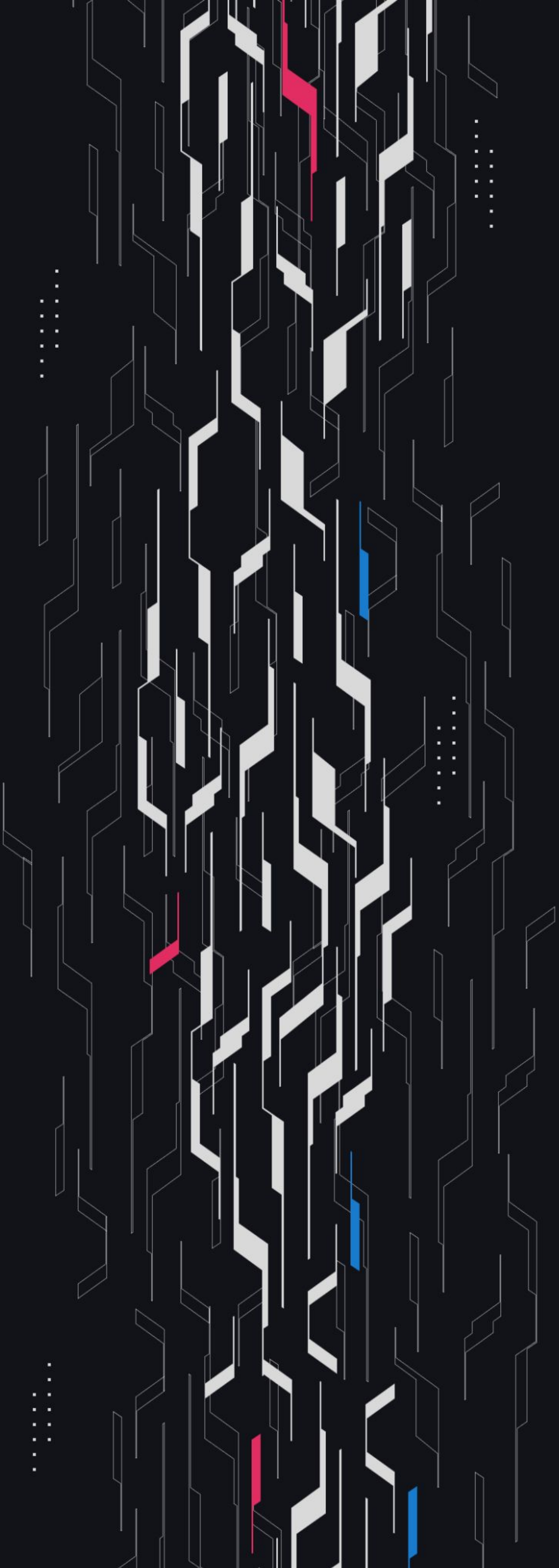
GUARDIAN

Impermax

V3 Tokenized Positions

Security Assessment

February 8th, 2025



Summary

Audit Firm Guardian

Prepared By Daniel Gelfand, Owen Thurm, Cosine,
Wafflemakr, 0xCiphky, Vladimir Zotov

Client Firm Impermax

Final Report Date February 8, 2025

Audit Summary

Impermax engaged Guardian to review the security of the Impermax V3 system allowing for tokenized LP collateral. From the 2nd of January to the 9th of January, a team of 6 auditors reviewed the source code in scope. All findings have been recorded in the following report.

Issues Detected Throughout the engagement 4 High/Critical issues were uncovered and promptly addressed by the Impermax team.

Security Recommendation Given the number of High and Critical issues detected as well as additional findings reported during remediation review, Guardian recommends that an independent security review of the protocol at a finalized frozen commit is conducted before deployment. Furthermore, Guardian recommends the testing infrastructure to be expanded to include all user flows such as the reinvest functionality. The developed fuzzing suite can be used to port over the codebase to Foundry from Truffle.

For a detailed understanding of risk severity, source code vulnerability, and potential attack vectors, refer to the complete audit report below.

🔗 Blockchain network: **Arbitrum**

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

📊 Code coverage & PoC test suite: https://github.com/GuardianAudits/impermax-fuzzing

Table of Contents

Project Information

Smart Contract Risk Assessment

Addendum

Project Overview

Project Summary

Project Name	Impermax
Language	Solidity
Codebase	https://github.com/Impermax-Finance/impermax-v3-core
Commit(s)	Initial commit: 5d3a936284e2814c235ceec261c71ee4cfaae727 Final commit: 4978c2ecdb59fadd91751fa848eb9336511f2f85

Audit Summary

Delivery Date	February 8, 2025
Audit Methodology	Static Analysis, Manual Review, Test Suite, Contract Fuzzing

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
● Critical	1	0	0	0	0	1
● High	3	0	0	0	1	2
● Medium	6	0	0	5	0	1
● Low	22	0	0	16	0	6

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 with Foundry 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 Foundry fuzzing suite.

ID	Description	Tested	Passed	Remediation	Run Count
COLL-01	TokenizedUniswapV3Position.redeem should never revert	✅	✅	✅	10M+
LIQUI-01	After a successful call to restructureBadDebt function, the position should be liquidatable	✅	✅	❌	10M+
LIQUI-02	After a successful reinvest call position should not be liquidatable	✅	❌	✅	10M+
LIQUI-03	After a successful call to restructureBadDebt function, the position should NOT be underwater.	✅	✅	✅	10M+
BORROW-01	After borrow the user's position is never liquidatable	✅	✅	✅	10M+
BORROW-02	After borrow the user's position is never underwater	✅	✅	✅	10M+
BORROW-03	After borrow, the borrowBalance must increase by the borrowAmount	✅	✅	✅	10M+
BORROW-04	When borrowAmount is 0, borrowedBalance should remain unchanged	✅	✅	✅	10M+
GLOB-01	Positions should always have > 0 liquidity	✅	❌	❌	10M+

Findings & Resolutions

ID	Title	Category	Severity	Status
C-01	Collateral-Free Borrows	Reentrancy	● Critical	Resolved
H-01	Auto-Compounding Can Lead To Liquidations	Validation	● High	Partially Resolved
H-02	Oracle Price DoS	DoS	● High	Resolved
H-03	Restructure Debt Locks Up Funds	Logical Error	● High	Resolved
M-01	Arbitrary NFTLP Code Can Be Used	Validation	● Medium	Acknowledged
M-02	Interest Lost On Low Decimal Tokens	Rounding	● Medium	Acknowledged
M-03	Borrow Rate Gaming	Logical Error	● Medium	Acknowledged
M-04	Interest Not Accumulated Before Admin Changes	Logical Error	● Medium	Acknowledged
M-05	ERC-777 Reentrancy In reinvest	Logical Error	● Medium	Resolved
M-06	Frontrun Restructure Bad Debt	Frontrunning	● Medium	Acknowledged
L-01	Liquidators Receive Unexpected Values	Logical Error	● Low	Acknowledged
L-02	Unnecessary Rounding Increment	Logical Error	● Low	Resolved
L-03	Safety Margin Updates Causes Liquidations	Warning	● Low	Acknowledged

Findings & Resolutions

ID	Title	Category	Severity	Status
L-04	Liquidate Can Be Called With Zero Repay	Validation	● Low	Acknowledged
L-05	Functions Naming Convention	Best Practices	● Low	Acknowledged
L-06	Incompatible Types	Warning	● Low	Resolved
L-07	Black Swan Events Lead To Value Theft	Validation	● Low	Acknowledged
L-08	Multiple NFTLPs Owned By ReservesManager	Logical Error	● Low	Acknowledged
L-09	Invalid Price Validations	Validation	● Low	Resolved
L-10	Protocol Revenue Affected By Liquidations	Logical Error	● Low	Acknowledged
L-11	Redundant Function Calls In ImpermaxV3Factory	Best Practices	● Low	Resolved
L-12	Repay Window Created By TWAP	Logical Error	● Low	Acknowledged
L-13	Zero Liquidity Positions Are Permitted	Validation	● Low	Acknowledged
L-14	Missing Reinvestor Protection	Validation	● Low	Acknowledged
L-15	Direct Interactions May Risk User Funds	Frontrunning	● Low	Acknowledged
L-16	Inconsistent Split Percentages	Validation	● Low	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
L-17	NFTLP positions Not Compatible With Router	Logical Error	● Low	Acknowledged
L-18	Unsafe Mint Function Used	Validation	● Low	Resolved
L-19	Split NFT Receiver Is Always The Owner	Logical Error	● Low	Acknowledged
L-20	Lack Of Minimum Borrow Size	Validation	● Low	Acknowledged
L-21	DoS Between Liquidatable & Underwater	DoS	● Low	Acknowledged
L-22	Configuration Leads To Gaming	Configuration	● Low	Acknowledged

C-01 | Collateral-Free Borrows

Category	Severity	Location	Status
Reentrancy	● Critical	ImpermaxV3Borrowable.sol	Resolved

Description

Currently the borrow function is at risk of cross-contract reentrancy. Consider the following scenario:

(1) ImpermaxV3Borrowable.borrow with borrowAmount > 0 with an NFTLP collateral that is enough to cover the borrow.

(2) Callback impermaxV3Borrow is called before tokenId's borrow balance is updated. At this point in time, the account's borrowed is 0. (3) Call ImpermaxV3Collateral.redeem with percentage=1e18 to get back the NFTLP. This is still in the context of the callback.

(4) require(IBorrowable(borrowable0).borrowBalance(tokenId) = 0 and require(IBorrowable(borrowable1).borrowBalance(tokenId) = 0 should pass successfully as the account wasn't updated yet.

(5) Back in ImpermaxV3Borrowable.borrow, canBorrow would be called, but the NFTLP position still has enough liquidity to cover the borrow and the getPositionData function would still return the proper values, even if Alice now holds the NFT instead of the ImpermaxV3Collateral contract.

(6) Ultimately, Alice borrowed a non-zero amount but also holds the entirety of her collateral NFTLP.

Recommendation

Ensure the ImpermaxV3Collateral contract holds the NFTLP collateral when there is an active borrow.

Resolution

Impermax Team: The issue was resolved in commit d71e5b6.

H-01 | Auto-Compounding Can Lead To Liquidations

Category	Severity	Location	Status
Validation	● High	TokenizedUniswapV3Position.sol: 257-305	Partially Resolved

Description PoC

The auto-compounding feature should help borrowers be more capital efficient by automatically reinvesting fees earned on uniswap. As the caller of the `reinvest` function receives a bounty for doing the work, the position's collateral value decreases through auto-compounding.

The `reinvest` function does not implement safety checks for the position's health. Therefore, it is possible that calling the `reinvest` function leads to positions being liquidatable, which is definitely not in the borrower's interest and should therefore be prevented.

It is also possible to auto-compound liquidatable positions and therefore potentially make them go underwater.

Recommendation

Revert at the end of the `reinvest` flow if the position is liquidatable. It would also make sense to let borrowers decide up to which point they want to auto-compound so that reinvestors can not push them to the edge of liquidation.

Resolution

Impermax Team: The issue was resolved in commit af10809.

H-02 | Oracle Price DoS

Category	Severity	Location	Status
DoS	● High	TokenizedUniswapV3Position.sol: 75	Resolved

Description

The function oraclePriceSqrtX96 aims to calculate the oracle price by gathering the TWAP across all Uniswap pools with a specific (token0, token1) pair and calculating the mean.

The pre-condition for a successful call to oraclePriceSqrtX96 is that for each Uniswap pool in the poolsList, the consult function does not revert.

However, if a pool was just created, there would not be an observation that was ORACLE_T seconds ago and Uniswap's observeSingle would revert: "dev Reverts if an observation at or before the desired observation timestamp does not exist"

Ultimately, a user can DoS the calculation of oracle pricing for an entire period just by deploying a different fee tier, which would prevent positionData retrieval and DoS other areas of the codebase such as function isUnderwater and consequently restructureBadDebt.

Recommendation

Consider wrapping the call to Uniswap's observe in a try-catch and then adjusting the number of observations by how many calls were successful.

Resolution

Impermax Team: The issue was resolved in commit 6f6ab04.

H-03 | Restructure Debt Locks Up Funds

Category	Severity	Location	Status
Logical Error	● High	ImpermaxV3Borrowable.sol	Resolved

Description

When a position's debt is restructured, the debt is forgiven, the losses socialized, and the position remains in the system. In certain cases, the position can become liquidatable after restructuring and be liquidated, and other times the position can become fully healthy and remain.

This incongruent behavior may be unexpected for the protocol, since some positions will effectively get a bail-out and remain. This can be used by malicious borrower's to keep lender's funds locked up indefinitely.

Recommendation

Consider if previously underwater positions should remain in the system after restructuring and allow them to be liquidated.

Resolution

Impermax Team: The issue was resolved in commit 92796e2.

Guardian Team: The owner of a position can frontrun the liquidator to call `restructureBadDebt` first so that the position is healthy again and then stuff the block so that the block gas limit is hit before the liquidator's transaction goes through. Afterwards the `blockOfLastRestructureOrLiquidation` check will fail.

M-01 | Arbitrary NFTLP Code Can Be Used

Category	Severity	Location	Status
Validation	● Medium	Global	Acknowledged

Description

The ImpermaxFactory contract allows for the creation of lending pools with a completely arbitrary NFTLP address. Consequently, a malicious user may create a lending pool with a malicious tokenized position which could lead to loss of user data and assets.

Recommendation

Validate that the NFTLP's passed into the ImpermaxFactory functions have been deployed through the respective tokenized position factory contracts.

Resolution

Impermax Team: Acknowledged. Validation is done through the whitelisting process at a higher level (generally in the UI).

M-02 | Interest Lost On Low Decimal Tokens

Category	Severity	Location	Status
Rounding	● Medium	BInterestRateModel.sol: 72	Acknowledged

Description

In the accrueInterest function, the interest calculation interestFactor.mul(_totalBorrows).div(1e18) can round down to zero for low-decimal tokens (such as USDC), particularly when the borrow rate or total borrows are low.

Specifically, when the product of interestFactor and _totalBorrows is less than 1e18, the division results in zero. This leads to a loss of interest for lenders, as the interestAccumulated is zero.

Furthermore, this can cause a discrepancy between totalBorrows and accountBorrows, since accountBorrows does not experience the same precision loss.

Recommendation

Use a higher precision for the rate values to accommodate tokens with fewer decimals, ensuring that the computed interest does not round to zero.

Resolution

Impermax Team: Acknowledged.

M-03 | Borrow Rate Gaming

Category	Severity	Location	Status
Logical Error	● Medium	ImpermaxV3Borrowable.sol: 62, 154	Acknowledged

Description

The `currentBorrowBalance` and `trackBorrow` functions accrue interest and therefore update the `totalBorrows` which would influence the `utilizationRate` and therefore the `borrowRate`. However, they do not have the `_update` modifier and therefore do not update the `borrowRate`.

Therefore the system potentially deals with a stale `borrowRate` and the lenders receive less yield than they should. Furthermore, the system allows for interest gaming, since calling `sync()` every second will provide a different debt accumulated than calling `sync()` once over the same timespans.

For example, with a decreasing `_kinkBorrowRate` every sync the borrow rates will also decrease and cause less to be repaid over the borrow's lifetime, which in turn is less yield for lenders.

Recommendation

Add the `_update` modifier to the `currentBorrowBalance` and `trackBorrow` functions. Furthermore, re-consider if the system show allow for varying sync times to lead to different costs on users.

Resolution

Impermax Team: Acknowledged. We call calculateBorrowRate when there is a supply/demand change in the contract.

M-04 | Interest Not Accumulated Before Admin Changes

Category	Severity	Location	Status
Logical Error	● Medium	BSetter.sol: 47-51	Acknowledged

Description

Interest should always be accrued before updating parameters that will impact the outcome of the next update to fairly calculate the accumulated interest.

The _setAdjustSpeed function in the BSetter contract for example can update the adjustSpeed variable that will influence the borrowRate, but it does not accrue interest before updating this variable.

Therefore the next time the borrowRate is updated it applies the updated adjustSpeed value on the whole timeElapsed since the last update while in reality a part of the timeElapsed should be calculated with the old adjustSpeed value to fairly update the borrowRate.

Recommendation

Accrue interest before updating parameters that will impact the outcome of interest related calculations.

Resolution

Impermax Team: Acknowledged.

M-05 | ERC-777 Reentrancy In reinvest

Category	Severity	Location	Status
Logical Error	● Medium	UniswapV3AC01.sol: 148-149	Resolved

Description

When the bounty is sent to the given bountyTo address with an ERC-777 token the receiver can re-enter the system. At this point, the fee growth values of the position were already reset, but the position had not received the liquidity yet.

Therefore the collateral value of the position is smaller than it is in reality. This could lead to the system seeing the position as liquidatable or underwater when the caller reenters the system.

Therefore a malicious actor can potentially abuse this state to either:

- Liquidate the position to steal from its owner
- Call restructureBadDebt to reduce the debt of the position and steal from the lenders (could be abused by the owner of the position)

Recommendation

Transfer the bounty to the given bountyTo address after all state changes occurred.

Resolution

Impermax Team: The issue was resolved in commit 33126d8.

M-06 | Frontrun Restructure Bad Debt

Category	Severity	Location	Status
Frontrunning	● Medium	ImpermaxV3Borrowable.sol	Acknowledged

Description

When restructureBadDebt is called, the exchangeRate will immediately decrease due to the reduction in _totalBalance without a proportional reduction in pool token supply. A user can frontrun the restructuring to redeem before the exchange rate drop to avoid the penalty.

The LP exploiting this receives yield without risk while increasing the risk of all the other LP as the bad debt is socialized among all other LPs. Furthermore, the LP taking advantage of the stepwise jump in the exchange rate can then mint the same amount of PoolTokens for a fraction of the price.

Recommendation

Consider a 2 step deposit/redeem process.

Resolution

Impermax Team: Acknowledged. I think the only real solution to this would be to introduce an unbonding period for redeem. But this would ruin the UX for a marginal benefit.

L-01 | Liquidators Receive Unexpected Values

Category	Severity	Location	Status
Logical Error	● Low	ImpermaxV3Collateral.sol: 112	Acknowledged

Description

The ImpermaxV3Collateral contract utilizes a 30-minute TWAP to determine the value of collateral and debt during the liquidation process. This approach helps provide a stable price by averaging out short-term market fluctuations.

However, when the liquidated tokens are claimed, Uniswap uses the spot price. This discrepancy between the TWAP and the spot price can lead to differences in the expected versus actual value received by liquidators.

For instance, if the spot price is lower than the TWAP, a liquidator may receive fewer tokens than anticipated, resulting in a possible loss.

Recommendation

Clearly document potential discrepancies between TWAP and spot pricing so users understand the associated risks

Resolution

Impermax Team: Acknowledged. Since liquidations will be handled by external contracts, all the PNL logic and liquidations strategy will be handled by the liquidator.

L-02 | Unnecessary Rounding Increment

Category	Severity	Location	Status
Logical Error	● Low	TokenizedUniswapV3Position.sol: 237	Resolved

Description

In the join function 1 wei is added to the resulting newFeeGrowthInside0LastX128 of the joined position to avoid fee over estimations.

However in cases where the newFeeGrowthInside0LastX128 result does not have precision loss this is an unnecessary addition. Instead rounding up can occur only when precision loss would occur.

Recommendation

Consider implementing tA0.add(tB0).add(newLiquidity - 1).div(newLiquidity) Instead of the existing tA0.add(tB0).div(newLiquidity).add(1) to accurately only round up when precision loss would occur.

Resolution

Impermax Team: The issue was resolved in commit 5d38704.

L-03 | Safety Margin Updates Causes Liquidations

Category	Severity	Location	Status
Warning	● Low	CSetter.sol: 39	Acknowledged

Description

The safetyMarginSqrt plays a crucial role when determining the liquidatable state of a position, as it's used to calculate the LOWEST and HIGHEST price as a factor of the oracle TWAP price.

Therefore, healthy position can become instantly liquidatable if this factor is increased by the admin. The same issue applies when updating the liquidationIncentive and liquidationFee.

Recommendation

Introduce a time lock feature so that these admin state changes include a delay, to allow users to take action before they are applied.

Resolution

Impermax Team: Acknowledged.

L-04 | Liquidate Can Be Called With Zero Repay

Category	Severity	Location	Status
Validation	● Low	Global	Acknowledged

Description

Functions throughout the codebase allow for an input of zero, e.g. functions `ImpermaxV3Borrowable.liquidate` for `repayAmount` and `ImpermaxV3Collateral.redeem` for `percentage`.

This would ultimately leads to events being emitted when no actual actions have been taken on a Position.

Recommendation

Consider validating against zero inputs.

Resolution

Impermax Team: Acknowledged.

L-05 | Functions Naming Convention

Category	Severity	Location	Status
Best Practices	● Low	Global	Acknowledged

Description

There are multiple public/external functions with names starting with an underscore. Some examples:

- `TokenizedUniswapV3Factory._setPendingAdmin`

- `CSetter._initialize`

- `BSetter._setReserveFactor`

However, this naming style goes against the solidity convention for external/public vs internal/private functions, more details here: https://docs.soliditylang.org/en/latest/style-guide.html#underscore-prefix-for-non-external-functions-and-variables

Additionally, this can create confusion for the reader, or allow bugs to be introduced, declaring a function external when it's suppose to be private/internal.

Recommendation

Consider adapting to the function naming convention, according to the solidity docs shared above.

Resolution

Impermax Team: Acknowledged.

L-06 | Incompatible Types

Category	Severity	Location	Status
Warning	● Low	IUniswapV3AC01.sol	Resolved

Description

Within the IUniswapV3AC01 the interface there are numerous functions where the return type does not match the expected return type.

For example, function PROTOCOL_SHARE() external view returns (address); expected an address to be returned in the interface definition but the PROTOCOL_SHARE is a uint.

Recommendation

Correct the interface definitions so the return types match.

Resolution

Impermax Team: The issue was resolved in commit 8a88634.

L-07 | Black Swan Events Lead To Value Theft

Category	Severity	Location	Status
Validation	● Low	ImpermaxV3Borrowable.sol: 105-126	Acknowledged

Description

• The borrow function allows to borrow up to the point of being liquidatable therefore a tiny price change afterward will make the position liquidatable

• The system uses TWAP prices at the moment to calculate the value of collateral and debt

This could be problematic in black swan events when prices drop or rise very quickly. It could lead to a situation where the TWAP price is off by more than the safety margin + liquidation fees compared to the current price.

In this case, anyone would be able to borrow funds that are worth more than the deposited collateral, use these funds to buy more collateral, and repeat the process to drain the protocol.

Recommendation

There are multiple things that can decrease the likelihood of this edge case:

• Add a percentage buffer on top of the liquidation threshold so that users are not able to borrow up to the point of being liquidatable

• Use a different price oracle that would be more accurate in such an edge case

Resolution

Impermax Team: Acknowledged.

L-08 | Multiple NFTLPs Owned By ReservesManager

Category	Severity	Location	Status
Logical Error	● Low	ImpermaxV3Collateral.sol: 138	Acknowledged

Description

During liquidations, the NFTLP token will we split twice, one for the liquidator, one for the reservesManager. Even though each tokenId can be in different ranges, the reservesManager will end up with hundreds of NFTLPs that will later need to be redeemed to collect the underlying.

Recommendation

Consider redeeming the NFTLP to receive the underlying tokens.

Resolution

Impermax Team: Acknowledged.

L-09 | Invalid Price Validations

Category	Severity	Location	Status
Validation	● Low	UniswapV3CollateralMath.sol	Resolved

Description

UniswapV3CollateralMath.newPosition validates that paSqrtX96 > Q32 and pbSqrtX96 < Q160 However, Q32 is below Uniswap's MIN_SQRT_RATIO and Q160 is above Uniswap's MAX_SQRT_RATIO, which leads to potentially invalid position objects being created.

Recommendation

Align the validation with Uniswap's min and max prices.

Resolution

Impermax Team: The issue was resolved in commit 672084e.

L-10 | Protocol Revenue Affected By Liquidations

Category	Severity	Location	Status
Logical Error	● Low	ImpermaxV3Borrowable.sol: 129	Acknowledged

Description

The current implementation of `ImpermaxV3Borrowable.liquidate` allows users to partially repay the current borrowed balance. However, depending on the previous health state, a partial repay can make the position healthy post liquidation.

This means that both the protocol ad the user could have received more fees during a full liquidation.

If the user has both tokens borrowed from the pool, and based on the liquidity structure of the position, there are cases where repaying one of the tokens first will allow the liquidator to trigger a second liquidation with the other token.

Recommendation

Protocol should be aware of the fees they are missing out with partial liquidations. Additionally, document this behavior, so users are aware of the impact of partial liquidations.

Resolution

Impermax Team: Acknowledged. Partial liquidations make us miss out on fees, but they make the protocol safer.

L-11 | Redundant Function Calls In ImpermaxV3Factory

Category	Severity	Location	Status
Best Practices	● Low	ImpermaxV3Factory.sol: 54, 63, 72	Resolved

Description

The createCollateral, createBorrowable0, and createBorrowable1 functions call the _getTokens function in the beginning without using the function's return values. Therefore, the call is redundant and can be removed.

Recommendation

Remove the redundant function calls. If there should be a check to ensure the two token addresses are set add a require statement instead.

Resolution

Impermax Team: The issue was resolved in commit 74c85cf.

L-12 | Repay Window Created By TWAP

Category	Severity	Location	Status
Logical Error	● Low	ImpermaxV3Borrowable.sol: 105	Acknowledged

Description

Due to the lagging nature of TWAP price, borrowers will have a time window to repay their loans, once they notice the price went against them, but the TWAP price has not catch up yet.

Therefore, liquidators will have a hard time finding liquidatable positions, as users will know ahead of time due to the pool price action.

Recommendation

Document this behavior to the users and liquidators.

Resolution

Impermax Team: Acknowledged.

L-13 | Zero Liquidity Positions Are Permitted

Category	Severity	Location	Status
Validation	● Low	TokenizedUniswapV3Position.sol	Acknowledged

Description

There is no zero check in the `mint` function of the `TokenizedUniswapV3Position` contract therefore zero liquidity positions are permitted.

It is also possible to split 0% of a position to create a new position with 0 liquidity. Such unexpected states should not occur and might lead to vulnerabilities with future updates.

Recommendation

Add zero checks to prevent the creation of zero liquidity positions.

Resolution

Impermax Team: Acknowledged. Through the router it’s easier to create a leveraged position by first minting a position with 0 liquidity.

L-14 | Missing Reinvestor Protection

Category	Severity	Location	Status
Validation	● Low	TokenizedUniswapV3Position.sol: 257-305	Acknowledged

Description

If multiple people call reinvest at the same time, the transactions of the users who paid less gas might be successful and the users have to pay the full gas costs, but the transaction does not have any necessary effect and the users will not receive a bounty:

- Bob calls reinvest
- Alice calls reinvest
- Bob paid more gas and the transaction went through first
- The position's fees are reinvested
- Bob receives a bounty for his work
- Alice's transaction went through and in between some fees were accrued on Uniswap
- A dust amount or even no fees are reinvested
- Alice does not receive a bounty or the bounty that does not compensate for her gas costs

This does also allow a griefing vector, a malicious actor could front-run the reinvest call on his position and split the NFTLP into two positions so that the reinvest caller wastes gas to reinvest a dust amount of fees.

Recommendation

Add a lastReinvest slippage check at the beginning of the reinvest function, or document that this function should be called over a contract that implements such a check.

Resolution

Impermax Team: Acknowledged.

L-15 | Direct Interactions May Risk User Funds

Category	Severity	Location	Status
Frontrunning	● Low	Global	Acknowledged

Description

The protocol relies on a two-step process for most of its depositing functionality. Users send funds to the contract or a Uniswap position first, and then call a function that mints the corresponding tokens or positions.

This works well for those who use the periphery contracts, since both actions occur atomically. However, users who interact directly with the protocol's core contracts may risk losing their funds if, for instance, the second transaction is frontrun.

Recommendation

Make it clear in the documentation that this risk exists and advise users to use the protocol's periphery contracts.

Resolution

Impermax Team: Acknowledged. No user should ever interact with the core directly.

L-16 | Inconsistent Split Percentages

Category	Severity	Location	Status
Validation	● Low	Global	Resolved

Description

TokenizedUniswapV2Position prevents a split percentage of 100% but TokenizedUniswapV3Position allows it.

Recommendation

Consider making the validation symmetrical.

Resolution

Impermax Team: The issue was resolved in commit 2b519ec.

L-17 | NFTLP positions Not Compatible With Router

Category	Severity	Location	Status
Logical Error	● Low	ITokenizedUniswapV3Position.sol: 29	Acknowledged

Description

The current implementation of TokenizedUniswapV3Position contains a Position struct that stores position data by tokenId. However, this struct will not be compatible with the periphery contracts, and the unclaimedFees0 and unclaimedFees1 are unexpected returned params.

Recommendation

Be aware of this issue, and update the periphery contracts and avoid integration issues

Resolution

Impermax Team: Acknowledged. We are aware of this, we will update the periphery once the audit for the core is completed.

L-18 | Unsafe Mint Function Used

Category	Severity	Location	Status
Validation	● Low	TokenizedUniswapV3Position.sol: 138	Resolved

Description

The ERC721 `mint` function is used multiple times in the codebase instead of the `safeMint` function. If the receiver is a contract and is not capable of handling NFT-related actions and/or calling other functions in the system, the NFTs will be stuck.

Recommendation

Consider using the `safeMint` function instead of the `mint` function to make sure the receiver is capable of handling NFTs.

Resolution

Impermax Team: The issue was resolved in commit 44b3100.

L-19 | Split NFT Receiver Is Always The Owner

Category	Severity	Location	Status
Logical Error	● Low	TokenizedUniswapV3Position.sol: 198	Acknowledged

Description

When calling the `split` function in the `TokenizedUniswapV3Position` contract, the receiver of the new NFT is always the owner.

The function call could come from an approved actor who wants to receive a part of the NFT which is not possible due to this hardcoded behavior

The approved actor is therefore allowed to receive the whole NFT but not a part of it.

Recommendation

Add a `to` parameter to the `split` function as in the `mint` function.

Resolution

Impermax Team: Acknowledged.

L-20 | Lack Of Minimum Borrow Size

Category	Severity	Location	Status
Validation	● Low	ImpermaxV3Borrowable.sol	Acknowledged

Description

Currently there isn't a minimum on the size of the borrow, which would require a very small collateral amount. In such a case, there may be not enough incentive for a liquidator to liquidate the borrower, and the protocol risks accumulation of bad debt.

Recommendation

Consider enforcing a minimum borrow size.

Resolution

Impermax Team: Acknowledged.

L-21 | DoS Between Liquidatable & Underwater

Category	Severity	Location	Status
DoS	● Low	ImpermaxV3Collateral.sol: 119-120	Acknowledged

Description

A position is not liquidatable if the position is underwater (has acquired bad debt). At this point anyone needs to call restructureDebt to forgive the debt and make the position liquidatable again. This can lead to DoS when prices are around the position's bad debt threshold and fluctuate slightly.

A liquidator could call liquidate and between the creation of the transaction and its execution, the position could go underwater as the prices changed slightly and the liquidate call reverts.

Then the liquidator would call restructureDebt but between the creation of the transaction and its execution the position could be healthy enough again to be liquidated without forgiving debt as the prices changed slightly again causing the restructureDebt call to revert.

This is not a big problem but it could be a bad user experience and delay the liquidation process, a process that should happen as fast as possible to avoid further damage.

Recommendation

Merge the restructureDebt and liquidate flows into one function, so that the restructureDebt flow is executed if the position is underwater before continuing with the liquidate flow.

Resolution

Impermax Team: Acknowledged. Before calling liquidate, the liquidator contract can check if the position is underwater. If it is, it can call restructureDebt before calling liquidate.

L-22 | Configuration Leads To Gaming

Category	Severity	Location	Status
Configuration	● Low	CSetter.sol	Acknowledged

Description

According to the CSetter, the minimum safety margin SAFETY_MARGIN_SQRT_MIN is 1e18.

With this setting, a user can borrow up to about ~1x post-liquidation collateral ratio and then become both liquidatable and underwater simultaneously, introducing bad debt into the collateral very quickly.

This also differs from the whitepaper constraints that the safetyMargin is between 150% and 250%.

Furthermore, in the case that the liquidation fee is zero, a liquidatable position owner can self-liquidate their position to take advantage of the discount instead of repaying their borrow, since self-liquidation would require less collateral for the same amount of assets due to the liquidationIncentive.

Recommendation

Firstly, adjust the minimum sqrt safety margin validation according to the whitepaper.
Secondly, ensure that a liquidation fee is active.

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 Audits

Founded in 2022 by DeFi experts, Guardian Audits is a leading audit firm in the DeFi smart contract space. With every audit report, Guardian Audits 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