GA GUARDIAN

Gamma

Position Managers

Security Assessment

July 22nd, 2025

Summary

Audit Firm Guardian

Prepared By Parker Stevens, Nicholas Chew, Zdravko Hristov, 0xCiphky, Mark Jonathas

Client Firm Gamma

Final Report Date July 22, 2025

Audit Summary

Gamma engaged Guardian to review the security of their position manager contracts to manage liquidity in Uniswap and Algebra. From the 9th of June to the 25th of June, a team of 5 auditors reviewed the source code in scope. All findings have been recorded in the following report.

Confidence Ranking

Given the lack of critical issues detected and minimal code changes following the main review, Guardian assigns a Confidence Ranking of 4 to the protocol. Guardian advises the protocol to consider periodic review with future changes. For detailed understanding of the Guardian Confidence Ranking, please see the rubric on the following page.

🔗 Blockchain network: **BNB Chain, Base**

✅ Verify the authenticity of this report on Guardian's GitHub: https://github.com/guardianaudits

Guardian Confidence Ranking

Confidence Ranking	Definition and Recommendation	Risk Profile
5: Very High Confidence	Codebase is mature, clean, and secure. No High or Critical vulnerabilities were found. Follows modern best practices with high test coverage and thoughtful design. **Recommendation:** Code is highly secure at time of audit. Low risk of latent critical issues.	0 High/Critical findings and few Low/Medium severity findings.
4: High Confidence	Code is clean, well-structured, and adheres to best practices. Only Low or Medium-severity issues were discovered. Design patterns are sound, and test coverage is reasonable. Small changes, such as modifying rounding logic, may introduce new vulnerabilities and should be carefully reviewed. **Recommendation:** Suitable for deployment after remediations; consider periodic review with changes.	0 High/Critical findings. Varied Low/Medium severity findings.
3: Moderate Confidence	Medium-severity and occasional High-severity issues found. Code is functional, but there are concerning areas (e.g., weak modularity, risky patterns). No critical design flaws, though some patterns could lead to issues in edge cases. **Recommendation:** Address issues thoroughly and consider a targeted follow-up audit depending on code changes.	1 High finding and ≥ 3 Medium. Varied Low severity findings.
2: Low Confidence	Code shows frequent emergence of Critical/High vulnerabilities (~2/week). Audit revealed recurring anti-patterns, weak test coverage, or unclear logic. These characteristics suggest a high likelihood of latent issues. **Recommendation:** Post-audit development and a second audit cycle are strongly advised.	2-4 High/Critical findings per engagement week.
1: Very Low Confidence	Code has systemic issues. Multiple High/Critical findings (≥5/week), poor security posture, and design flaws that introduce compounding risks. Safety cannot be assured. **Recommendation:** Halt deployment and seek a comprehensive re-audit after substantial refactoring.	≥5 High/Critical findings and overall systemic flaws.

Table of Contents

Project Overview

Project Summary

Project Name	Gamma
Language	Solidity
Codebase	https://github.com/GuardianOrg/HypervisorNFPM-gammapositionmanagers-fuzz, https://github.com/GuardianOrg/MultiPositionManagergamma-posmanager-fuzz
Commit(s)	Initial commit(s): 4fa8879de218db155b108b395609fc359638c507, 48943099b31180c466302da7b2a098e91d284405 Final commit: e9731a4032b798ebbbf1f8baafdebe7dd2e7bdd4, c5b2f2c06c6dee9c0de1cdb7be1fe359584b39af

Audit Summary

Delivery Date	July 22, 2025
Audit Methodology	Static Analysis, Manual Review, Test Suite, Contract Fuzzing

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
Critical	0	0	0	0	0	0
High	2	0	0	0	0	2
Medium	8	0	0	2	0	6
Low	30	0	0	19	0	11
Info	9	0	0	6	0	3

Audit Scope & Methodology

Scope and details:
Algebra (HypervisorNFPM - commit: 4fa8879de218db155b108b395609fc359638c507)

contract,source,total,comment
HypervisorNFPM/contracts/UniProxyETH.sol,181,290,65
HypervisorNFPM/contracts/MultiFeeDistribution.sol,301,525,137
HypervisorNFPM/contracts/HypervisorNFPM.sol,483,743,129
HypervisorNFPM/contracts/ClearingV3NFPM.sol,177,273,64
HypervisorNFPM/contracts/libraries/RewardCalculations.sol,119,157,12
HypervisorNFPM/contracts/libraries/PositionValue.sol,75,96,4
HypervisorNFPM/contracts/libraries/PositionManagementLibrary.sol,192,228,10
source count: {
total: 2514,
source: 1528,
comment: 482,
single: 387,
block: 95,
mixed: 8,
empty: 402,
todo: 0,
blockEmpty: 0,
commentToSourceRatio: 0.29426129426129427}

UniV4 (MultipositionManager - commit: 48943099b31180c466302da7b2a098e91d284405)

contract,source,total,comment
MultiPositionManager/src/UniProxyV2.sol,100,152,28
MultiPositionManager/src/PoolManagerUtils.sol,417,485,11
MultiPositionManager/src/MultiPositionManager.sol,598,778,68
MultiPositionManager/src/ClearingV3.sol,124,164,15
source count: {
total: 1709,
source: 1245,
comment: 139,
single: 92,
block: 47,
mixed: 2,
empty: 244,
todo: 0,
blockEmpty: 0,
commentToSourceRatio: 0.10466867469879518}

Audit Scope & Methodology

Vulnerability Classifications

Severity	Impact: *High*	Impact: *Medium*	Impact: *Low*
Likelihood: *High*	🔴 Critical	🟠 High	🟡 Medium
Likelihood: *Medium*	🟠 High	🟡 Medium	🟢 Low
Likelihood: *Low*	🟡 Medium	🟢 Low	🟢 Low

Impact

High Significant loss of assets in the protocol, significant harm to a group of users, or a core functionality of the protocol is disrupted.

Medium A small amount of funds can be lost or ancillary functionality of the protocol is affected. The user or protocol may experience reduced or delayed receipt of intended funds.

Low Can lead to any unexpected behavior with some of the protocol's functionalities that is notable but does not meet the criteria for a higher severity.

Likelihood

High The attack is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount gained or the disruption to the protocol.

Medium An attack vector that is only possible in uncommon cases or requires a large amount of capital to exercise relative to the amount gained or the disruption to the protocol.

Low Unlikely to ever occur in production.

Audit Scope & Methodology

Methodology

Guardian is the ultimate standard for Smart Contract security. An engagement with Guardian entails the following:

- Two competing teams of Guardian security researchers performing an independent review.
- A dedicated fuzzing engineer to construct a comprehensive stateful fuzzing suite for the project.
- An engagement lead security researcher coordinating the 2 teams, performing their own analysis, relaying findings to the client, and orchestrating the testing/verification efforts.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross-referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts. Comprehensive written tests as a part of a code coverage testing suite.
- Contract fuzzing for increased attack resilience.

Invariants Assessed

During Guardian's review of Gamma, fuzz-testing was performed on the protocol's main functionalities. Given the dynamic interactions and the potential for unforeseen edge cases in the protocol, fuzz-testing was imperative to verify the integrity of several system invariants.

Throughout the engagement the following invariants were assessed for a total of 5,000,000+ runs with a prepared fuzzing suite.

ID	Description	Tested	Passed	Remediation	Run Count
GLOB-01	Liquidity not increased after donate operation	✅	✅	✅	5,000,000+
USER-01	User shares not decreased after withdraw	✅	✅	✅	5,000,000+
USER-02	User token balance not increased after withdraw	✅	✅	✅	5,000,000+
USER-03	User shares not decreased after stake	✅	✅	✅	5,000,000+
USER-04	User shares not increased after unstake	✅	✅	✅	5,000,000+
USER-05	Stake user data not updated for amounts	✅	✅	✅	5,000,000+
USER-06	Unstake user data not updated for amounts	✅	✅	✅	5,000,000+
USER-07	Stake user data not updated for amount	✅	✅	✅	5,000,000+
USER-08	Token not claimed after get rewards	✅	✅	✅	5,000,000+
USER-09	User did not receive tokens after get rewards	✅	✅	✅	5,000,000+

Invariants Assessed

ID	Description	Tested	Passed	Remediation	Run Count
USER-10	Excessive reward amount received after get rewards	✅	✅	✅	5,000,000+
USER-11	Token not claimed after unstake	✅	✅	✅	5,000,000+
USER-12	Slippage error for token A during deposit	✅	✅	✅	5,000,000+
USER-13	Slippage error for token B during deposit	✅	✅	✅	5,000,000+
USER-14	Token balance A not decreased after deposit	✅	✅	✅	5,000,000+
USER-15	Token balance B not decreased after deposit	✅	✅	✅	5,000,000+
USER-16	Ether not decreased after deposit	✅	✅	✅	5,000,000+
USER-17	Token balance not decreased after deposit	✅	✅	✅	5,000,000+
USER-18	User shares not increased after deposit	✅	✅	✅	5,000,000+
POS-01	Total shares not decreased after withdraw	✅	✅	✅	5,000,000+
POS-02	MFD shares not increased after stake	✅	✅	✅	5,000,000+
POS-03	MFD shares not decreased after unstake	✅	✅	✅	5,000,000+
POS-04	MFD shares not increased after stake	✅	✅	✅	5,000,000+
POS-05	User shares not decreased after stake	✅	✅	✅	5,000,000+

Invariants Assessed

ID	Description	Tested	Passed	Remediation	Run Count
POS-06	Hypervisor shares not increased after deposit	✅	✅	✅	5,000,000+
POS-07	MFD shares not increased after deposit	✅	✅	✅	5,000,000+
POS-08	MFD stakes not increased after stake	✅	✅	✅	5,000,000+
POS-09	MFD stakes not decreased after unstake	✅	✅	✅	5,000,000+
POS-10	MFD stakes not increased after stake	✅	✅	✅	5,000,000+
POS-11	MFD stakes not increased after deposit	✅	✅	✅	5,000,000+
POS-12	Improper tick spacing for base lower during rebalance	✅	✅	✅	5,000,000+
POS-13	Improper tick spacing for base upper during rebalance	✅	✅	✅	5,000,000+
POS-14	Improper tick spacing for limit lower during rebalance	✅	✅	✅	5,000,000+
POS-15	Improper tick spacing for limit upper during rebalance	✅	✅	✅	5,000,000+
POS-16	Mismatch stored position base lower	✅	✅	✅	5,000,000+
POS-17	Mismatch stored position base upper	✅	✅	✅	5,000,000+
POS-18	Mismatch stored position limit lower	✅	✅	✅	5,000,000+
POS-19	Mismatch stored position limit upper	✅	✅	✅	5,000,000+

Invariants Assessed

ID	Description	Tested	Passed	Remediation	Run Count
POS-20	Tick mismatch base lower during rebalance	✅	✅	✅	5,000,000+
POS-21	Tick mismatch base upper during rebalance	✅	✅	✅	5,000,000+
POS-22	Tick mismatch limit lower during rebalance	✅	✅	✅	5,000,000+
POS-23	Tick mismatch limit upper during rebalance	✅	✅	✅	5,000,000+
POS-24	Position change base lower during compound	✅	✅	✅	5,000,000+
POS-25	Position change base upper during compound	✅	✅	✅	5,000,000+
POS-26	Position change limit lower during compound	✅	✅	✅	5,000,000+
POS-27	Position change limit upper during compound	✅	✅	✅	5,000,000+
POS-28	No liquidity minted during mint liquidity	✅	✅	✅	5,000,000+
POS-29	Position not increased during mint liquidity	✅	✅	✅	5,000,000+
POS-30	Improper tick spacing tick lower during mint liquidity	✅	✅	✅	5,000,000+
POS-31	Improper tick spacing tick upper during mint liquidity	✅	✅	✅	5,000,000+
POS-32	Position not decreased during decrease liquidity	✅	✅	✅	5,000,000+
POS-33	Liquidity decreased during compound	✅	✅	✅	5,000,000+

Invariants Assessed

ID	Description	Tested	Passed	Remediation	Run Count
POS-34	Amount A not collected during decrease liquidity	✅	✅	✅	5,000,000+
POS-35	Amount B not collected during decrease liquidity	✅	✅	✅	5,000,000+
POS-36	Hypervisor token A balance not increased after deposit	✅	✅	✅	5,000,000+
POS-37	Hypervisor token B balance not increased after deposit	✅	✅	✅	5,000,000+
REW-01	Reward not decreased for token A during compound	✅	✅	✅	5,000,000+
REW-02	Reward not decreased for token B during compound	✅	✅	✅	5,000,000+

Invariants Assessed

During Guardian's review of Gamma, fuzz-testing was performed on the protocol's main functionalities. Given the dynamic interactions and the potential for unforeseen edge cases in the protocol, fuzz-testing was imperative to verify the integrity of several system invariants.

Throughout the engagement the following invariants were assessed for a total of 5,000,000+ runs with a prepared fuzzing suite.

ID	Description	Tested	Passed	Remediation	Run Count
GLOB-01	Unexpected amount of positions stored during rebalance	✅	✅	✅	5,000,000+
GLOB-02	Unexpected positions length change during compound	✅	✅	✅	5,000,000+
GLOB-03	Transfer ownership unexpectedly successful	✅	✅	✅	5,000,000+
POS-01	Invalid tick spacing for lower tick during rebalance	✅	✅	✅	5,000,000+
POS-02	Invalid tick spacing for upper tick during rebalance	✅	✅	✅	5,000,000+
POS-03	Position mismatch for lower tick during rebalance	✅	✅	✅	5,000,000+
POS-04	Position mismatch for upper tick during rebalance	✅	✅	✅	5,000,000+
POS-05	Position mismatch for tick spacing during rebalance	✅	✅	✅	5,000,000+
POS-06	Position mismatch for fee during rebalance	✅	✅	✅	5,000,000+
POS-07	Position mismatch for currency 0 during rebalance	✅	✅	✅	5,000,000+

Invariants Assessed

ID	Description	Tested	Passed	Remediation	Run Count
POS-08	Position mismatch for currency 1 during rebalance	✅	✅	✅	5,000,000+
POS-09	Fees left over for token A during compound	✅	✅	✅	5,000,000+
POS-10	Fees left over for token B during compound	✅	✅	✅	5,000,000+
POS-11	Positions did not increase liquidity during compound	✅	✅	✅	5,000,000+
POS-12	Liquidity mismatch during compound	✅	✅	✅	5,000,000+
POS-13	TokenA LP has not increased during compound	✅	✅	✅	5,000,000+
POS-14	TokenB LP has not increased during compound	✅	✅	✅	5,000,000+
POS-15	Fees left over for token A during claim fee	✅	✅	✅	5,000,000+
POS-16	Fees left over for token B during claim fee	✅	✅	✅	5,000,000+
POS-17	MultiPositionManager balance not increased for token A during claim fee	✅	✅	✅	5,000,000+
POS-18	MultiPositionManager balance not increased for token B during claim fee	✅	✅	✅	5,000,000+
POS-19	Fee mismatch during set fee	✅	✅	✅	5,000,000+
POS-20	Contract has non-zero balance during rescue ERC20	✅	✅	✅	5,000,000+
POS-21	Recipient balance not increased correctly during rescue ERC20	✅	✅	✅	5,000,000+

Invariants Assessed

ID	Description	Tested	Passed	Remediation	Run Count
POS-22	MultiPositionManager balance unchanged during pull liquidity	✅	✅	✅	5,000,000+
POS-23	LP amount unchanged during pull liquidity	✅	✅	✅	5,000,000+
POS-24	Position still has liquidity after pull liquidity	✅	✅	✅	5,000,000+
POS-25	Invalid position passed during position lookup	✅	✅	✅	5,000,000+
USER-01	User token A balance not increased after withdraw	✅	✅	✅	5,000,000+
USER-02	User token B balance not increased after withdraw	✅	✅	✅	5,000,000+
USER-03	User shares not decreased after withdraw	✅	✅	✅	5,000,000+
USER-04	Total shares not decreased after withdraw	✅	✅	✅	5,000,000+
USER-05	User token A balance not decreased after deposit	✅	✅	✅	5,000,000+
USER-06	User token B balance not decreased after deposit	✅	✅	✅	5,000,000+
USER-07	MultiPositionManager token A balance not increased after deposit	✅	✅	✅	5,000,000+
USER-08	MultiPositionManager token B balance not increased after deposit	✅	✅	✅	5,000,000+
USER-09	User shares not increased after deposit	✅	✅	✅	5,000,000+
USER-10	Total shares not increased after deposit	✅	✅	✅	5,000,000+

Invariants Assessed

ID	Description	Tested	Passed	Remediation	Run Count
CLEAR-01	Address not added to free deposit list	✅	✅	✅	5,000,000+
CLEAR-02	Address not removed from free deposit list	✅	✅	✅	5,000,000+

Findings & Resolutions

ID	Title	Category	Severity	Status
H-01	No Share Slippage Protection In deposit	Frontrunning	High	Resolved
H-02	Wrong Usage Of collectRewards Return Values	Logical Error	High	Resolved
M-01	Unfortunate Withdraw Timing Can Lead To Loss	MEV	Medium	Resolved
M-02	Fee Checked Incorrectly	Logical Error	Medium	Resolved
M-03	Tick Bounds Can Be Miscalculated	Logical Error	Medium	Resolved
M-04	Overflow In Price Calculation Not Fixed	Math	Medium	Resolved
M-05	Algebra Oracle Vulnerable To TWAP Lookback DoS	DoS	Medium	Acknowledged
M-06	Rebalance Can Revert During Emergency Mode	DoS	Medium	Resolved
M-07	Clearing Reverts On Inactive Limit Positions	DoS	Medium	Resolved
M-08	decreaseLiquidity Doesn't Collect Funds	Unexpected Behavior	Medium	Acknowledged
L-01	Excess Native Refunds May Revert For Contracts	Best Practices	Low	Acknowledged
L-02	FeeRecipient Update Can Be Frontrun	Documentation	Low	Acknowledged
L-03	Unnecessary Approvals	Unexpected Behavior	Low	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
L-04	Unused ReentrancyGuard In ClearingV3.sol	Informational	Low	Resolved
L-05	Incorrect Error Usage	Informational	Low	Resolved
L-06	Incorrect Price For Negative Tick	Logical Error	Low	Acknowledged
L-07	mintLiquidity Mishandles Fees	Unexpected Behavior	Low	Acknowledged
L-08	getTotalAmountsPlusFees May Be Stale	Warning	Low	Acknowledged
L-09	Possible 0 Token Transfer	Rewards	Low	Resolved
L-10	setNftIds Doesn't Collect Funds	Unexpected Behavior	Low	Acknowledged
L-11	Unnecessary Check In addReward()	Best Practices	Low	Acknowledged
L-12	simulateOneSecondGrowth Is Not Used	Best Practices	Low	Acknowledged
L-13	Slippage Not Enforced For Limit Positions	Frontrunning	Low	Acknowledged
L-14	Unnecessary Assignment	Best Practices	Low	Acknowledged
L-15	Staking Token Can Be Set As Reward Token	Best Practices	Low	Resolved
L-16	Clearing Doesn't Consider Liquidity	Informational	Low	Acknowledged

Findings & Resolutions

ID	Title	Category	Severity	Status
L-17	CEI Pattern Not Followed	Best Practices	Low	Acknowledged
L-18	Withdrawals Can Be Weaponized	Unexpected Behavior	Low	Acknowledged
L-19	Potential 0 Transfer On Withdraw	Validation	Low	Resolved
L-20	ETH Surplus May Not Be Refunded	Logical Error	Low	Resolved
L-21	inMin Is Not Enforced When liquidity = 0	Validation	Low	Acknowledged
L-22	UniProxy.transferETH() Is Permissionless	Informational	Low	Acknowledged
L-23	Whitelisted Depositors Are Not Exempted	Validation	Low	Resolved
L-24	UniProxyETH Is Vulnerable To Tokens With Hooks	Unexpected Behavior	Low	Acknowledged
L-25	Zero-Interval TWAP Always Returns Zero	Logical Error	Low	Resolved
L-26	Misaligned Limit Positions Due To Slot0.tick	Unexpected Behavior	Low	Acknowledged
L-27	Inconsistent Price Threshold Check	Math	Low	Acknowledged
L-28	Lack Of Checkpointing When Setting New Fee	Logical Error	Low	Resolved
L-29	mintLiquidity Causes Accounting Mismatch	Unexpected Behavior	Low	Acknowledged

Findings & Resolutions

ID	Title	Category	Severity	Status
L-30	Zero Shares Received During Deposit	Logical Error	Low	Resolved
I-01	Incorrect Comment For removeRewardToken	Documentation	Info	Resolved
I-02	Incorrect Comment For Managers	Documentation	Info	Acknowledged
I-03	Missing Zero Address Check In Constructor	Validation	Info	Resolved
I-04	Use Require Over Assert	Best Practices	Info	Resolved
I-05	Redundant Fee Initialization	Informational	Info	Acknowledged
I-06	Liquidity May Not Fit Into uint128	Informational	Info	Acknowledged
I-07	Outdated Comments	Documentation	Info	Acknowledged
I-08	Position Manager Can Hold Its Tokens	Unexpected Behavior	Info	Acknowledged
I-09	Ambiguous Error Handling	Error	Info	Acknowledged

H-01 | No Share Slippage Protection In deposit

Category	Severity	Location	Status
Frontrunning	High	MultiPositionManager.sol	Resolved

Description PoC

In MultiPositionManager, deposits do not enforce a minimum share amount, exposing users to front-running attacks that manipulate pool prices to dilute share allocation.

Attack Scenario:
1. Vault holds equal liquidity in two pools (e.g., USDC/WETH, fee tiers 0.3% and 1%).
2. Bob deposits 1000 USDC and 1 WETH, expecting ~1000 shares based on a balanced vault (assume total0=10_000 USDC, total1=10 WETH, totalSupply=10_000).
3. Alice front-runs Bob by manipulating prices across the two pools:
Swaps USDC → WETH in pool A (increasing WETH price).
Swaps WETH → USDC in pool B (lowering WETH price).
1. This skews the getTotalAmounts result:
Total token balances (e.g., total0 = 20,000 USDC, total1 = 20 WETH) double, but ratio remains balanced.
1. Bob's deposit is now worth a smaller portion of the pool, and he receives only 500 shares instead of 1000.
2. Attacker backruns the deposit and swaps back to bring prices and pool back to original values

As a result, when Bob withdraws he receives less than he deposited losing funds in the process. Depending on the pool type and liquidity available, significant funds could be lost which would go to other depositors of the pool.

Recommendation

Introduce a minShares parameter to deposit to ensure depositors receive their expected amount of shares. Also, consider implementing a price deviation check (similar to HypervisorNFPM), which would prevent the price manipulation that is needed for this attack vector.

Resolution

Gamma Team: The issue was resolved in commit c12e771.

H-02 | Wrong Usage Of collectRewards Return Values

Category	Severity	Location	Status
Logical Error	High	HypervisorNFPM.sol: 597	Resolved

Description PoC

HypervisorNFPM._collectAndClaim() incorrectly uses the amounts for the collected pending rewards returned by farmingCenter.collectRewards(). It checks if they are a positive number and claims the rewards only if they are. The problem with this approach is that every time an Algebra position enters or exits the farms, any pending rewards are collected.

This action can be triggered by anyone via call to NonFungiblePositionManager.increaseLiquidity(). The malicious user can add minimum amount of liquidity to the base and limit positions, which will collect any pending rewards. Therefore, if _collectAndClaim() is executed without accrual of new rewards, nothing will be claimed. This will disturb the rewards distribution for MultiFeeDistribution.

For example:
• call increaseLiquidity to stop the rewards claiming if huge rewards have accumulated
• stake their tokens which will call Hypervisor.getReward() , but because both rewards are 0, it will not do anything
• When the rewards are actually sent to the contract, the malicious staker will have received a part of them.

Additional impacts are:
• Rewards being distributed to the wrong receiver because there is a transferReceiver() function which can change it before the next claim.
• the emitted RewardsCollected() event will use wrong values as reward and bonusReward are pending rewards.

Recommendation

```
(uint256 reward, uint256 bonusReward) = farmingCenter.collectRewards(key, tokenId);
farmingCenter.collectRewards(key, tokenId);
// Claim and send to MultiFeeDistribution.  0 amount is max amount.
if (reward > 0) {
farmingCenter.claimReward(key.rewardToken, address(receiver), 0);
uint256 reward = farmingCenter.claimReward(key.rewardToken, address(receiver), 0);
}
if (bonusReward > 0) {
farmingCenter.claimReward(key.bonusRewardToken, address(receiver), 0);
uint256 bonusReward = farmingCenter.claimReward(key.bonusRewardToken, address(receiver), 0);
}
emit RewardsCollected(
tokenId,
reward,
bonusReward
);
```

Resolution

Gamma Team: The issue was resolved in commit af8baa0.

M-01 | Unfortunate Withdraw Timing Can Lead To Loss

Category	Severity	Location	Status
MEV	Medium	PoolManagerUtils.sol: 210-217	Resolved

Description

When a user calls `withdraw`, they provide an `outMin` param which is an array of `amount0/amount1` values that should be returned for each position.

If they call withdraw and while their transaction is pending, the admin calls rebalance and reduces the number of base positions or simply moves the liquidities to different ticks, then the `outMin` array may not correlate to the expected return amounts of the previous positions.

Therefore, the slippage checks via the `outMin` parameter can be exploited to steal value from the withdrawal.

Here is an example scenario of when the admin reduces the number of base positions:
• There are 5 base positions. The admin rebalances to 3 base positions.
• The user had previously initiated a withdraw transaction with roughly 20% of value expected from each base position.
• Now that there are only 3 base positions, the user should expect to receive ~33% of value from each base position.
• The difference between 33% and 20% can be extracted due to MEV.

Recommendation

One potential mitigation would be to require the user to input a base positions array into the withdraw call that would be checked against the current state of the base positions.

More generally, it may be a good idea to enforce a deadline parameter so that the users' withdraw transaction cannot stay pending, potentially while the base positions array is updated due to rebalances.

Resolution

Gamma Team: The issue was resolved in commit 73f92a8.

M-02 | Fee Checked Incorrectly

Category	Severity	Location	Status
Logical Error	Medium	MultiPositionManager.sol: 326	Resolved

Description

When setting the fee that is sent to the treasury, the new fee value is checked incorrectly. Instead, the previous fee value is checked to ensure that it is not less than one.

```
if (fee < 1) revert InvalidFee();
```

Recommendation

Instead, check that the `newFee` parameter is not less than one and revert if it is.

Resolution

Gamma Team: The issue was resolved in commit 73f92a8.

M-03 | Tick Bounds Can Be Miscalculated

Category	Severity	Location	Status
Logical Error	Medium	ClearingV3.sol: 138	Resolved

Description

The `deposit` function in the `UniProxyV2` contract calls `clearance.clearDeposit` to ensure all positions have their current tick within the allowed range. If any are out of range, the call reverts.

However, when there are no limit positions—such as when `limitWidth` is set to zero during a rebalance—the `_checkTicks` function receives zero values for those positions from `getPositions`.

These zero-value entries are still included in the loop that determines the lowest and highest ticks. As a result, they can be incorrectly selected as the min or max tick, distorting the tick bounds.

This may cause a real position with an out-of-range tick to appear valid, allowing the deposit to proceed when it should have reverted.

This undermines the tick range enforcement and introduces the risk of deposits occurring outside of the configured bounds.

Recommendation

In the `getPositions` and `currentTicks` functions, return early if limit positions are not set, similar to how this case is handled elsewhere in the contract.

Resolution

Gamma Team: The issue was resolved in commit 73f92a8.

M-04 | Overflow In Price Calculation Not Fixed

Category	Severity	Location	Status
Math	Medium	ClearingV3NFPM.sol: 191	Resolved

Description

In the previous audit, Issue 31 raised the possibility of overflow for pairs with very large price.

The original price calculation was:

```
uint256 price = FullMath.mulDiv(uint256(sqrtPrice) * uint256(sqrtPrice),
PRECISION, 2**(96 * 2));
```

which was fixed to:

```
FullMath.mulDiv(uint256(sqrtPrice) * 1e18, uint256(sqrtPrice) * 1e18,
2**(96 * 2));
```

This was fixed in the `_deposit` function of `HypervisornNFPM.sol`. However, the original calculation still exists in the `checkPriceChange` function of `ClearingV3NFPM.sol`.

By using the original implementation, overflow can occur when performing:

`uint256(sqrtPriceBefore) * uint256(sqrtPriceBefore)`.

Recommendation

Update the price calculation `ClearingV3NFPM.sol` to the corrected version which prevents overflow.

Resolution

Gamma Team: The issue was resolved in commit af8baa0.

M-05 | Algebra Oracle Vulnerable To TWAP Lookback DoS

Category	Severity	Location	Status
DoS	Medium	ClearingV3NFPM.sol: 210	Acknowledged

Description

In `HypervisorNFPM`, a TWAP-based price check is performed before deposits, using Algebra's Volatility Oracle. The oracle computes TWAP via stored timepoints, which are indexed by a `uint16 timepointIndex` with a maximum of 65,535 entries (`uint16.max`). Timepoints are recorded during swaps, but only once per block.

If a swap occurs in a block and a timepoint is recorded, any subsequent swaps in the same block won't create additional entries. Over time, once the `timepointIndex` overflows and older timepoints are overwritten. The issue lies with how if the caller specifies a `twapInterval` that is older than the oldest stored timepoint, the `_getTimepointsAt` function will revert.

Attack vector:
If an attacker triggers a swap in every block, the oracle fills up and starts overwriting past timepoints. On chains like BSC, where the average block time is ~0.75s, the full 65,535 timepoint buffer can be filled in:

0.75s * 65,535 ≈ 49,000 seconds ≈ 13.6 hours

The `HypervisorNFPM` vault uses a `twapInterval` of 86,400 seconds (24 hours) for stable pools. This interval will eventually become unreachable in the oracle, causing TWAP reads to revert and permanently blocking deposits unless the interval is reduced.

Recommendation

Constraint `twapInterval` to a safe maximum, calculated as: `twapInterval = avg block time * 65,535`. Alternatively, consider using a non-twap oracle such as Chainlink to obtain price.

Resolution

Gamma Team: Acknowledged.

M-06 | Rebalance Can Revert During Emergency Mode

Category	Severity	Location	Status
DoS	Medium	HypervisorNFPM.sol: 521	Resolved

Description

The `rebalance` function burns and re-mints positions, then calls `_approveAndEnterFarming` to approve and enter the new farming position.

However, `_approveAndEnterFarming` does not check whether `isEmergencyWithdrawActivated` is enabled in the `AlgebraEternalFarming` contract.

If emergency mode is active, `enterFarming` will revert inside `farmingCenter`, blocking the entire rebalance process. Without rebalancing, the protocol cannot maintain its intended strategy.

Furthermore, If the current tick moves out of range, certain functions will be blocked due to implemented checks—such as `deposit`, which requires the current tick to be in range.

Recommendation

Add a check for `isEmergencyWithdrawActivated()` in `_approveAndEnterFarming`, and return early if true, similar to how the `isIncentiveDeactivated` check is implemented.

Resolution

Gamma Team: The issue was resolved in commit af8baa0.

M-07 | Clearing Reverts On Inactive Limit Positions

Category	Severity	Location	Status
DoS	Medium	ClearingV3.sol: 160-169	Resolved

Description

`Clearing._checkTicks()` ensures that each tick in `currentTicks` is between `lowestTick` and `highestTick`, which are derived from all base and limit positions.

However, even inactive limit positions (e.g., when `limitWidth = 0`) are included in `currentTicks`. These inactive positions have uninitialized pool keys, defaulting their ticks to 0.

As a result, if `lowestTick > 0` or `highestTick < 0`, `_checkTicks()` will incorrectly revert, causing a denial of service for deposits.

Recommendation

You can change `MultiPositionManager.currentTicks()` to return a magic value for inactive positions and then skip the iteration in the for loop of `_checkTicks` if the `currentTick` is equal to the magic value.

Resolution

Gamma Team: The issue was resolved in commit 73f92a8.

M-08 | decreaseLiquidity Doesn't Collect Funds

Category	Severity	Location	Status
Unexpected Behavior	Medium	HypervisorNFPM.sol: 722-735	Acknowledged

Description

HypervisorNFPM has two functions that allow the owner to manually control positions on behalf of the contract - mintLiquidity and decreaseLiquidity.

The first one can be used to create new positions by adding liquidity and using them to farm incentives. The second one is used to pull liquidity from positions to the contract.

The problem is that decreaseLiquidity doesn't call _zeroBurn and farmingCenter.collectRewards().

This means that:

• if the position being decreased is base or limit the generated fees will be collected together with the removed liquidity and all of them will be attributed to the vault participants, i.e no cut for the protocol team.

• if the position being decreased is not base or limit, all the incentives generated in the farm will be stuck there as there is no way for the contract to call collectRewards with the id of that position. An exception is if the owner calls setNftIds and sets the position as either base or limit to claim the rewards, but this can disrupt the normal flow of the contract.

Recommendation

Call _zeroBurn() at the beginning of decreaseLiquidity and also collect the received incentives if any.

Resolution

Gamma Team: Acknowledged.

L-01 | Excess Native Refunds May Revert For Contracts

Category	Severity	Location	Status
Best Practices	Low	MultiPositionManager.sol: 606-615	Acknowledged

Description

In the _transferIn function, if someone sends too much native currency (like ETH), the contract tries to refund the extra using .transfer.

While that works fine for normal wallets (EOAs), it becomes a problem if the sender is a contract. The .transfer method only forwards 2300 gas, which isn't enough for contracts that need more gas in their receive() or fallback() functions.

So if the sender is a contract that expects to do anything when receiving ETH, this refund logic will revert.

Recommendation

Instead of using .transfer, use .call{value: ...}("") to forward the ETH.

```
(bool success, ) = payable(msg.sender).call{value: msg.value - amount}("");
require(success, "Refund failed");
```

Resolution

Gamma Team: Acknowledged.

L-02 | FeeRecipient Update Can Be Frontrun

Category	Severity	Location	Status
Documentation	Low	HypervisorNFPM.sol: 342	Acknowledged

Description

In `HypervisorNFPM.sol`, the admin sets the `feeRecipient` in each call to `rebalance()`. The comment indicates `_feeRecipient Address of recipient of % of fees since last rebalance`, however this will not always be true.

If a user performs any action such as a deposit or withdraw, it will call `_zeroBurn()` which will collect the fees and transfer them to the current `feeRecipient`.

Recommendation

Note this or update the comment to indicate that the fees may be received to the currently set address in the case of user deposits or withdrawals.

Resolution

Gamma Team: Acknowledged.

L-03 | Unnecessary Approvals

Category	Severity	Location	Status
Unexpected Behavior	Low	ClearingV3.sol: 51-54	Resolved

Description

The ClearingV3.sol contract performs token approvals but does not perform any transfers.

Recommendation

Remove the token approval logic to avoid confusion of the contract's purpose.

Resolution

Gamma Team: The issue was resolved in commit 73f92a8.

L-04 | Unused ReentrancyGuard In ClearingV3.sol

Category	Severity	Location	Status
Informational	Low	ClearingV3.sol: 13	Resolved

Description

The `ReentrancyGuard.sol` contract is inherited but not used in the `ClearingV3.sol`.

Recommendation

Remove the inherited contract to avoid confusion and excess code size.

Resolution

Gamma Team: The issue was resolved in commit 73f92a8.

L-05 | Incorrect Error Usage

Category	Severity	Location	Status
Informational	Low	MultiFeeDistribution.sol: 103C56-103C67	Resolved

Description

`InvalidBurn()` used in check for reward tokens.

Recommendation

TBD

Resolution

Gamma Team: The issue was resolved in commit af8baa0.

L-06 | Incorrect Price For Negative Tick

Category	Severity	Location	Status
Logical Error	Low	ClearingV3NFPM.sol: 200	Acknowledged

Description

The `getSqrtTwapX96` function in the `ClearingV3NFPM` contract computes a square-root TWAP over a specified interval.

However, when the tick difference is negative and not evenly divisible by the `_twapInterval`, the function fails to round the tick down as required.

Instead, it truncates toward zero, resulting in an incorrect (too high) tick value and thus an inaccurate price

Recommendation

Implement rounding toward negative infinity when the tick difference is negative and has a remainder. Specifically, if `tickCumulatives[1] - tickCumulatives[0]` is negative and `(delta % _twapInterval) = 0`, decrement the tick by one.

Resolution

Gamma Team: Acknowledged.

L-07 | mintLiquidity Mishandles Fees

Category	Severity	Location	Status
Unexpected Behavior	Low	HypervisorNFPM.sol: 573	Acknowledged

Description

The `mintLiquidity` function creates a new liquidity position with a fresh `tokenId`. While this new position is entered into a farming position, it is not accessible by `MultiFeeDistribution` for reward claims.

Currently, `getReward` only claims rewards for the base and limit `tokenIds` set in the vault. As a result, rewards for the newly minted position remain unclaimable until the vault owner explicitly calls `setNftIds` to update one of the tracked token IDs.

On a similar note, when `setNftIds` is called, the rewards for the current base and limit positions are not collected first. As result, these rewards are lost once the new `tokenIds` are written.

Recommendation

1. Consider implementing a `getReward` function that allows for the passing in of a `tokenId` to allow rewards to be claimed for additional positions beyond the base and limit positions.

2. Consider calling `getReward` on the current base and limit positions before `setNftIds` updates.

Resolution

Gamma Team: Acknowledged.

L-08 | getTotalAmountsPlusFees May Be Stale

Category	Severity	Location	Status
Warning	Low	HypervisorNFPM.sol: 457	Acknowledged

Description

The `getTotalAmountsPlusFees` function returns the total of `token0` and `token1` including fees, using `calculatePositionFee` to compute the latest fees and add them to the currently owed amounts. However, this doesn't account for how Algebra pools handle rebase tokens/donations.

At the start of major pool interactions, if the actual token balances exceed expected reserves, the surplus is treated as a donation and distributed to active liquidity providers as additional fees—provided there is non-zero liquidity.

Since `getTotalAmountsPlusFees` doesn't account for this, these additional fees may not be reflected in the returned values.

While this isn't an issue for core `HypervisorNFPM` functions (which call `zeroBurn` first), it could lead to stale or slightly inaccurate fee values when calling `getTotalAmountsPlusFees` directly.

Recommendation

Be aware that `getTotalAmountsPlusFees` may be stale or slightly inaccurate in the following scenario.

Resolution

Gamma Team: Acknowledged.

L-09 | Possible 0 Token Transfer

Category	Severity	Location	Status
Rewards	● Low	MultiFeeDistribution.sol: 462	Resolved

Description

`MultiFeeDistribution._updateReward()` transfers `newRewards / fee` as a fee to the owner unconditionally.

If one of the reward tokens is a token that reverts on 0 transfers, users can grief the actions of the contract by sending as little as 1 wei of that token (if this token wasn't accrued currently).

Recommendation

Check if the amount after the division is positive

```
diff
•                   if (newRewards > 0) {
+                   if (newRewards / fee > 0) {
IERC20(rewardToken).safeTransfer(owner(), newRewards/fee);
currentBalance = IERC20(rewardToken).balanceOf(address(this));
newRewards = newRewards - (newRewards/fee);  // Actual new rewards after fee
}
```

Resolution

Gamma Team: The issue was resolved in commit af8baa0.

L-10 | setNftIds Doesn't Collect Funds

Category	Severity	Location	Status
Unexpected Behavior	Low	HypervisorNFPM.sol: 709-713	Acknowledged

Description

`HypervisorNFPM.setNftIds()` is a helpful utility function which can be used by the owner to change any of the two position ids.

Similar to the issue described in the other report about `decreaseLiquidity()`, `setNftIds` doesn't call `_zeroBurn()` and `_collectAndClaim()`.

The following problem exists if the function is used to change the ids for a prolonged period of time - accrued fees and incentive rewards will be either lost if the old ids are not restored or they will be wrongly distributed because they will not be counted towards the total funds for any new deposits in the window where `setNftId` took place.

Recommendation

Call `_zeroBurn()` and `_collectAndClaim()` before changing the ids.

Resolution

Gamma Team: Acknowledged.

L-11 | Unnecessary Check In addReward()

Category	Severity	Location	Status
Best Practices	Low	MultiFeeDistribution.sol: 155-161	Acknowledged

Description

MultiFeeDistribution.addReward() checks if the token to be added is an active reward twice, which is unnecessary and only causes higher gas expenditure.

```
(bool isRewardTokenExist, ) = _isRewardTokenExist(_rewardToken);
if (isRewardTokenExist) revert ActiveReward();
if (managers[msg.sender]) revert InsufficientPermission();
for (uint i; i < rewardTokens.length; i ++) {
if (rewardTokens[i] = _rewardToken) revert ActiveReward();
}
```

Recommendation

Remove one of the checks, for example the first one since it's more expensive.

```
-       (bool isRewardTokenExist, ) = _isRewardTokenExist(_rewardToken);
-       if (isRewardTokenExist) revert ActiveReward();
if (managers[msg.sender]) revert InsufficientPermission();
for (uint i; i < rewardTokens.length; i ++) {
if (rewardTokens[i] == _rewardToken) revert ActiveReward();
}
rewardTokens.push(_rewardToken);
```

Resolution

Gamma Team: Acknowledged.

L-12 | simulateOneSecondGrowth Is Not Used

Category	Severity	Location	Status
Best Practices	Low	RewardCalculations.sol	Acknowledged

Description

The `RewardCalculations.simulateOneSecondGrowth()` function is never called in the codebase.

Recommendation

Consider removing it.

Resolution

Gamma Team: Acknowledged.

L-13 | Slippage Not Enforced For Limit Positions

Category	Severity	Location	Status
Frontrunning	Low	PoolManagerUtils.sol: 124	Acknowledged

Description

The `inMin` and `outMin` parameters are user-supplied slippage controls used during `withdraw`, `rebalance` and `compound` to ensure the user or protocol uses or receives a minimum acceptable amount of tokens when adding or removing liquidity.

However, this check is not applied when minting or burning liquidity from limit positions. Instead, `_mintLiquidityForAmounts` and `burnLiquidityForShare` are called with hardcoded zero slippage tolerance:

```
(uint256 amountOut0, uint256 amountOut1) = burnLiquidityForShare(
poolManager,
limitPositions[i],
shares,
totalSupply,
[uint256(0), uint256(0)] // @audit zero slippage tolerance
);
```

This bypasses the intended slippage protection and opens up the vault to price manipulation or frontrunning attacks, where:
• A user may receive significantly less than expected during a `withdraw`.
• The vault may unintentionally deploy liquidity in a distorted ratio during `rebalance` or `compound`

Recommendation

1. Apply `inMin` and `outMin` by passing the relevant slippage values into `_mintLiquidityForAmounts` and `burnLiquidityForShare` for limit positions instead of [0, 0].
2. In `UniProxyV2`, `getOutMinForShares` should also calculate `outMin` for limit positions.

Resolution

Gamma Team: Acknowledged.

L-14 | Unnecessary Assignment

Category	Severity	Location	Status
Best Practices	Low	HypervisorNFPM.sol: 84	Acknowledged

Description

The constructor of `HyperVisorNFPM` sets the value of `incentiveMaker` twice - once outside the if statement and once inside of it - to the same value, which is redundant.

Recommendation

Delete the second assignment.

Resolution

Gamma Team: Acknowledged.

L-15 | Staking Token Can Be Set As Reward Token

Category	Severity	Location	Status
Best Practices	Low	MultiFeeDistribution.sol: 141-145	Resolved

Description

`MultiFeeDistribution.setStakingToken()` doesn't check if `_stakingToken` is not an active reward token. This allows bypassing the check in `addReward()` and adding a reward token as a staking token.

Recommendation

Consider reverting the `setStakingToken()` transaction if `_stakingToken` is an active reward token.

Resolution

Gamma Team: The issue was resolved in commit af8baa0.

L-16 | Clearing Doesn't Consider Liquidity

Category	Severity	Location	Status
Informational	Low	Clearing.sol	Acknowledged

Description

When ticks are checked in `Clearing`, all positions are included, even inactive ones (i.e `liquidity = 0`). It may be possible that some positions have 0 liquidity after withdrawals or if a there wasn't enough liquidity when limit positions were created. These positions will still impact the clearing validation.

Recommendation

Make sure that's the correct behavior.

Resolution

Gamma Team: Acknowledged.

L-17 | CEI Pattern Not Followed

Category	Severity	Location	Status
Best Practices	Low	MultiPositionManager.sol	Acknowledged

Description

`MultiPositionManager.deposit()` and `MultiPositionManager.withdraw()` functions doesn't follow the `CEI` pattern - they transfer tokens before writing to the contract state.

This gives the execution flow the recipient for pools with native token or pools with tokens with hooks. This opens up possibilities for read-only reentrancy and unexpected behavior.

For example, a user may use the fund transfer in `withdraw()` to transfer more tokens to the `MultiPositionManager` and cause wrong `Withdraw()` event data emission.

Recommendation

Consider following the `CEI` pattern.

Resolution

Gamma Team: Acknowledged.

L-18 | Withdrawals Can Be Weaponized

Category	Severity	Location	Status
Unexpected Behavior	Low	MultiPositionManager.sol	Acknowledged

Description

When users withdraw, funds are taken out of the base and limit positions. Because there isn't any delay between depositing and minting, users can atomically withdraw their deposits and pull liquidity out of the positions and griefing the manager from collecting fees.

Recommendation

Consider implementing a delay between deposit and withdraw

Resolution

Gamma Team: Acknowledged.

L-19 | Potential 0 Transfer On Withdraw

Category	Severity	Location	Status
Validation	Low	MultiPositionManager.sol: 201-202	Resolved

Description

`MultiPositionManager.withdraw()` transfers `amount0` and `amount1` of the currencies unconditionally, even if they are zeroes. If used with some tokens that revert on 0 transfers, withdrawals may be blocked.

Recommendation

Skip the transfers if the amounts are 0s.

Resolution

Gamma Team: The issue was resolved in commit b7667f7.

L-20 | ETH Surplus May Not Be Refunded

Category	Severity	Location	Status
Logical Error	Low	MultiPositionManager.sol: 607	Resolved

Description PoC

In the MultiPositionManager.deposit() function, when interacting with pools where token0 = address(0) (i.e., native token), the _transferIn() function is responsible for handling incoming native tokens and refunding any excess via the msg.value > amount check. However, a logic flaw exists: if amount = 0, _transferIn() returns early without executing the refund logic.

This results in a scenario where any native tokens sent along with the call are silently retained, causing a loss of funds for the user. This can happen when:
• The pool's current tick is above the upper bound of both the base and limit positions, making the total0 = 0.
• The deposit is contributing only token1, so amount = 0 is passed to _transferIn().
• There are no idle token0 funds or pending fees in the contract, so no internal balance offsets this.
• The user unintentionally sends a nonzero msg.value, expecting a partial refund.

Currently, this situation is prevented by the Clearing contract, which blocks such deposits. However, future changes to the clearing logic or the whitelist may reopen this vulnerability.

Recommendation

Don't skip the refund for native tokens.

```
function _transferIn(address from, Currency currency, uint256 amount) internal {
•   if (amount = 0) return;
if (currency.isAddressZero()) {
if (msg.value < amount) revert InvalidDepositAmount(amount);
if (msg.value > amount)
payable(msg.sender).transfer(msg.value - amount);
} else if (amount = 0) {
IERC20(Currency.unwrap(currency)).safeTransferFrom(from, address(this), amount);
}
}
```

Resolution

Gamma Team: The issue was resolved in commit b7667f7.

L-21 | inMin Is Not Enforced When liquidity = 0

Category	Severity	Location	Status
Validation	Low	PoolManagerUtils.sol: 188-193	Acknowledged

Description

`PoolManagerUtils._mintLiquidityForAmounts()` enforces the `inMin` constraint only when `liquidity > 0`.

However, `mintLiquidities()` may call `_mintLiquidityForAmounts()` with a positive `liquidity` value but with actual token amounts equal to zero—for instance, if the contract's token balances were already depleted before this call.

In such a case, the function skips minting (since `liquidity = 0`), and no check is performed against `inMin`, even if the expected minimum input amount was positive. This leads to a silent mismatch between intended and actual behavior.

Recommendation

Consider validating the added amounts against `inMin` even if there was no liquidity added.

Resolution

Gamma Team: Acknowledged.

L-22 | UniProxy.transferETH() Is Permissionless

Category	Severity	Location	Status
Informational	Low	UniProxyV2.sol: 90-93	Acknowledged

Description

The UniProxy.transferETH() function used for making ETH transfers is external and permissionless which means any funds in the contract can be taken out by users.

Recommendation

Never hold funds in the contract.

Resolution

Gamma Team: Acknowledged.

L-23 | Whitelisted Depositors Are Not Exempted

Category	Severity	Location	Status
Validation	Low	UniProxyV2.sol: 55	Resolved

Description

The `ClearingV3` contract contains a list with depositors (`freeDepositList`) that should not be subject of the clearing logic.

However, this list is never utilized and in result the whitelisted entities are no different than every other user.

Recommendation

Consider executing `clearDeposit` in `UniProxyV2` only if `msg.sender` is not whitelisted.

```
+ if (clearance.getListed(pos, msg.sender)) {
clearance.clearDeposit(to, pos);
+ }
```

Resolution

Gamma Team: The issue was resolved in commit b7667f7.

L-24 | UniProxyETH Is Vulnerable To Tokens With Hooks

Category	Severity	Location	Status
Unexpected Behavior	Low	UniProxyETH.sol	Acknowledged

Description

`UniProxyETH.depositETH()` and `UniProxyETH.depositETHAndStake()` perform these three actions in the following order:

• `clearDeposit()`

• transfer token from user

• execute the `Hypervisor` deposit

If the token to be transferred has hooks, the depositor will be able to bypass the clearing mechanism, including manipulating the price outside of the bounds of the `TWAP` and effectively draining the contract.

Recommendation

Move the token transfer from the second step in the description before the call to `clearDeposit()` and use `deposit0` and `deposit1` as transferred amounts. This is okay since there is a refund mechanism at the end of the function.

Resolution

Gamma Team: Acknowledged.

L-25 | Zero-Interval TWAP Always Returns Zero

Category	Severity	Location	Status
Logical Error	Low	ClearingV3NFPM.sol: 200	Resolved

Description

The `getSqrtTwapX96` function in the `ClearingV3NFPM` contract is intended to return the square-root TWAP price over a specified interval.

When the interval is zero, it should return the current price. However, while the current price is fetched, it is not assigned to the function's return variable.

As a result, calls with a zero interval always return zero. Consequently, when `checkPriceChange` uses this function, a zero interval leads to a zero price.

This causes the price-change threshold check to be exceeded and the function to revert, effectively blocking deposits.

Recommendation

Fix `getSqrtTwapX96` so that when the interval is zero, the fetched current price is correctly assigned to the return variable

Resolution

Gamma Team: The issue was resolved in commit af8baa0.

L-26 | Misaligned Limit Positions Due To Slot0.tick

Category	Severity	Location	Status
Unexpected Behavior	Low	MultiPositionManager.sol: 707	Acknowledged

Description PoC

In `MultiPositionManager`, limit positions are centered around `slot0.tick`, assuming it reflects the current price. However, this is inaccurate when a swap ends exactly at a tick boundary.

As per Uniswap's swap logic, when `result.sqrtPriceX96 = step.sqrtPriceNextX96` at the end of a swap step, and the direction is `zeroForOne`, the protocol sets the current tick to `tickNext - 1`.

Example scenario:
• Initial setup: Stable pool tick range [-20, 20], current tick: 0, initial price: X
• A swap `token1 > token0` moves tick to 1
• An equal swap back from `token0 > token1` brings price back to exactly X
• However, current tick is now -1, not 0 due to Uniswap's handling of `zeroForOne` swaps

As a result, limit positions of 1 tick width are deployed at range [-2,0], leading to inefficient limit position placement and therefore missed fee opportunities.

Recommendation

Consider allowing admin to specify exactly which tick and ranges to deploy limit liquidity in (similar to how `HypervisorNFPM` is implemented).

Alternatively, use `slot0.sqrtPriceX96` to derive the actual current tick using `TickMath.getSqrtPriceAtTick()`. This ensures limit positions are accurately centered at the true price.

Resolution

Gamma Team: Acknowledged.

L-27 | Inconsistent Price Threshold Check

Category	Severity	Location	Status
Math	Low	ClearingV3NFPM.sol: 192-193	Acknowledged

Description

`ClearingV3NFPM.checkPriceChange()` compares the Algebra pool's spot price to a given TWAP price and reverts if the price deviates more than a set threshold.

While the computation works when `spotPrice > twapPrice`, it fails in the opposite case - it calculates the price change as if the price went from `spotPrice` to `twapPrice` instead of `twapPrice` to `spotPrice`.

The code doing the math is the following:

```
if (price * 10_000 / priceBefore > _priceThreshold || priceBefore * 10_000 / price >
_priceThreshold)
revert("Price change overflow");
```

Let's say:
• `priceBefore = $200` (TWAP price)
• `price = $100` (Spot price)

This is a 50% deviation from the `TWAP`. However, value checked against `_priceThreshold` will be the maximum between
1. `price * 10_000 / priceBefore` = $100 * 10000 / 200 = 5000$
2. `priceBefore * 10_000 / price` = $200 * 10000 / 100 = 20000$ (100%)

In result, the price change from `$200` to `$100` will be considered as a `100%` change instead of `50%`.

Recommendation

Compute an absolute delta value between the two prices and calculate the deviation as `delta * 10_000 / priceBefore + 10_000`.

Resolution

Gamma Team: Acknowledged.

L-28 | Lack Of Checkpointing When Setting New Fee

Category	Severity	Location	Status
Logical Error	Low	HypervisorNFPM.sol: 656-660	Resolved

Description

In the hypervisor contract, the admin is able to set the percentage of fees that will collected as protocol fees.

However, the new value is set without firstly calling `_zeroBurn()` which transfers the relevant fee values to the `feeRecipient`.

Therefore, the admin can set this value to its lowest value, e.g. 1, which will transfer them all fees that are collected.

Contrast this to the `MultiPositionManager` contract which does successfully call `_zeroBurn()` prior to setting the new fee.

Similarly, the fees are not checkpointed in `MultiFeeDistribution.sol` either when resetting the fee.

Recommendation

Call `_zeroBurn()` prior to updating the fee value in `HypervisorNFPM.sol`. Call `_updateRewards()` in `MultiFeeDistribution.sol`.

Resolution

Gamma Team: The issue was resolved in commit af8baa0.

L-29 | mintLiquidity Causes Accounting Mismatch

Category	Severity	Location	Status
Unexpected Behavior	Low	HypervisorNFPM.sol: 746-772	Acknowledged

Description

When `HypervisorNFPM.mintLiquidity()` is invoked, funds from the contract are provided as a liquidity for a new Algebra position, however these funds are not accounted for in `getDepositAmount()` and `withdraw()`.

This means that almost any call to `mintLiquidity()` will result in broken accounting - loss for current depositors and profit for the new ones if liquidity is pulled from that position in the future.

Recommendation

Consider:
1. tracking the balances of the minted positions via this function
2. pulling funds from them on `withdraw()`

Resolution

Gamma Team: Acknowledged.

L-30 | Zero Shares Received During Deposit

Category	Severity	Location	Status
Logical Error	Low	HypervisorNFPM.sol: 189	Resolved

Description

The `deposit` function in `HypervisorNFPM` does not verify that the calculated shares are greater than zero.

This issue was previously raised and partially addressed by adding a slippage check on `deposit0` and `deposit1`.

However, that check does not prevent zero-share outcomes caused by rounding errors, price manipulation, or small deposit amounts.

This can lead to situations where a user deposits tokens but receives zero shares in return—resulting in loss of funds.

Notably, the `MultiPositionManager` implementation explicitly guards against this by checking that `shares > 0`.

Recommendation

Add an explicit check:

`require(shares > 0, "ZeroShares");`

Resolution

Gamma Team: The issue was resolved in commit 73f92a8.

I-01 | Incorrect Comment For removeRewardToken

Category	Severity	Location	Status
Documentation	Info	MultiFeeDistribution.sol: 166	Resolved

Description

The comment above `MultiFeeDistribution.removeRewardToken` indicates:

`* @notice Add a new reward token to be distributed to stakers.`

Recommendation

Update the comment to indicate that the function removes a reward token instead of adds.

Resolution

Gamma Team: The issue was resolved in commit af8baa0.

I-02 | Incorrect Comment For Managers

Category	Severity	Location	Status
Documentation	Info	MultiFeeDistribution.sol: 66	Acknowledged

Description

The comment for the `managers` mapping in `MultiFeeDistribution` states that the managers are "Addresses approved to call mint". However, these managers can only add or remove reward tokens.

Recommendation

Update the comment to indicate the managers' ability.

Resolution

Gamma Team: Acknowledged.

I-03 | Missing Zero Address Check In Constructor

Category	Severity	Location	Status
Validation	Info	MultiPositionManager.sol: 109	Resolved

Description

In the constructor, `_token0` and `_token1` are assigned to `currency0` and `currency1`. While `_token0` may be the zero address to represent native ETH, `_token1` should never be zero. Currently, there's no check to enforce this.

Recommendation

Add a validation check:

`if (_token1 = address(0)) revert ZeroAddress();`

Resolution

Gamma Team: Resolved.

I-04 | Use Require Over Assert

Category	Severity	Location	Status
Best Practices	Info	MultiPositionManager.sol: 818	Resolved

Description

In the `deposit` and `calcSharesAndAmounts` function, the code uses `assert(shares > 0);`, which triggers a `panic: assertion failed (0x01)` if the condition fails.

This revert reason is opaque and hinders debugging. `assert` should be reserved for testing internal errors and invariants.

In production, `require` should be used instead to provide a clear and meaningful revert reason.

Recommendation

Replace with a require and a clear error message:

`require(shares > 0, "ZeroShares");`

Resolution

Gamma Team: Resolved.

I-05 | Redundant Fee Initialization

Category	Severity	Location	Status
Informational	Info	HypervisorNFPM.sol: 44	Acknowledged

Description

In `HypervisorNFPM`, the fee variable is initialized to 1 at declaration, and then set to 1 again in the constructor.

Since both values are the same, the initial assignment is unnecessary. The same redundant pattern is present in the `MultiFeeDistribution` contract.

Recommendation

Remove the redundant fee = 1 assignment and allow the constructor to handle the initialization.

Resolution

Gamma Team: Acknowledged.

I-06 | Liquidity May Not Fit Into uint128

Category	Severity	Location	Status
Informational	Info	Hypervisor.sol	Acknowledged

Description

Whenever the Hypervisor is minting new liquidity, it should be a number fitting in `uint128`. However, it's possible to have it overflow this type, especially when the price is large.

In this case, the Hypervisor can be put in a state where all of its liquidity adding functions are blocked - `deposit()` when `directDeposit()` is turned on, `rebalance()` because positions are unconditional and they spend the whole balance, etc...

Recommendation

Currently, `decreaseLiquidity()` can be used to pull liquidity out of the positions if needed, but also consider adding more granular control to `rebalance()`, i.e not using the whole balance, but part of it or having optional positions.

Resolution

Gamma Team: Acknowledged.

I-07 | Outdated Comments

Category	Severity	Location	Status
Documentation	Info	MultiFeeDistribution.sol	Acknowledged

Description

There are some outdated comments in `MultiFeeDistribution`. For example, it's stated that staked tokens cannot be withdrawn for `defaultLockDuration`, but there is no such functionality implemented.

Recommendation

Consider refactoring the comments.

Resolution

Gamma Team: Acknowledged.

I-08 | Position Manager Can Hold Its Tokens

Category	Severity	Location	Status
Unexpected Behavior	Info	MultiPositionManager.sol	Acknowledged

Description

`MultiPositionManager.deposit()` doesn't allow specifying the position manager contract itself as a recipient, but the ERC20 `transfer()` function is left unchanged which allows direct transfers to the contract. This breaks the expectation that the contract will not hold its tokens.

Recommendation

If the goal is for the contract to never hold the tokens, override the transfer function and revert if the recipient is `MultiPositionManager`.

Resolution

Gamma Team: Acknowledged.

I-09 | Ambiguous Error Handling

Category	Severity	Location	Status
Error	Info	MultiPositionManager.sol: 147	Acknowledged

Description

`MultiPositionManager.deposit()` reverts with `ZeroAddress()` error if the `to` parameter is equal to `address(0)`, but also if it's equal to `address(this)`, which can be deceiving for integrators.

Recommendation

Consider using a more general error, like `InvalidAddress()`.

Resolution

Gamma Team: Acknowledged.

Disclaimer

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Guardian to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. Guardian’s position is that each company and individual are responsible for their own due diligence and continuous security. Guardian’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by Guardian is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

Notice that smart contracts deployed on the blockchain are not resistant from internal/external exploit. Notice that active smart contract owner privileges constitute an elevated impact to any smart contract’s safety and security. Therefore, Guardian does not guarantee the explicit security of the audited smart contract, regardless of the verdict.

About Guardian

Founded in 2022 by DeFi experts, Guardian is a leading audit firm in the DeFi smart contract space. With every audit report, Guardian upholds best-in-class security while achieving our mission to relentlessly secure DeFi.

To learn more, visit https://guardianaudits.com

To view our audit portfolio, visit https://github.com/guardianaudits

To book an audit, message https://t.me/guardianaudits