PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For Hydrex Incentives Campaign

04 Dec 2025

paladinsec.co

info@paladinsec.co

Table of Contents

Table of Contents 2

Disclaimer 3

1 Overview 4

1.1 Summary 4

1.2 Contracts Assessed 5

1.3 Findings Summary 6

1.3.1 Global 7

1.3.2 GaugeIncentiveCampaign 7

1.3.3 IncentiveCampaignManager 7

1.3.4 GaugeFactoryIncentiveCampaign 7

1.3.5 GaugeFactoryIncentiveCampaignProxy 8

1.3.6 VoterV5_GaugeLogic 8

2 Findings 9

2.1 Global 9

2.1.2 Issues & Recommendations 9

2.2 GaugeIncentiveCampaign 11

2.2.1 Privileged Functions 11

2.2.2 Issues & Recommendations 12

2.3 IncentiveCampaignManager 16

2.3.1 Privileged Functions 16

2.3.2 Issues & Recommendations 17

2.4 GaugeFactoryIncentiveCampaign 18

2.4.1 Privileged Functions 18

2.4.2 Issues & Recommendations 18

2.5 GaugeFactoryIncentiveCampaignProxy 19

2.5.1 Privileged Functions 19

2.5.2 Issues & Recommendations 19

2.6 VoterV5_GaugeLogic 20

2.6.1 Privileged Functions 20

2.6.2 Issues & Recommendations 20

Disclaimer

Paladin Blockchain Security ("Paladin") has conducted an independent audit to verify the integrity of and highlight any vulnerabilities or errors, intentional or unintentional, that may be present in the codes that were provided for the scope of this audit. This audit report does not constitute agreement, acceptance or advocation for the Project that was audited, and users relying on this audit report should not consider this as having any merit for financial advice in any shape, form or nature. The contracts audited do not account for any economic developments that may be pursued by the Project in question, and that the veracity of the findings thus presented in this report relate solely to the proficiency, competence, aptitude and discretion of our independent auditors, who make no guarantees nor assurance that the contracts are completely free of exploits, bugs, vulnerabilities or deprecation of technologies. Further, this audit report shall not be disclosed nor transmitted to any persons or parties on any objective, goal or justification without due written assent, acquiescence or approval by Paladin.

All information provided in this report does not constitute financial or investment advice, nor should it be used to signal that any persons reading this report should invest their funds without sufficient individual due diligence regardless of the findings presented in this report. Information is provided 'as is', and Paladin is under no covenant to the completeness, accuracy or solidity of the contracts audited. In no event will Paladin or its partners, employees, agents or parties related to the provision of this audit report be liable to any parties for, or lack thereof, decisions and/or actions with regards to the information provided in this audit report.

Cryptocurrencies and any technologies by extension directly or indirectly related to cryptocurrencies are highly volatile and speculative by nature. All reasonable due diligence and safeguards may yet be insufficient, and users should exercise considerable caution when participating in any shape or form in this nascent industry.

The audit report has made all reasonable attempts to provide clear and articulate recommendations to the Project team with respect to the rectification, amendment and/or revision of any highlighted issues, vulnerabilities or exploits within the contracts provided. It is the sole responsibility of the Project team to sufficiently test and perform checks, ensuring that the contracts are functioning as intended, specifically that the functions therein contained within said contracts have the desired intended effects, functionalities and outcomes of the Project team. Paladin retains the right to re-use any and all knowledge and expertise gained during the audit process, including, but not limited to, vulnerabilities, bugs, or new attack vectors. Paladin is therefore allowed and expected to use this knowledge in subsequent audits and to inform any third party, who may or may not be our past or current clients, whose projects have similar vulnerabilities. Paladin is furthermore allowed to claim bug bounties from third-parties while doing so.

1 Overview

This report has been prepared for Hydrex Incentives Campaign on the Ethereum network. Paladin provides a user-centred examination of the smart contracts to look for vulnerabilities, logic errors or other issues from both an internal and external perspective.

1.1 Summary

Project Name	Hydrex Incentives Campaign
URL	https://www.hydrex.fi/
Platform	Ethereum
Language	Solidity
Preliminary	https://github.com/hydrexfi/hydrex-contracts/tree/35ddfaac781e856d14edb4196fca22d87448f5d0
Resolution	https://github.com/hydrexfi/hydrex-contracts/tree/7bc02d6149c4f154ca6ee833678deef28bad9d26

1.2 Contracts Assessed

Name	Contract	Live Code Match
GaugeIncentiveCampaign		PENDING
IncentiveCampaignManager		PENDING
GaugeFactoryIncentiveCampaign		PENDING
GaugeFactoryIncentiveCampaignProxy		PENDING
VoterV5_GaugeLogic		PENDING

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
Governance	1	-	-	1
High	1	1	-	-
Medium	2	2	-	-
Low	2	2	-	-
Informational	3	2	-	1
Total	**9**	**7**	**-**	**2**

Classification of Issues

Severity	Description
Governance	Issues under this category are where the governance or owners of the protocol have certain privileges that users need to be aware of, some of which can result in the loss of user funds if the governance's private keys are lost or if they turn malicious, for example.
High	Exploits, vulnerabilities or errors that will certainly or probabilistically lead towards loss of funds, control, or impairment of the contract and its functions. Issues under this classification are recommended to be fixed with utmost urgency.
Medium	Bugs or issues that may be subject to exploit, though their impact is somewhat limited. Issues under this classification are recommended to be fixed as soon as possible.
Low	Effects are minimal in isolation and do not pose a significant danger to the project or its users. Issues under this classification are recommended to be fixed nonetheless.
Informational	Consistency, syntax or style best practices. Generally pose a negligible level of risk, if any.

1.3.1 Global

ID	Severity	Summary	Status
1	GOV	Centralized upgrade and asset control across upgradeable gauge ecosystem	ACKNOWLEDGED
2	INFO	Missing basic validation and no-op checks in setters and factory creation	✓ RESOLVED

1.3.2 GaugeIncentiveCampaign

ID	Severity	Summary	Status
3	HIGH	Nested initializer: child initialize() calls parent initialize() which also has the initializer modifier	✓ RESOLVED
4	MEDIUM	Parent gauge logic not disabled: required GaugeV2 functions not overridden in non-staking GaugeIncentiveCampaign	✓ RESOLVED
5	MEDIUM	Unclear handling of reward token received as pool fees	✓ RESOLVED
6	LOW	Missing return value for Merkl campaign creation prevents reliable event emission	✓ RESOLVED
7	INFO	Unused modifier check for distribution address	✓ RESOLVED
8	INFO	Claim fees callable during emergency mode	ACKNOWLEDGED

1.3.3 IncentiveCampaignManager

ID	Severity	Summary	Status
9	LOW	Conflated hasOverride assumes both type and Merkl config are set	✓ RESOLVED

1.3.4 GaugeFactoryIncentiveCampaign

No issues found.

1.3.5 GaugeFactoryIncentiveCampaignProxy

No issues found.

1.3.6 VoterV5_GaugeLogic

No issues found.

2 Findings

2.1 Global

The issues listed in this section apply to the protocol as a whole. Please read through them carefully and take care to apply the fixes across the relevant contracts.

2.1.2 Issues & Recommendations

Issue #1	Centralized upgrade and asset control across upgradeable gauge ecosystem
Severity	GOVERNANCE
Description	A single owner (without enforced timelock or role separation) can upgrade implementations, reconfigure reward routing, toggle emergency states, and sweep tokens. This concentration of control enables unannounced logic changes or fund redirection if keys are compromised or misused.
Recommendation	Document trust model by clearly stating that Owner and other managing roles are trusted and any single compromised role can disrupt protocol operation, and consider timelocks for critical operations. Implement governance hardening by using a multi-signature wallet with sufficient signers and thresholds (or document it if it already is).
Resolution	ACKNOWLEDGED

Issue #2	Missing basic validation and no-op checks in setters and factory creation
Severity	INFORMATIONAL
Description	• `IncentiveCampaignManager`: `setDefaultDistributorType` and `setDefaultMerklConfig` accept values identical to current ones, emitting events without state change. Overrides (`setDistributorType`, `setMerklConfig`) also allow no-op updates. This creates noisy logs and weakens operational guarantees. • `GaugeFactoryIncentiveCampaign`: `createGaugeV2` lacks basic parameter validation (e.g., `_token` and `_distribution` non-zero and contracts), and admin setters (`setPermissionsRegistry`, `setIncentiveCampaignManagerDefault`) do not check for same-as-current values, allowing no-op updates.
Recommendation	Add same-value guards in all admin setters to skip no-op updates and event spam.
Resolution	RESOLVED

2.2 GaugeIncentiveCampaign

`GaugeIncentiveCampaign` is a non□ staking gauge that acts as a pass□ through for governance emissions and a fee router for Algebra pools. It does not track user balances or accept deposits — all staking and harvest entry points should revert.

On each epoch, the `Voter` (distribution) calls `notifyRewardAmount` with a whitelisted reward token and then the gauge pulls the net amount, reads the effective distributor type and Merkl configuration from `IncentiveCampaignManager`, and immediately forwards the tokens to the configured Merkl distribution creator.

2.2.1 Privileged Functions

- `setCampaignManager`
- `notifyRewardAmount`

2.2.2 Issues & Recommendations

Issue #3	**Nested initializer: child initialize() calls parent initialize() which also has the initializer modifier**
Severity	HIGH SEVERITY
Description	`GaugeIncentiveCampaign.initialize` invokes `super.initialize` while both functions are marked with `initializer`. With OpenZeppelin `Initializable`, nesting initializer-guarded functions is unsafe: it can revert during deployment (double-initialization guard) and block clean upgrade paths (no reinitializer versioning). This risks failed gauge deployments and brittle future upgrades.
Recommendation	Refactor to the standard OZ pattern: • In parent, extract logic into `__GaugeV2_init(...)` marked `internal onlyInitializing`. • Child should call `__GaugeV2_init(...)` instead of `super.initialize`. Alternatively, version with `initializer` (parent = v1) and `reinitializer(2)` (child), thus avoiding nested `initializer` calls. If needed, call `_disableInitializers()` to prevent accidental direct initialization on the implementation.
Resolution	RESOLVED

Issue #4	**Parent gauge logic not disabled: required GaugeV2 functions not overridden in non-staking GaugeIncentiveCampaign**
Severity	MEDIUM SEVERITY
Description	Contract claims "All staking/deposit flows are intentionally disabled; no user balances are tracked," but multiple `GaugeV2` functions that enable staking, reward claiming, accounting, and admin token management remain inherited and callable because they are not overridden. While deposits will be impossible since Algebra pool does not inherit ERC20, these functions can still be invoked with misleading errors or unintended side effects. Affected functions include: • User/state mutating: `depositAll`, `emergencyWithdraw`, `emergencyWithdrawAmount`, `withdrawAllAndHarvest` • Reward claim: `getReward()` (no params) • Accounting/views: `updateRewardToken`, `availableBalance`, `lastTimeRewardApplicable`, `rewardPerToken`, `rewardForDuration`, `periodFinish`, `left` • Admin token management: `addRewardToken`, `removeRewardToken`, `sweepTokens` These expose unintended functionality (e.g. enabling deposits/emergency withdrawal paths, reward token mutation) in a supposed "non-staking" gauge, creating inconsistent state assumptions, potential silent balance tracking, and future upgrade ambiguity.
Recommendation	Explicitly override all listed functions to revert with dedicated custom errors (e.g. `StakingDisabled()`, `RewardsDisabled()`, `AdminDisabled()`). Consider not inheriting `GaugeV2` if staking functionality is not needed, to avoid confusion, and simply implement the required interfaces.
Resolution	RESOLVED

Issue #5	Unclear handling of reward token received as pool fees
Severity	MEDIUM SEVERITY
Description	`_claimFees()` explicitly zeroes balances when `token0` or `token1` equals `rewardToken`, so any reward token accrued as Algebra pool fees is ignored and never forwarded to internal bribe nor Merkl. No alternative path exists to process these balances other than using `sweepTokens` functionality.
Recommendation	Define explicit policy: either include `rewardToken` in fee sweep or segregate and forward via `_forwardToMerkl` with a distinct event. Also consider documenting the chosen behavior clearly.
Resolution	RESOLVED

Issue #6	Missing return value for Merkl campaign creation prevents reliable event emission
Severity	LOW SEVERITY
Description	`IMerklDistributionCreator.createCampaign` does not return the `campaignId` it creates. `GaugeIncentiveCampaign._forwardToMerkl` assumes `params.campaignId` (input) equals the final campaign identifier and emits it, but cannot confirm creation or handle distributor-side mutation/versioning. Lack of a returned value blocks verification and future-proofing.
Recommendation	Modify interface to `function createCampaign(CampaignParameters memory params) external returns (bytes32 campaignId);` Capture returned value in `_forwardToMerkl` and emit that in `IncentiveDistributed`.
Resolution	RESOLVED

Issue #7	Unused modifier check for distribution address
Severity	INFORMATIONAL
Description	`notifyRewardAmount` uses `require(msg.sender == DISTRIBUTION, "Only distribution")` instead of the existing `onlyDistribution` modifier from `GaugeV2`, creating inconsistent revert reasons (string vs custom error `OnlyDistributor()`) and duplicate logic.
Recommendation	Replace inline `require` with `onlyDistribution` modifier for consistency.
Resolution	RESOLVED

Issue #8	Claim fees callable during emergency mode
Severity	INFORMATIONAL
Description	`_claimFees()` lacks an emergency guard. Users can trigger fee sweeping while `emergency == true`, contradicting expected full suspension of gauge operations.
Recommendation	Gate `_claimFees()` (and public wrapper) with `isNotEmergency` or an explicit `if (emergency) revert`.
Resolution	ACKNOWLEDGED

2.3 IncentiveCampaignManager

`IncentiveCampaignManager` is an Ownable registry that defines how each gauge should route its received reward tokens into external incentive campaigns. It maintains per□ token defaults and per gauge□ and□ token overrides for a `DistributorType` (e.g., MERKL) and a `MerklConfig` blob containing distribution configuration.

Gauges resolve their routing by calling `getEffectiveConfig(gauge, token)`, then, if the effective type is MERKL, they forward rewards to the returned distributionCreator using the provided parameters.

2.3.1 Privileged Functions

- `setDefaultDistributorType`
- `setDefaultMerklConfig`
- `setDistributorType`
- `setMerklConfig`
- `clearOverride`

2.3.2 Issues & Recommendations

Issue #9	Conflated hasOverride assumes both type and Merkl config are set
Severity	LOW SEVERITY
Description	`hasOverride[g][t]` is toggled by either `setDistributorType` or `setMerklConfig`, but getters (`getDistributorType`, `getMerklConfig`, `getEffectiveConfig`) treat it as if both fields are overridden. If only type is set (e.g., MERKL) without a Merkl configuration, getters return a zero configuration instead of defaulting. This conflation can lead to unexpected zero values and misconfigurations.
Recommendation	Consider requiring a Merkl configuration when updating the default distribution type (best is to set both of them in one transaction).
Resolution	RESOLVED

2.4 GaugeFactoryIncentiveCampaign

`GaugeFactoryIncentiveCampaign` is an upgradeable factory that deploys incentive campaign gauges as `BeaconProxy` instances pointing to a shared `GaugeIncentiveCampaign` implementation managed through an upgradeable beacon.

Upon initialization, it records a permissions registry, deploys a beacon via `BeaconFactoryAdmin`, and stores a default `IncentiveCampaignManager` used for all subsequently created gauges.

Gauge creation `createGaugeV2` builds the initialization `calldata` (campaign manager, Algebra pool, voter distribution) and instantiates a new `BeaconProxy`, tracking its address and emitting a `GaugeCreated` event.

2.4.1 Privileged Functions

- `setPermissionRegistry`
- `setIncentiveCampaignManagerDefault`
- `activateEmergencyMode`
- `stopEmergencyMode`
- `setDistribution`
- `setGaugeRewarder`
- `setCampaingManagerForGauges`
- `setCentralTokenPool`

2.4.2 Issues & Recommendations

No issues found.

2.5 GaugeFactoryIncentiveCampaignProxy

`GaugeFactoryIncentiveCampaignProxy` is a thin wrapper around `TransparentUpgradeableProxy` used to host the upgradeable `GaugeFactoryIncentiveCampaign` logic. It stores only the proxy admin (which controls upgrades) and delegates all non□ admin calls to the factory implementation.

2.5.1 Privileged Functions

None.

2.5.2 Issues & Recommendations

No issues found.

2.6 VoterV5_GaugeLogic

`VoterV5_GaugeLogic` is the delegatecalled module that encapsulates all gauge-creation logic for `VoterV5`, keeping the main voter contract small. It maps a requested `GaugeType` to a configured DEX factory and gauge factory, validates the provided pool according to its type (classic pairs, Algebra/Gamma ALMs, Uniswap V3/V4 strategies, Morpho single-asset vaults, or Algebra pools for incentive campaigns), and enforces whitelist checks and "`isPair`" constraints for non-governance callers via the `PermissionsRegistry`.

For each new gauge, it derives tokenA/tokenB from the pool, constructs internal and external bribe contracts, selects the appropriate `oToken` or falling back to base, and deploys the gauge through the type-specific factory.

2.6.1 Privileged Functions

- None

2.6.2 Issues & Recommendations

No issues found.