GUARDIAN

Impermax

Tokenized Aerodrome Positions

Security Assessment

August 8th, 2025

Summary

Audit Firm Guardian

Prepared By Owen Thurm, Nicholas Chew, 0xCiphky, Michael Lett

Client Firm Impermax

Final Report Date August 8, 2025

Audit Summary

Impermax engaged Guardian to review the security of their Impermax V3 Tokenized Aerodrome Positions. From the 30th of July to the 5th of August, a team of 4 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: **Base**

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

📊 PoC test suite: https://github.com/GuardianOrg/impermax-v3-core-impermaxtokenizedaerodromeposition-team1 , https://github.com/GuardianOrg/impermax-v3-core-impermaxtokenizedaerodromeposition-team2 , https://github.com/GuardianOrg/impermax-v3-core-impermaxtokenizedaerodromeposition-fuzz

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 Information

Project Overview 5

Audit Scope & Methodology 6

Smart Contract Risk Assessment

Invariants Assessed 9

Findings & Resolutions 11

Addendum

Disclaimer 23

About Guardian 24

Project Overview

Project Summary

Project Name	Impermax
Language	Solidity
Codebase	https://github.com/Impermax-Finance/impermax-v3-core
Commit(s)	Initial commit: 9d9561e104f8aea60373bcdab46b6582f618a81e Final commit: 8056dff0641b314884213c84ee4a1dace1d9ca97

Audit Summary

Delivery Date	August 8, 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	1	0	0	0	0	1
● Medium	2	0	0	1	0	1
● Low	6	0	0	6	0	0
● Info	2	0	0	1	0	1

Audit Scope & Methodology

Scope and details:

```
contract,source,total,comment
impermax-v3-core/contracts/extensions/TokenizedAeroCLPosition.sol,187,257,18
source count: {
  total: 257,
  source: 187,
  comment: 18,
  single: 12,
  block: 6,
  mixed: 2,
  empty: 54,
  todo: 0,
  blockEmpty: 0,
  commentToSourceRatio: 0.0962566844919786}
```

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 Impermax, 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 10,000,000+ runs with a prepared fuzzing suite.

ID	Description	Tested	Passed	Remediation	Run Count
AERO-01	Liquidity should be added to the pool	✅	✅	✅	10M+
AERO-02	Tick lower must be less than tick upper	✅	✅	✅	10M+
AERO-03	Swap should return non-zero output amount	✅	❌	✅	10M+
LIQUI-01	After a successful call to restructureBadDebt function, the position should NOT be underwater.	✅	✅	✅	10M+
BORROW-01	When borrowAmount is 0, borrowedBalance should remain unchanged.	✅	✅	✅	10M+
BORROW-02	After borrow the user's position is never liquidatable	✅	❌	❌	10M+
BORROW-03	After borrow the user's position is never underwater	✅	✅	✅	10M+
REDEEM-01	After a successful remove from collateral call position should not be liquidatable.	✅	✅	✅	10M+
REDEEM-02	After a successful remove from collateral call position should not be underwater.	✅	✅	✅	10M+
CLM-01	claimPositionV3: call to claim failed	✅	❌	❌	10M+

Invariants Assessed

ID	Description	Tested	Passed	Remediation	Run Count
CLAIM-01	After a successful claim position call position should not be liquidatable.	✅	❌	❌	10M+
CLAIM-02	After a successful claim position call position should not be underwater.	✅	✅	✅	10M+
SPLIT-01	Split should not revert with invalid token ID when splitting 100%	✅	❌	✅	10M+
COLL-01	TokenizedAeroCLPosition.redeem should never revert	✅	✅	✅	10M+
GLOBAL-01	There should never be a position that is underwater but not liquidatable	✅	✅	✅	10M+
GLOBAL-02	TokenizedAeroCLPosition should not hold token0 or token1	✅	❌	❌	10M+

Findings & Resolutions

ID	Title	Category	Severity	Status
H-01	ETH Refund Can Revert Critical Functions	DoS	● High	Resolved
M-01	Full Split Causes Revert	Logical Error	● Medium	Resolved
M-02	getPositionData Will Revert For Certain Prices	DoS	● Medium	Acknowledged
L-01	Unclaimed Fees Are Lost After mint	Logical Error	● Low	Acknowledged
L-02	Lack Of Slippage Protection	MEV	● Low	Acknowledged
L-03	ecrecover Allows Signature Malleability	Signatures	● Low	Acknowledged
L-04	Unclaimed Dust After split	Warning	● Low	Acknowledged
L-05	Rewards Cannot Be Claimed By EOA	Unexpected Behavior	● Low	Acknowledged
L-06	Unused Tokens Not Returned	Logical Error	● Low	Acknowledged
I-01	Naming Convention For _addGauge	Informational	● Info	Resolved
I-02	Gas Optimization For nonReentrant Modifier	Gas Optimization	● Info	Acknowledged

H-01 | ETH Refund Can Revert Critical Functions

Category	Severity	Location	Status
DoS	● High	TokenizedAeroCLPosition.sol: 167	Resolved

Description

The increaseLiquidity function in the TokenizedAeroCLPosition contract allows users to add liquidity to an existing position.

It does so by withdrawing the user's position from the gauge, calling increaseLiquidity on the NonfungiblePositionManager with the original and additional amounts, and then redepositing the position.

However, the NonfungiblePositionManager's increaseLiquidity function ends by calling refundETH, which sends any residual ETH in the contract back to msg.sender. In this context, msg.sender is the TokenizedAeroCLPosition contract itself.

Since the contract lacks a fallback function, the refund fails and causes the entire transaction to revert.

A malicious actor could exploit this by sending a small amount of ETH to the NonfungiblePositionManager contract before a increaseLiquidity call, ensuring that the refundETH call fails and reverts the transaction.

The same issue exists in the split function, which also calls mint on the NonfungiblePositionManager, triggering a similar refund.

This is more critical since split is used in liquidation flows—meaning an attacker could block or delay liquidations by intentionally triggering a refund failure.

Recommendation

Consider adding a fallback function to ensure the contract can safely receive ETH refunds and avoid unexpected reverts.

Resolution

Impermax Team: The issue was resolved in commit 1f6e4b3.

M-01 | Full Split Causes Revert

Category	Severity	Location	Status
Logical Error	Medium	TokenizedAeroCLPosition.sol: 167	Resolved

Description

The `split` function allows users to split their position by a specified percentage, with 100% being the maximum.

However, if a user attempts a full (100%) split, the `decreaseAndMint` function in the `NfpmAeroInteractions` library burns the original `tokenId`, since no liquidity remains in the original position.

The issue arises when the `split` function subsequently attempts to redeposit the now-burned `tokenId` into the gauge.

This causes a revert, making a full split impossible. Additionally, because the original token is burned without claiming any pending fees from the gauge, those rewards are lost for the user.

A similar issue occurs when a user passes in 0% to `split` — the function attempts to mint a new position with no liquidity, which results in the new NFT never being minted.

Recommendation

Prevent 0% and 100% splits by modifying the logic to require `percentage > 0 & percentage < 1e18`, or alternatively, add checks to ensure fees are claimed and that the original token is not redeposited after being burned.

Resolution

Impermax Team: The issue was resolved in commit 9d9a4ee.

M-02 | getPositionData Will Revert For Certain Prices

Category	Severity	Location	Status
DoS	Medium	TokenizedAeroCLPosition.sol: 108	Acknowledged

Description

The getPositionData function is intended to return price and liquidity information for a given position.

It computes values such as currentPrice, lowestPrice, and highestPrice using the current price and a user-defined safety margin. These values are constrained using the safe160 helper to ensure they fit within a uint160.

However, in certain token pairs where the price can near the maximum limit representable in Uniswap V3, the computed highestPrice can overflow the uint160 range.

When this occurs, the safe160 check will revert the transaction, even though the position itself remains valid within Uniswap.

This is particularly problematic because getPositionData is used in several critical functions, including liquidation checks. A revert in this context could block or delay important protocol operations.

Recommendation

Ensure newly added token pairs do not result in price ranges exceeding the uint160 max, as this can disrupt core functions.

Resolution

Impermax Team: Acknowledged.

L-01 | Unclaimed Fees Are Lost After mint

Category	Severity	Location	Status
Logical Error	● Low	TokenizedAeroCLPosition.sol: 133	Acknowledged

Description

When `mint` is called, it deposits into the gauge, which triggers a collection of any accrued LP fees. These fees are transferred to `TokenizedAeroCLPosition`.

However, unless the caller explicitly invokes `skim`, these tokens remain unclaimed and can be taken by anyone.

As a result, the user who called `mint` may unintentionally forfeit their accrued LP fees if they do not immediately follow up with a `skim`.

Recommendation

Consider calling `skim(msg.sender)` at the end of the `mint` function.

Resolution

Impermax Team: Acknowledged.

L-02 | Lack Of Slippage Protection

Category	Severity	Location	Status
MEV	● Low	NfpmAeroInteractions.sol: 49	Acknowledged

Description

The increaseLiquidity function enables users to add liquidity to their existing position. However, both amount0Min and amount1Min are hardcoded to zero, meaning no slippage protection is applied during the minting process. This exposes users to unfavorable price movements or MEV attacks.

The same issue exists in the split function, which reduces a position by a specified percentage and mints a new position with the withdrawn liquidity. Here too, amount0Min and amount1Min are set to zero, exposing users to similar risks.

Recommendation

Consider adding slippage protection by allowing user-defined amount0Min and amount1Min values or by setting reasonable minimum thresholds to reduce the risk of poor execution or MEV exploitation.

Resolution

Impermax Team: Acknowledged.

L-03 | ecrecover Allows Signature Malleability

Category	Severity	Location	Status
Signatures	● Low	ImpermaxERC721.sol: 160	Acknowledged

Description

`ImpermaxERC721.sol` uses vanilla `ecrecover` for signer recovery, which is susceptible to malleability due to signature variations.

This does not cause any immediate damage since the signed permit itself does not change. However, this should still be considered for improvement.

Recommendation

Consider using `OpenZeppelin`'s ECDSA library for signer recovery to mitigate malleability risks.

Resolution

Impermax Team: Acknowledged.

L-04 | Unclaimed Dust After split

Category	Severity	Location	Status
Warning	● Low	TokenizedAeroCLPosition.sol: 167	Acknowledged

Description

`split` removes some liquidity from the current LP position and mints a new one. Due to Aero rounding during minting, small residual (dust) token amounts can remain in the `TokenizedAeroCLPosition` contract.

These residual tokens left behind can be skimmed by anyone, If not reclaimed, the owner of the original position loses these tokens.

Recommendation

Considering calling `skim` to the position owner after `split`.

Resolution

Impermax Team: Acknowledged.

L-05 | Rewards Cannot Be Claimed By EOA

Category	Severity	Location	Status
Unexpected Behavior	● Low	TokenizedAeroCLPosition.sol: 215	Acknowledged

Description

In the `claim` function, `_checkAuthorizedCollateral` obtains owner of the `tokenId` from the Collateral contract:

```
address collateral = _requireOwned(tokenId);
address owner = IERC721(collateral).ownerOf(tokenId);
```

This assumes that every user holding the wrapper NFT will deposit into the Collateral contract. If the wrapper NFT is still in a EOA wallet or separate contract, then the `ownerOf` call will revert. This blocks anyone from calling `claim` when the wrapper NFT is not being used as collateral.

Recommendation

Consider if this is expected behavior and document this risk for users.

Resolution

Impermax Team: Acknowledged.

L-06 | Unused Tokens Not Returned

Category	Severity	Location	Status
Logical Error	Low	TokenizedAeroCLPosition.sol: 190	Acknowledged

Description

The increaseLiquidity function in the TokenizedAeroCLPosition contract allows users to add liquidity to an existing position. The user must first transfer token0 and token1 to the contract, then call increaseLiquidity.

The function withdraws the user's position from the gauge, attempts to increase its liquidity using the transferred amounts, and then redeposits the position.

However, if the provided token amounts are unbalanced, the actual liquidity added may use only part of the transferred tokens.

Any remaining tokens stay in the contract without being returned to the user. These leftover tokens can later be claimed by anyone through the skim function, resulting in a potential loss of funds for the user.

Recommendation

Consider modifying increaseLiquidity to automatically return unused tokens to the user. Alternatively, clearly document this behaviour and ensure users are advised to call skim immediately after increaseLiquidity to recover any remaining tokens.

Resolution

Impermax Team: Acknowledged.

I-01 | Naming Convention For _addGauge

Category	Severity	Location	Status
Informational	● Info	TokenizedAeroCLPosition.sol: 232	Resolved

Description

The function _addGauge is declared external but is named with a leading underscore, a convention usually reserved for private or internal functions.

Recommendation

Rename _addGauge to addGauge to align with standard Solidity conventions.

Resolution

Impermax Team: Resolved.

I-02 | Gas Optimization For nonReentrant Modifier

Category	Severity	Location	Status
Gas Optimization	● Info	TokenizedAeroCLPosition.sol: 262	Acknowledged

Description

The nonReentrant modifier in TokenizedAeroCLPosition.sol currently relies on a bool flag (_notEntered) to prevent reentrancy.

However, this approach incurs high gas costs because each call involves a storage write from zero to nonzero (and vice versa).

Recommendation

Update the nonReentrant modifier to use a uint256 two-state pattern (as recommended by OpenZeppelin), which avoids zero-value writes and reduces gas usage by approximately 10,000–15,000 per call.

```
// storage
uint256 private constant _NOT_ENTERED = 1;
uint256 private constant _ENTERED     = 2;
uint256 private _status;
// in your constructor or initialize:
_status = _NOT_ENTERED;
// modifier
modifier nonReentrant() {
require(_status = _ENTERED, "Impermax: REENTERED");
_status = _ENTERED;
_;
_status = _NOT_ENTERED;
}
```

Resolution

Impermax 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