @dev Mapping of modules and access control roles to accounts and
///      whether they holds that role.
/// @dev module address => role => account address => bool.
///
/// @custom:invariant Modules can only mutate own account roles.
/// @custom:invariant Only modules can mutate not own account roles.
/// @custom:invariant Account can always renounce own roles.
/// @custom:invariant Roles only exist for enabled modules.
mapping(address => mapping(bytes32 => mapping(address => bool))) private
    _moduleRoles;
```

~~G-1~~ Remove unnecessary indexed in Event declarations

TOPIC	STATUS	GAS SAVINGS
Gas optimization	Fixed	Low

In multiple interfaces events are declared with indexed attribute for uint variables representing different kinds of amounts. Indexing these attributes is not useful since rarely will they be used for searching. However, indexed attributes will result in less gas efficient event emission.

Therefore, consider removing indexed attributes across codebase for `uint` parameters in events (unless uint parameter acts as enum).

~~G-2~~ Cache cross contract reads

TOPIC	STATUS	GAS SAVINGS
Gas optimization	Fixed	Low

In `BondingCurveBase_v1:_buyOrder()` there are two instances of reading same cross contract variable.

```
// #1 instance
```

```
__Module_orchestrator.fundingManager().token().safeTransferFrom(
    _msgSender(), address(this), _depositAmount
);

// #2 instance
_processProtocolFeeViaTransfer(
    collateralTreasury,
    __Module_orchestrator.fundingManager().token(),
    protocolFeeAmount
);
```

In `RedeemingBondingCurveBase_v1:_sellOrder()` there are three instances of reading same cross contract variable.

```
// #1 instance
if (
    (collateralRedeemAmount + projectCollateralFeeCollected)
        > __Module_orchestrator.fundingManager().token().balanceOf(
            address(this)
        )
) {

// #2 instance
_processProtocolFeeViaTransfer(
    collateralreasury,
    __Module_orchestrator.fundingManager().token(),
    protocolFeeAmount
);

// #3 instance
__Module_orchestrator.fundingManager().token().transfer(
    _receiver, collateralRedeemAmount
);
```

Consider caching call `__Module_orchestrator.fundingManager().token()` for more gas efficient execution.

~~I-1~~ Fee on transfer tokens not supported

TOPIC	STATUS	IMPACT
Standards	Addressed	Informational ✳

The `LM_PC_Staking_v1` module's implemented functionality does not properly handle fee-on-transfer tokens, so they are not currently supported. For example, see `stake()`

function.

RESPONSE BY INVERTER

> We acknowledge that the LM_PC_Staking_v1 module does not support fee-on-transfer tokens. This limitation is by design and has been clearly documented:
>
> (1) We've added explicit comments in the code to alert developers to this constraint.
> (2) Our user interface will display warnings to inform users about this limitation.
>
> We advise users to be aware of this restriction when interacting with or implementing the LM_PC_Staking_v1 module.

I-2 Modules can be upgraded or shutdown without warning

TOPIC	STATUS	IMPACT
Upgradeability	Acknowledged	Informational ✳

In `Governor_v1`, the `COMMUNITY_MULTISIG_ROLE` can directly call `forceUpgradeBeaconAndRestartImplementation()` which upgrades the implementation of a `InverterBeacon` contract, and thus upgrades all modules that use it as a beacon:

```
/// @inheritdoc IGovernor_v1
function forceUpgradeBeaconAndRestartImplementation(
    address beacon,
    address newImplementation,
    uint newMinorVersion
)
    external
    onlyRole(COMMUNITY_MULTISIG_ROLE)
    accessibleBeacon(beacon)
    validAddress(newImplementation)
{
    IInverterBeacon_v1(beacon).upgradeTo(
        newImplementation, newMinorVersion, true
    );
    emit BeaconForcefullyUpgradedAndImplementationRestarted(
        beacon, newImplementation, newMinorVersion
    );
}
```

Reference: Governor_v1.sol#L319-L336

This upgrade is not opt-in and may give users little time to react to the upgrade, depending on how the addresses with the `COMMUNITY_MULTISIG_ROLE` are set. Upgrades can also be set by users with either the `TEAM_MULTISIG_ROLE` or `COMMUNITY_MULTISIG_ROLE`, but it is set with a timelock delay before the upgrade can execute, which allows users to react. Modules can also be shutdown by either role, preventing interaction, with the intent being to mitigate the effect of discovered vulnerabilities so they can be patched.

Ensuring these permissioned wallets are themselves behind a timelock, are sufficiently secure, and transparent is advised. Users should become aware of these potentials before deciding to interact with protocols using this system.

RESPONSE BY INVERTER

> We acknowledge the importance of transparent and secure upgrade mechanisms. Our protocol implements a dual multisig approach to prioritize security without compromise:
>
> - The Community Multisig, composed of diverse stakeholders including community representatives, external advisors, and some team members (who do not hold a majority), has the authority to approve critical on-chain upgrades.
> - The Team Multisig, consisting of Inverter team members, handles routine operational tasks.
>
> For live contracts, we employ TimeLock mechanisms for major upgrades. This introduces a predetermined waiting period, allowing users to review changes and opt-out if desired. The duration of this period is carefully considered to balance urgency and user safety.
>
> Additionally, we provide users with the option to opt out of the automatic upgrade system entirely, empowering them to decide whether to adopt specific upgrades.
>
> These measures, detailed in our Security Guidelines, aim to protect users while maintaining the protocol's ability to address critical issues promptly. We remain committed to transparent communication about any potential changes or upgrades. To ensure full transparency, we will publish detailed information about these multisigs on our social media channels and in our documentation. This will allow the community to verify the composition and structure of these multisigs for themselves and perform their own checks.

~~I-3~~ Tokens with callbacks on transfers are not supported

TOPIC	STATUS	IMPACT
Standards	Addressed	Informational ✳

In the `PP_Streaming_v1` and `PP_Simple_v1` modules, token transfers are performed to external receivers. If token contracts support callbacks to external receivers, these receivers may intentionally cause order batches to revert.

Depending on how these failures are handled in concrete payment client implementation, this may block all pending payouts or cause some form of DoS, in which case it would be necessary to retry `processPayments()` calls.

RESPONSE BY INVERTER

> We acknowledge that the `PP_Streaming_v1` and `PP_Simple_v1` modules do not support tokens with callbacks on transfers. This limitation is by design and has been clearly documented:
>
> (1) We've added explicit comments in the code to alert developers to this constraint.
> (2) Our user interface will display warnings to inform users about this limitation.
>
> We advise users and integrators to be aware of this restriction when interacting with or implementing these modules.

~~I-4~~ OrchestratorOwner role has control over the protocol

TOPIC	STATUS	IMPACT
Trust model	Addressed	Informational ✳

The OrchestratorOwner role has the most control over the protocol, and depending on the modules set, it could steal users' assets by adjusting values by calling privileged functions. For the most part, privileged functions are moderated by timelocks that allow users to react to malicious updates, but there are functions in some current modules without these guards that could be taken advantage of. Users should be aware that the protocol is not completely free of centralization issues, and trust in privileged roles is required.

RESPONSE BY INVERTER

> The role of OrchestratorOwner is a crucial part of our protocol's design, and users do need to trust this role. This is an intentional design decision to allow for the necessary flexibility and management of workflows. However, we have implemented measures to limit its power and protect users. For instance, OrchestratorOwners cannot add or remove modules to a live workflow without going through a timelock period.
>
> While we acknowledge that some level of trust in privileged roles is required, we've designed the system to balance flexibility and security. We continuously work on reducing centralization risks where possible without compromising the protocol's ability to evolve and address critical issues.
>
> We encourage users to review our documentation and Security Guidelines for a comprehensive understanding of the protocol's trust model and security measures. We remain committed to transparency and will continue to communicate any changes or upgrades that affect the protocol's governance structure.

Disclaimer

Specifically, for the avoidance of doubt, this report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project. In this report you may through hypertext or other computer links, gain access to websites operated by persons other than Macro. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such websites' owners. You agree that Macro is not responsible for the content or operation of such websites, and that Macro shall have no liability to your or any other person or entity for the use of third party websites. Macro assumes no responsibility for the use of third party software and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.