Inverter Network Audit Competition on Hats.finance

Introduction to Hats.finance

Hats.finance builds autonomous security infrastructure for integration with major DeFi protocols to secure users' assets. It aims to be the decentralized choice for Web3 security, offering proactive security mechanisms like decentralized audit competitions and bug bounties. The protocol facilitates audit competitions to quickly secure smart contracts by having auditors compete, thereby reducing auditing costs and accelerating submissions. This aligns with their mission of fostering a robust, secure, and scalable Web3 ecosystem through decentralized security solutions.

About Hats Audit Competition

Hats Audit Competitions offer a unique and decentralized approach to enhancing the security of web3 projects. Leveraging the large collective expertise of hundreds of skilled auditors, these competitions foster a proactive bug hunting environment to fortify projects before their launch. Unlike traditional security assessments, Hats Audit Competitions operate on a time-based and results-driven model, ensuring that only successful auditors are rewarded for their contributions. This pay-for-results ethos not only allocates budgets more efficiently by paying exclusively for identified vulnerabilities but also retains funds if no issues are discovered. With a streamlined evaluation process, Hats prioritizes quality over quantity by rewarding the first submitter of a vulnerability, thus eliminating duplicate efforts and attracting top talent in web3 auditing. The process embodies Hats Finance's commitment to reducing fees, maintaining project control, and promoting high-quality security assessments, setting a new standard for decentralized security in the web3 space.

Inverter Network Overview

Inverter is a modular protocol for Primary Issuance Markets_ enabling maximum value capture from token economies

Competition Details

- Type: A public audit competition hosted by Inverter Network
- Duration: 2 weeks
- Maximum Reward: $54,164.51
- Submissions: 160
- Total Payout: $54,164.51 distributed among 55 participants.

Scope of Audit

Project Overview

Inverter Protocol is designed to provide a flexible and extensible way for any project or protocol to exchange assets between parties programmatically. Its specific focus is on enabling the issuance and distribution of tokens through Primary Issuance Markets (PIMs).

At its core, the Inverter Protocol consists of a modular architecture that seamlessly integrates different modules and existing protocols. This modular approach enables developers to create new modules that can be added to the protocol, allowing for an ever-expanding range of use cases and applications. The aforementioned PIMs employ algorithms to dynamically issue tokens based on real-time data and market conditions tailored to meet specific goals and KPIs relevant to each token's custom use case.

The Inverter Protocol aims to provide the ground on which to build a diverse range of applications and economies, from tokenization verticals as base-layer blockchains and dApp tokens to community currencies, from IP-NFTs and creative work to real-world assets and cooperatives, from micro-credit insurance pools to tokenized invoice-based SME receivables.

Documentation & Documents

We created an **onboarding video** for this audit, outlining the architecture, codebase and repository as a great starting point: Link

Further Documents

- Documentation: Link
- Technical Specification: Link
- Security Guideline: Link

Audit Competition Considerations

While our protocol is open to be used by anyone with any token, we communicate (in the contracts comments as well as on our UI) that there are certain tokens that will not work properly within our system and will lead to issues. These are:

- Tokens that change their balance without our contracts knowing explicitly (such as Fee on Transfer Tokens or Rebasing Tokens)
- Tokens that are using callback mechanisms, as these could (if abused/malicious) brick a workflow. As the selection of the specific token is up to the administrator of a workflow, the behavior of these specific tokens is acceptable for us, as we will clearly communicate any risks prior to them creating their workflow.

Audit Competition Scope

Below is a list of the contracts within the audit's scope. This includes **EVERY** contract within the `src/` folder.

```
src
├── common
│   └── LinkedIdList.sol
├── external
│   ├── fees
│   │   ├── FeeManager_v1.sol
│   │   └── interfaces
│   │       └── IFeeManager_v1.sol
│   ├── forwarder
│   │   └── interfaces
```

```
│   │       └── ITransactionForwarder_v1.sol
│   │   └── TransactionForwarder_v1.sol
│   ├── governance
│   │   └── interfaces
│   │       └── IGovernor_v1.sol
│   │   └── Governor_v1.sol
│   └── interfaces
│       └── IERC2771Context.sol
├── factories
│   ├── interfaces
│   │   ├── IModuleFactory_v1.sol
│   │   └── IOrchestratorFactory_v1.sol
│   ├── ModuleFactory_v1.sol
│   └── OrchestratorFactory_v1.sol
├── modules
│   ├── authorizer
│   │   ├── IAuthorizer_v1.sol
│   │   └── extensions
│   │       ├── AUT_EXT_VotingRoles_v1.sol
│   │   └── role
│   │       ├── AUT_Roles_v1.sol
│   │       ├── AUT_TokenGated_Roles_v1.sol
│   │       └── interfaces
│   │           └── IAUT_TokenGated_Roles_v1.sol
│   │           └── IAUT_EXT_VotingRoles_v1.sol
│   ├── base
│   │   ├── IModule_v1.sol
│   │   └── Module_v1.sol
│   ├── fundingManager
│   │   ├── bondingCurve
│   │   │   ├── abstracts
│   │   │   │   ├── BondingCurveBase_v1.sol
│   │   │   │   ├── RedeemingBondingCurveBase_v1.sol
│   │   │   │   ├── VirtualCollateralSupplyBase_v1.sol
│   │   │   │   └── VirtualIssuanceSupplyBase_v1.sol
│   │   │   ├── FM_BC_Bancor_Redeeming_VirtualSupply_v1.sol
│   │   │   ├── FM_BC_Restricted_Bancor_Redeeming_VirtualSupply_v1.sol
│   │   │   ├── FM_BC_Tools.sol
│   │   │   ├── formulas
│   │   │   │   ├── BancorFormula.sol
│   │   │   │   └── Utils.sol
│   │   │   ├── interfaces
│   │   │   │   ├── IBancorFormula.sol
│   │   │   │   ├── IBondingCurveBase_v1.sol
│   │   │   │   ├── IERC20Issuance_v1.sol
│   │   │   │   ├── IFM_BC_Bancor_Redeeming_VirtualSupply_v1.sol
│   │   │   │   ├── IRedeemingBondingCurveBase_v1.sol
│   │   │   │   ├── IVirtualCollateralSupplyBase_v1.sol
│   │   │   │   └── IVirtualIssuanceSupplyBase_v1.sol
│   │   │   └── tokens
│   │   │       └── ERC20Issuance_v1.sol
│   │   ├── IFundingManager_v1.sol
│   │   └── rebasing
│   │       ├── abstracts
```

```
│   │       │   ├── ElasticReceiptTokenBase_v1.sol
│   │       │   ├── ElasticReceiptTokenUpgradeable_v1.sol
│   │       │   └── ElasticReceiptToken_v1.sol
│   │       ├── FM_Rebasing_v1.sol
│   │       └── interfaces
│   │           ├── IERC20Metadata.sol
│   │           ├── IERC20.sol
│   │           └── IRebasingERC20.sol
│   ├── lib
│   │   ├── LibMetadata.sol
│   │   └── SafeMath.sol
│   ├── logicModule
│   │   ├── abstracts
│   │   │   ├── ERC20PaymentClientBase_v1.sol
│   │   │   └── oracleIntegrations
│   │   │       └── UMA_OptimisticOracleV3
│   │   │           ├── IOptimisticOracleIntegrator.sol
│   │   │           ├── OptimisticOracleIntegrator.sol
│   │   │           └── optimistic-oracle-v3
│   │   │               ├── AncillaryData.sol
│   │   │               ├── ClaimData.sol
│   │   │               └── interfaces
│   │   │                   ├──
OptimisticOracleV3CallbackRecipientInterface.sol
│   │   │                   └── OptimisticOracleV3Interface.sol
│   │   ├── interfaces
│   │   │   ├── IERC20PaymentClientBase_v1.sol
│   │   │   ├── ILM_PC_Bounties_v1.sol
│   │   │   ├── ILM_PC_PaymentRouter_v1.sol
│   │   │   ├── ILM_PC_KPIRewarder_v1.sol
│   │   │   ├── ILM_PC_RecurringPayments_v1.sol
│   │   │   └── ILM_PC_Staking_v1.sol
│   │   ├── LM_PC_Bounties_v1.sol
│   │   ├── LM_PC_PaymentRouter_v1.sol
│   │   ├── LM_PC_KPIRewarder_v1.sol
│   │   ├── LM_PC_RecurringPayments_v1.sol
│   │   └── LM_PC_Staking_v1.sol
│   ├── paymentProcessor
│   │   ├── interfaces
│   │   │   └── IPP_Streaming_v1.sol
│   │   ├── IPaymentProcessor_v1.sol
│   │   ├── PP_Simple_v1.sol
│   │   └── PP_Streaming_v1.sol
├── orchestrator
│   ├── abstracts
│   │   └── ModuleManagerBase_v1.sol
│   ├── interfaces
│   │   ├── IModuleManagerBase_v1.sol
│   │   └── IOrchestrator_v1.sol
│   └── Orchestrator_v1.sol
└── proxies
    ├── interfaces
    │   ├── IInverterBeacon_v1.sol
    │   ├── IInverterProxyAdmin_v1.sol
```

```
│   └── IInverterTransparentUpgradeableProxy_v1.sol
├── InverterBeacon_v1.sol
├── InverterBeaconProxy_v1.sol
├── InverterProxyAdmin_v1.sol
└── InverterTransparentUpgradeableProxy_v1.sol
```

Medium severity issues

- **Potential Fund Loss Due to Reorgs When Using `create` Opcode on Polygon**

  As the protocol plans to deploy on Polygon, where blockchain reorgs are frequent (with an average depth of 15-20 per day), caution is needed to address reorg-specific issues. The protocol uses factories to deploy orchestrators and modules, utilizing the `create` opcode, which considers only the factory nonce. This poses a risk because deployed modules may hold user funds. For instance, if a user deploys an orchestrator with a funding manager module and deposits funds in a subsequent transaction, a reorg could lead to fund mismanagement. An attacker could exploit this by front-running the deployment transaction and redirecting the funds to their control. It is suggested to use `create2` for deploying new modules, using `msg.sender` as the salt.

  Link: Issue #18

- **Rebalance Issue Causes Asset Loss for New Depositors After Transfers**

  The orchestrator in the project can transfer assets from the funding manager. To maintain accurate user balances, the `_rebase` function should adjust `_bitsPerToken` after each transfer. When the supply target is zero, `_rebase` does not update `_bitsPerToken`, leading to potential losses for new depositors. For instance, if there are 1000 active bits and a 500 supply target, `_bitsPerTokens` is 2. If the orchestrator moves 500 assets, reducing the supply target to zero, subsequent user deposits are not properly updated. A user depositing 1000 assets would not see `_bitsPerTokens` updated, and when redeeming, would get fewer assets back, as `_bitsPerTokens` updates only before burning. This discrepancy results in unfair asset distribution.

  Link: Issue #38

- **Vulnerability Allows Unauthorized Fund Transfers via Misuse of executeTxFromModule() in FundingManager**

  A significant vulnerability has been identified, which allows an attacker to transfer all tokens from the `FundingManager` to themselves. This issue stems from two main weaknesses:

  1. **Permission Oversight**: The `Orchestrator_v1` contract has an `executeTx()` function that allows the owner to make calls to any contract. Alarmingly, any module within the system can also assume this permission through `executeTxFromModule()`, effectively posing as the orchestrator.
  2. **Flawed Access Control**: The `FundingManager` contract uses a vulnerable access control called `onlyOrchestrator()`, which only checks if the `msg.sender` is the orchestrator instead of verifying the owner's role, as `onlyOrchestratorOwner()` does.

  This flaw enables any module to call `transferOrchestratorToken()` indirectly through `executeTxFromModule()`, bypassing proper access controls. It is recommended to use `onlyOrchestratorOwner()` for critical functions like `transferOrchestratorToken()` and implement stricter controls on `executeTxFromModule()`.

Link: Issue #50

- **User Blacklisted by USDC Can't Unstake Tokens in LM_PC_Staking_v1 Contract**

  The `LM_PC_Staking_v1.sol` contract allows users to stake tokens such as USDC to earn rewards. However, problems arise if a user is blacklisted by USDC, as illustrated by a scenario involving Alice. After staking USDC, Alice realizes she is blacklisted when she attempts to unstake. Since USDC's transfer function checks for blacklisted addresses, the transaction fails, leaving Alice unable to retrieve her staked tokens.

  To resolve this, it is recommended to modify the `unstake` function to accept a recipient address parameter, allowing the staked tokens to be transferred to a non-blacklisted address. This ensures users can retrieve their assets even under blacklist conditions.

  Link: Issue #54

- **Inconsistent Shutdown Mechanism in Different Proxy Implementations Poses Potential Security Risk**

  The protocol uses a beacon proxy pattern for efficient module deployment, either through `InverterTransparentUpgradeableProxy_v1` for flexible updates or `InverterBeaconProxy_v1` for a fixed update path. Each proxy reads the implementation contract differently: the former sets the implementation during construction and cannot adapt to shut downs initiated by governance, causing issues if a faulty version is deployed. This problem persists as `InverterTransparentUpgradeableProxy_v1` can't properly "shutdown" by reverting to a zero address due to code validity checks. To address this, proposing to implement a mapping in the beacon to track shutdowns and adjusting the proxy to check this mapping dynamically can mitigate the issue.

  Link: Issue #55

- **Lack of Assertion ID Validation Allows Unauthorized Callback to Reset Pending State**

  The function `assertionResolvedCallback` within a contract is essential for integrating with OOv3. However, the function doesn't verify if `assertionId` actually exists, allowing malicious users to exploit this. Specifically, malicious users can direct the address of `LM_PC_KPIRewarder_v1` as the `callbackRecipient` for an assertion not created by `LM_PC_KPIRewarder_v1`.

  In an attack scenario, a user can create and dispute an assertion through OOv3, calling `settleAssertion` to make `assertionResolvedCallback` run on `LM_PC_KPIRewarder_v1`. Since the `assertionId` isn't validated, an attacker can reset the `assertionPending` flag to false, allowing for new assertions to be processed, contrary to the intended single active assertion limitation. The recommendation is to check the existence of `assertionId` before resolving the assertion to prevent this exploit.

  Link: Issue #65

- **Staking Contract Incorrectly Uses Stakers' Funds for Incentives Leading to Potential Exploits**

The contract [LM_PC_Staking_v1] inherits from `ERC20PaymentClientBase_v1` and is treated as a regular `paymentClient` module, which isn't appropriate for staking. In staking, end users provide funds to incentivize other stakers. When the contract runs out of funds for an operation, it attempts to draw from the `FundingManager`. However, if the `FundingManager` lacks funds, the system can be exploited using a "pull-rug" and "pyramid" scheme. A malicious actor could exploit this by creating a staking contract with attractive terms, luring users in, and causing the last stakers to lose their funds. It's recommended to ensure the staking contract always has sufficient separate funds and not mix staked funds with reward distributions.

Link: Issue #70

- **DoS Vulnerability in LM_PC_KPIRewarder_v1 Due to Incorrect Assertion Handling**

  In the LM_PC_KPIRewarder_v1 contract, an asserter submits KPI data to the UMA oracle, which accepts bonds in various currencies like USDC/USDT. If an asserter submits incorrect data, an exploiter can dispute this using a blocklisted address, causing Denial-of-Service (DoS) on the UMA's settleAssertion function. This blocks the assertionResolvedCallback, disrupting the entire LM_PC_KPIRewarder_v1 logic. The problem arises because assertions can't be processed if assertionPending remains true. A potential solution involves a backup mechanism allowing new assertions to indicate if a contract is stuck, which can reset assertionPending to false.

  Attachments include a Proof of Concept and an optional revised code file suggesting the backup logic.

  Link: Issue #75

- **Admin Bypass of Orchestrator Module Checks for Critical Components**

  An admin can bypass critical checks when adding new modules to the orchestrator, specifically the `IFundingManager_v1`, `IAuthorizer_v1`, or `IPaymentProcessor_v1` modules. Normally, these modules must pass `_enforcePrivilegedModuleInterfaceCheck` and another check. The scenario involves the admin initially passing these checks with `initiateSetFundingManagerWithTimelock`, then canceling this process with `cancelFundingManagerUpdate`. The admin can then use `initiateAddModuleWithTimelock` to introduce a new, potentially malicious, module and finalize this with `executeSetFundingManager`. This sequence allows the admin to gain undue control over the system. Mitigation involves ensuring `cancelFundingManagerUpdate` also cancels the module removal process to prevent this bypass.

  Link: Issue #77

- **Admin Can Bypass Crucial Security Checks and Timelock Mechanisms**

  An admin can bypass crucial security checks and timelocks in the system. This happens because certain functions, namely `executeSetAuthorizer`, `executeSetFundingManager`, and `executeSetPaymentProcessor`, do not verify that the provided address matches the one specified during the initiation phase (`initiateSetAuthorizerWithTimelock`, `initiateSetFundingManagerWithTimelock`, and `initiateSetPaymentProcessorWithTimelock`). This loophole allows the admin to execute changes with a different address than originally specified, enabling unauthorized control and compromising the system's integrity. The recommended mitigation is to ensure that these `execute` functions always validate the address against what was initially set, thereby maintaining the timelock enforcement and overall system security.

Link: Issue #78

- **Vulnerability in Bounty Payout Role Check Allows Unauthorized Contributor Additions**

  In the smart contract [LM_PC_Bounties_v1], there are defined roles including BOUNTY_ISSUER_ROLE, CLAIMANT_ROLE, and VERIFIER_ROLE to ensure transparent and decentralized distribution of bounties to contributors. The issue arises because the length of the `contributors` array provided by the verifier is not checked against the `_claimRegistry[claimId].contributors`. This gap allows a claimant to update the contributors' list to include their address with a significant reward just before the verifier executes the `verifyClaim` function. Consequently, more payments than intended can be transferred. This exploit can result in unauthorized funds being claimed, undermining the system's transparency and security. A potential fix involves comparing the length of the provided `contributors` array with the stored list in `_claimRegistry`.

  Link: Issue #82

- **Unlimited activePaymentReceivers array can cause DoS in staking contracts**

  `PP_Streaming_v1`, a payment processor, pays out funds over a period rather than all at once. This requires users to claim their streams, tracked through the `activePaymentReceivers` mapping. When `_addPayment` is called, it adds `paymentReceivers` if they are not already present, leading to the potential for an unbounded array size. A malicious user can exploit this by staking minuscule amounts with many addresses, causing the array to become excessively large and gas costs for claiming rewards to exceed limits, effectively bricking the `LM_PC_Staking_v1` contract. Fixing this would involve updating the payment processor, but real users with pending rewards might face reverts during claims. Alternative solutions include avoiding large arrays for pending users or adding functions to remove specific streams.

  Link: Issue #85

- **Raw Token Contract Calls May Result in Incorrect Transfer Confirmation**

  PP_Simple_v1.sol and **PP_Streaming_v1.sol** have raw calls to a token contract on lines 126 and 726, respectively. These calls might mistakenly be treated as successful transfers when the target `token_` address is not actually a contract. This happens because calls to non-contract addresses return `true`. Consequently, the following conditional check will pass, causing the `TokensReleased` event to be emitted incorrectly and notifying the client about paid tokens when none were transferred. This raises issues like potential phishing attacks and hidden issues in other parts of the project's modular code. As a fix, it is recommended to use the **SafeERC20** library from OpenZeppelin. Additionally, checks on payment order creation should be implemented to ensure correct token addresses.

  Link: Issue #118

- **Prevent JIT Liquidity Exploits in FM_Rebasing_v1 by Implementing Withdrawal Windows**

FM_Rebasing_v1 is susceptible to just-in-time (JIT) liquidity attacks. Users can front-run transactions that increase the bit value by depositing large quantities of assets, and subsequently back-run these transactions by quickly withdrawing their assets, enabling them to extract a portion of the rewards with minimal risk. An example demonstrates how a user, Bob, could see a transaction, deposit to own 50% of the pool, and then withdraw to claim 50% of the resulting tokens, effectively capturing rewards meant for active depositors. To mitigate this, it is recommended to implement a withdrawal window to deter such manipulative actions.

Link: Issue #128

- **Bancor Virtual Supply Vulnerable to MEV Attack Due to Adjustable Reserve Ratio**

  The function `FM_BC_Bancor_Redeeming_VirtualSupply_v1.setReserveRatioForSelling` is vulnerable to a MEV (Miner Extractable Value) attack. If an attacker notices a reduction in the `reserveRatioForSelling`, they could exploit this by front-running the transaction. By using a flash loan, the attacker could initiate a large buy transaction before the parameter changes and immediately sell after the change, thereby realizing significant profit in one transaction batch. The provided proof of concept illustrates this scenario, showcasing how an attacker can front-run and back-run to exploit the system. This vulnerability can lead to misuse of the bonding curve mechanism, potentially impacting all genuine users of the protocol. To mitigate such risks, dynamic adjustments of the reserve ratio with buy or sell transactions might be necessary.

  Link: Issue #131

- **Potential Loss of Collateral Funds Due to Virtual Supply Manipulation in FundingManager**

  A critical vulnerability in the `BondingCurve FundingManager` allows for potential total loss of collateral funds. This issue arises because the balance of the `FundingManager` can be altered independently from its virtual `collateral` and `issuance` supply. By exploiting the `setVirtualCollateralSupply()` and `setVirtualIssuanceSupply()` functions, an attacker can perform risk-free sandwich attacks. For instance, an attacker can front-run a transaction to raise the collateral supply with a minimal buy order and then back-run by selling just bought issuance tokens, nearly emptying the `FundingManager`. Although the severity and frequency of such attack scenarios vary, the vulnerability remains a significant risk. The coded proof of concept demonstrates this exploit clearly. Recommendations for mitigating this problem will be added in the comments.

  Link: Issue #155

Low severity issues

- **Missing `_disableInitializers()` in Constructor Risk in Two Contract Files**

  `OrchestratorFactory_v1.sol` and `ModuleFactory.sol` lack a `_disableInitializers()` call in their constructors. Without this, implementation contracts can be exploited by attackers using the `init` function, potentially causing unexpected behavior. It is recommended to invoke `_disableInitializers` to lock the contract upon deployment, preventing such vulnerabilities.

  Link: Issue #8

- **Failed ERC20 Transfers Not Properly Handled in RedeemingBondingCurveBase_v1.sol**

Failed token transfers are not properly handled. The current implementation uses `transfer`, which doesn't account for tokens that return false instead of reverting. For compliance with the ERC20 specification, the return value of the transfer must be checked, suggesting the use of `safeTransfer` instead.

Link: Issue #10

- **Ensure PaymentOrder Struct Validates start and end Timestamps Correctly**

  In the `PaymentClientBase` contract, the `validPaymentOrder` modifier checks only `recipient`, `token`, and `amount` fields in the `PaymentOrder` struct. However, it overlooks the `start` and `end` fields. It is recommended to include a `_ensureValidRange` function to ensure that `end` is greater than or equal to `start`.

  Link: Issue #16

- **Replace `ecrecover` with OpenZeppelin's `ECDSA.recover` to prevent signature malleability**

  Using EVM's `ecrecover` is susceptible to signature malleability. It's recommended to use the `ECDSA.recover` method from the OpenZeppelin library, which ensures that the `v` and `s` values of the signatures are validated, thereby mitigating potential risks associated with malleable signatures.

  Link: Issue #17

- **Fee Bypass Vulnerability in BondingCurveFundingManagerBase Allows Exploiting Fee Calculation**

  A vulnerability exists in the `BondingCurveFundingManagerBase`, where setting a 2% fee can be bypassed if the `buy` function is repeatedly called with an amount of `49`, causing the fee calculation to round to zero. This exploit allows users to buy tokens without paying fees, especially in low-activity or low-cost environments like L2 chains. A suggested fix is to implement a `minBuyAmount` state variable.

  Link: Issue #22

- **Add Event Emission to castVote Function for Better Transparency**

  The function `AUT_EXT_VotingRoles_v1.sol::castVote` lacks event emission, which is essential for transparency and informing users about important changes. This omission can hinder users' ability to track contract changes. Adding event emissions will enhance transparency by providing a clear record of vote casting.

  Link: Issue #27

- **Time Validation Issue in PP_Streaming_v1 Contract Due to Incorrect Operator**

  The time validation check in the `validTimes` function is incorrectly implemented, using the `&&` operator instead of `||`. This error causes invalid times to be treated as valid, potentially putting the protocol in an unexpected state. The suggested fix is to update the code to use `||` in the return statement.

  Link: Issue #53

- **Delay in Module Replacement Causes DoS in Timelock Functions**

  The `initiateSetFundingManagerWithTimelock`, `initiateSetAuthorizerWithTimelock`, and `initiateSetPaymentProcessorWithTimelock` functions add a new module and remove the old one without changing the total number of modules. However, if the maximum module limit is reached, these functions will revert, requiring an additional timelock period to remove a module first. This can cause a Denial of Service (DoS). To address this, the `moduleLimitNotExceeded` check should be bypassed for these functions.

  Link: Issue #56

- **Function Fails to Check Pending Modules, Risking Module Limit Bypass**

  The `moduleLimitNotExceeded` function currently verifies only the number of existing modules, ignoring those pending addition. This allows the owner to repeatedly call `initiateAddModuleWithTimelock`, potentially exceeding the module limit. The solution is to update the check to include both existing and pending modules, ensuring the module limit is not surpassed.

  Link: Issue #57

- **Duplicate Module Titles Cause Incorrect Address Returns in Orchestrator**

  When adding a new module to the orchestrator, the system fails to check for existing modules with the same title, causing potential duplicates. This results in incorrect module addresses returned by the `findModuleAddressInOrchestrator` function, leading to possible execution of unintended modules and security vulnerabilities. Implementing a check for unique module titles is recommended to mitigate this issue.

  Link: Issue #58

- **Rounding errors in Funding Manager's price calculations for issuance and collateral tokens**

  The `getStaticPriceForBuying()` and `getStaticPriceForSelling()` functions, which calculate token prices using Aragon's BatchedBancorMarketMaker formula, are prone to rounding errors. This arises from discrepancies in decimal places between collateral and issuance tokens. Normalizing supplies or adjusting the PPM variable are suggested solutions to prevent rounding the price to zero.

  Link: Issue #59

- **Restrict Governor_v1.acceptOwnership() function access to COMMUNITY_MULTISIG_ROLE only**

  `Governor_v1.acceptOwnership()` function currently allows both `COMMUNITY_MULTISIG_ROLE` and `TEAM_MULTISIG_ROLE` to access it, contrary to the Natspec documentation that states only `COMMUNITY_MULTISIG_ROLE` should have access. This discrepancy could lead to unauthorized access, breaking the protocol's intended design. It is recommended to restrict the function to `COMMUNITY_MULTISIG_ROLE` only.

  Link: Issue #60

- **Incorrect Function Selector Used in BondingCurveBase_v1 Causing Fee Calculation Errors**

  In `BondingCurveBase_v1`, the function `_getFunctionFeesAndTreasuryAddresses` is called with an incorrect function selector for `_buyOrder`, leading to mismatched fee calculations. The correct selector should be `0xd88e833f`, but currently, it's `0xebc8b020`. This discrepancy causes `getCollateralWorkflowFeeAndTreasury` to return the default collateral fee instead of the intended value, potentially affecting order execution and fee calculations. Recomputing the function selector correctly can resolve this issue.

  Link: Issue #61

- **postAssertion Function Assumptions About Asserter Bond Payment are Incorrect**

  The `postAssertion` function assumes that the `asserter` address pays for the bond, but instead, the `_msgSender()` always pays it. This discrepancy makes it impossible for the module to pay for the bond, breaking protocol design. A proposed solution is to revise `assertDataFor` so that the bond transfers from the `asserter`, or add checks for sufficient balance when the module itself is the `asserter`.

  Link: Issue #64

- **Voting Role Manager Vulnerability Allows Malicious Takeover and Unauthorized Execution**

  The `AUT_EXT_VotingRoles_v1` module lacks a minimum threshold enforcement, allowing any voter to remove others if the threshold is set to zero or one. This could lead to a complete takeover of the voting role manager contract. It is recommended to enforce a minimum threshold of two voters to prevent malicious takeovers.

  Link: Issue #67

- **ERC165 Implementation Issue in OptimisticOracleIntegrator Missing Interface Checks**

  The protocol's implementation of ERC165 to check interface support in its `Module_v1` may lead to significant issues depending on external integrations. Specifically, `IOptimisticOracleIntegrator` should include the `OptimisticOracleV3CallbackRecipientInterface` to ensure mandatory functions are recognized when an assertion in `OO_V3` is made. The recommendation is to override `supportsInterface` accordingly.

  Link: Issue #74

- **Ensure Storage Gaps in Upgradeable Contracts to Prevent Variable Overwrites**

  For upgradeable contracts, a storage gap is necessary to add new state variables without affecting storage compatibility. The Inverter contracts lack this, risking failure if the base contract has new variables. Identified contracts include `Orchestrator_v1.sol`, `FM_Rebasing_v1.sol` and others. Proposed fixes involve adding storage gaps and using `ERC165Upgradeable`.

  Link: Issue #84

- **initiateAddModuleWithTimelock Doesn't Ensure Module is Not a Privileged Module**

The function `initiateAddModuleWithTimelock()` is designed to add new logic modules to the Orchestrator but fails to ensure that only logic modules are added and not privileged modules. This oversight allows privileged modules to be added without removing existing ones, contradicting the intended behavior. It is recommended to check if the module supports the `IModule_v1` interface and to revert if it also supports any privileged interfaces.

Link: Issue #86

- **Indexed Dynamic Arrays in Event Emit Retrieve Hash Instead of Data**

  The `LM_PC_Bounties_v1` contract has an event `ClaimAdded` where the dynamic array `Contributor[]` is indexed. This indexing returns a keccak256 hash, resulting in meaningless 32-byte values. This could disrupt DApp operations and lead to data loss. Suggested fix: remove indexing from dynamic arrays in events.

  Link: Issue #91

- **Tokens Intended as Workflow Fees are Incorrectly Transferred by Orchestrator**

  All funding managers have the `transferOrchestratorToken` function that allows an orchestrator to pull funds without considering accumulated fees. This can lead to transferring tokens meant for fees, causing a substantial loss for the protocol. This issue affects the withdrawal logic and may lead to denial of service (DoS) when user funds are expected to be available.

  Link: Issue #101

- **ElasticReceiptTokenBase_v1 Incompatible with ERC2771 due to msg.sender Usage**

  The protocol uses ERC2771 for meta transactions, allowing users to sign a transaction and have someone else pay for it. However, `ElasticReceiptTokenBase_v1` is incompatible with ERC2771 due to its use of `msg.sender` instead of `_msgSender`. A recommended fix involves changing the inheritance structure to enable `_msgSender` usage.

  Link: Issue #104

- **New major module versions should delete previous ones to avoid compatibility issues**

  When new modules with major version updates are added, previous versions are not removed, leading to the possibility of new users using outdated versions. This can cause compatibility issues and security vulnerabilities. It is recommended that old versions be deleted when new major versions are introduced.

  Link: Issue #108

- **Remove Redundant `_earned` Call in `_distributeRewards` for Gas Optimization**

  The function `_distributeRewards` in the LM_PC_Staking_v1 contract calls `_earned`, which is unnecessary since `_update`, executed before `_distributeRewards`, already includes `_earned`. Replacing `uint amount = _earned(recipient, rewardValue)` with `uint amount = rewards[recipient]` saves gas by avoiding redundant calculations.

  Link: Issue #112

- **Vote Misattribution Due to Blockchain Re-orgs in Motion Submission**

  Motions are created via `AUT_EXT_VotingRoles_v1.createMotion`, and voters cast their votes using `AUT_EXT_VotingRoles_v1.castVote`. In case of blockchain reorganization on Polygon, an attacker could potentially misdirect votes intended for one motion to another due to reordering of transactions. Calculating `motionId` as a hash of `target`, `action`, and `motionCount` is suggested to mitigate this issue.

  Link: Issue #117

- **Reentrancy Issue in PP_Streaming_v1 Contract During Token Claiming Process**

  The `claimPreviouslyUnclaimable` function in the `PP_Streaming_v1` contract has a reentrancy vulnerability. When users call this function, they can reenter the contract before `_outstandingTokenAmounts` is updated, potentially resulting in temporary fund locks. To fix this, state changes should occur before external calls.

  Link: Issue #119

- **Rewards Rounded Down to Zero Causes Denial of Service in Assertion Callback**

  In the `assertionResolvedCallback` function of `LM_PC_KPIRewarder_v1.sol`, a potential issue occurs when calculating rewards. The expression `rewardAmount += achievedReward * (trancheRewardValue / trancheEnd);` rounds down the value to zero if `trancheRewardValue` is less than `trancheEnd`, resulting in lost rewards. Additionally, this creates a denial of service scenario as new assertions cannot be posted because `assertionPending` remains true. Removing the curly braces can prevent this precision loss.

  Link: Issue #120

- **Overflow vulnerability in staking contracts allows attacker to brick contract with specific token**

  The `stake` functions in both `LM_PC_Staking_v1` and `LM_PC_KPIRewarder_v1` contracts have a flaw when handling tokens like cUSDCv3. If an amount of `type(uint256).max` is staked, it can lead to an overflow, potentially bricking the contracts and trapping any reward tokens sent. This issue affects deployments on Polygon and Linea as well.

  Link: Issue #126

- **Incorrect Documentation on Maximum Deposit Amount Allowable Before Transaction Revert**

  The developer's comment incorrectly stated the maximum `_depositAmount` allowed before a transaction reverts using the Bancor Formula. While the comment mentions a limit of `10^20`, tests showed transactions with values up to `100_000_000_000_000_000_000e18` did not revert.

  Link: Issue #132

- **Nonce Manipulation and Front-Running Vulnerability in TransactionForwarder_v1**

  The protocol's custom multicall implementation, which supports EIP2771, has a vulnerability that allows a malicious party to exploit the nonce system. An attacker can manipulate on-chain states to invalidate legitimate transactions using a non-reusable nonce, leading to potential transaction failures and financial loss.

Link: Issue #133

- **Modify stake function to allow smaller amounts for existing stakingQueue users**

  The `stake` function's `minimumStake` check restricts existing users in the `stakingQueue` from adding new amounts below the `minimumStake`. This can be resolved by modifying the `stake` function to apply the `minimumStake` check only to users not already in the `stakingQueue`, allowing smaller additional stakes.

  Link: Issue #136

- **Missing Deadline Checks in Contract Functions May Lead to Loss of Funds**

  Certain functions in a contract lack deadline checks, creating potential issues. Users may experience loss of funds due to outdated slippage parameters if transactions are delayed in the mempool. This gap can result in MEV exploitation. Introducing a deadline parameter to these functions would address these problems by ensuring timely transaction execution.

  Link: Issue #137

- **Function Initialization Can Cause Reversion in Bancor Redeeming VirtualSupply When Token Decimals Mismatch**

  The `FM_BC_Bancor_Redeeming_VirtualSupply_v1.init()` function can be initialized with certain token decimals and issuance supply combinations that make calling the `buy` function impossible without it reverting. Specifically, if an issuance token with 19 decimals has a `virtualIssuanceSupply` set to 1, any `buy` call will fail due to rounding issues in the `calculatePurchaseReturn` function. The recommendation is to add a check in `init()` ensuring a minimum `virtualIssuanceSupply` compatible with token decimals.

  Link: Issue #139

- **Inaccurate Event Emissions in OptimisticOracleIntegrator for Resolved Assertions**

  The `OptimisticOracleIntegrator::assertionResolvedCallback(...)` function emits the `DataAssertionResolved(...)` event regardless of whether an assertion is resolved truthfully, contrary to UMA documentation. This discrepancy can lead to misinterpretation of transaction outcomes, affecting audits, monitoring, and trust in the system's reporting accuracy. The function should be modified to only emit the event for truthful assertions.

  Link: Issue #148

- **Users Can Lose Tokens Due to Rounding Errors in Bancor Formula**

  Users can lose expected value when the `BancorFormula` returns zero, which can occur when the `issuance` or `collateral` token supply drastically outweighs the other. This issue results in rounding problems, leading to zero token returns for users during `buy` or `sell` orders. To prevent this, it is recommended to ensure that mint and redeem amounts are non-zero in the relevant functions.

Link: Issue #157

- **Unspecific Compiler Versions in Several Solidity Files May Pose Security Risks**

  Most Solidity files use a fixed `0.8.23` version, but some non-interface files have unspecified compiler versions, including LinkedIdList.sol, AUT_TokenGated_Roles_v1.sol, LibMetadata.sol, AncillaryData.sol, and ClaimData.sol. This can be a security risk if a vulnerable or outdated compiler version is inadvertently used. The recommendation is to pin these files to `0.8.23`.

  Link: Issue #158

Conclusion

The Hats.finance audit competition for the Inverter Network highlighted several security issues in the protocol's contracts, ranging from medium to low severity. Among the critical medium severity issues were vulnerabilities that could lead to unauthorized fund transfers, potential reorg risks on Polygon, and improper staking mechanisms potentially exploiting new depositors. The assessment identified 55 auditors, from 160 submissions, earning a total payout of $54,164.51.

Key insights suggest improvements such as using `create2` instead of `create` for module deployment to ensure better security against front-running attacks, managing rebalances more efficiently to prevent depositor losses, and strengthening access controls in the FundingManager logic. Additionally, improvements are recommended to the staking processes, governance roles, and the handling of assertions and reward calculations.

Overall, while the audit emphasized proactive mitigation strategies, implementing the suggested fixes and stronger access controls will be crucial. Most importantly, reviewing and adapting the proposed solutions will aid in securing the Inverter Network's modular architecture and safeguard against future exploits in the DeFi ecosystem.

Disclaimer

This report does not assert that the audited contracts are completely secure. Continuous review and comprehensive testing are advised before deploying critical smart contracts.

The Inverter Network audit competition illustrates the collaborative effort in identifying and rectifying potential vulnerabilities, enhancing the overall security and functionality of the platform.

Hats.finance does not provide any guarantee or warranty regarding the security of this project. Smart contract software should be used at the sole risk and responsibility of users.