Quantstamp Security Assessment Certificate

October 26th 2021 — Quantstamp Verified

APWine AMM 2nd Audit

This audit report was prepared by Quantstamp, the leading blockchain security company.

Executive Summary

Type	AMM, DeFi
Auditors	Christoph Michel, Research Engineer Mohsen Ahmadvand, Senior Research Engineer Ed Zulkoski, Senior Security Engineer
Timeline	2021-09-15 through 2021-10-26
EVM	London
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	Weight rebalancing math SPEC.md
Documentation Quality	Medium
Test Quality	Low

Source Code

Repository	Commit
apwine-amm	43ed707
None	ed2733 (re-audit)

Total Issues	**38** (35 Resolved)
High Risk Issues	**14** (14 Resolved)
Medium Risk Issues	**2** (2 Resolved)
Low Risk Issues	**14** (12 Resolved)
Informational Risk Issues	**3** (2 Resolved)
Undetermined Risk Issues	**5** (5 Resolved)

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or is reasonably likely to lead to moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low-impact in view of the client's business circumstances.
Informational	The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.
Undetermined	The impact of the issue is uncertain.
Unresolved	Acknowledged the existence of the risk, and decided to accept it without engaging in special efforts to control it.
Acknowledged	The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).
Resolved	Adjusted program implementation, requirements or constraints to eliminate the risk.
Mitigated	Implemented actions to minimize the impact or likelihood of the risk.

Summary of Findings

The audit is a continuation of a previous audit of the AMM. Several changes have been done to address the previous findings. The AMM is based on Balancer's AMMs and adds novel features like dynamically changing the weights when new yield is received and providing liquidity while adjusting the pool weights. Overall, we found several new high-severity issues in the second round of audit. The specification has improved, but we believe APWine should further improve the general documentation as well as the inline code documentation. The test coverage should be improved to address the discovered issues in the report and any other potential logical errors. Adding novel features to existing protocols is risky and requires thorough research. We strongly advise conducting comprehensive simulations to ensure that economical assumptions hold true before publicly deploying the pool. In the re-audit phase, we **strictly** reviewed the changes that were **deemed as fix-relevant** for the previously identified issues and **discarded all the other changes.**

ID	Description	Severity	Status
QSP-1	wrong `calcPoolInGivenSingleOut` calculation	High	Fixed
QSP-2	wrong `calcSingleOutGivenPoolIn` calculation	High	Fixed
QSP-3	Exiting pool does not pay out pool assets	High	Fixed
QSP-4	LP total supply excludes opt-out LP tokens	High	Mitigated
QSP-5	`_scaleUnderlyingWeights` decimal issues	High	Mitigated
QSP-6	`getRedeemableExpiredTokens` decimal issues	High	Fixed
QSP-7	Single-sided LP provisioning with weight readjustments is exploitable	High	Fixed
QSP-8	`removeLiquidity` should implement a minimum return check	Medium	Fixed
QSP-9	`MAX_OUT_RATIO` not respected when exiting pool	Medium	Fixed
QSP-10	getEquivalentPoolShares always returns 0	Low	Mitigated
QSP-11	`initialize` functions can be frontrun	Low	Acknowledged
QSP-12	`PoolTokenDeployed` event not fired	Low	Fixed
QSP-13	MasterChef pool rewards can be underfunded	Low	Mitigated
QSP-14	MasterChef ignores `_withUpdate` parameter	Low	Fixed
QSP-15	`calcUpdatedWeights` loss of precision	Low	Mitigated
QSP-16	Can use AMM with tokens that are unrelated to pool	Low	Fixed
QSP-17	Tokens to expire inequality	Low	Mitigated
QSP-18	Dangerous `delegatecall`	Low	Mitigated
QSP-19	Missing `LiquidityAdded` events	Low	Mitigated
QSP-20	Privileged Roles and Ownership	Low	Acknowledged
QSP-21	Track already added LP tokens	Low	Mitigated
QSP-22	Missing parameter validation	Informational	Fixed
QSP-23	Unbounded iteration in `massUpdatePools`	Informational	Acknowledged
QSP-24	Funds mistakenly sent to the AMM cannot be recovered	Informational	Fixed
QSP-25	No exit fee when exiting pool	Undetermined	Fixed
QSP-26	AMM pushes 0 to `periodSwitchBlock`	Undetermined	Fixed
QSP-27	Switchperiod's new underlying balance	Undetermined	Mitigated
QSP-28	`renewTokens` does not set `lastPeriodTokensRenewed`	Undetermined	Mitigated
QSP-29	`removeLiquidity` does not lower LP total supply	High	Fixed
QSP-30	`withdrawExpiredToken` does not lower LP total supply	High	Fixed
QSP-31	[False Positive] `_getUpdatedUnderlyingWeightAndYield` uses inverse spot price	High	Fixed
QSP-32	`lastPeriodRegistered` is only set initially in `finalize`	High	Fixed
QSP-33	`lastPeriodOfLP` is never set	High	Fixed
QSP-34	`_renewFYTPool` does not set new fyToken	High	Fixed
QSP-35	`MasterChef` does not include the `Multiplier` logic	High	Fixed
QSP-36	`WeightsUpdated` event not fired	Low	Fixed
QSP-37	`PairCreated` event emitted twice	Low	Fixed
QSP-38	`_upgradePoolRewardsIfNeeded` is underspecified and not tested	Undetermined	Mitigated

Quantstamp Audit Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Methodology

The Quantstamp auditing process follows a routine series of steps:

1. Code review that includes the following
   i. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
   ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
2. Testing and automated analysis that includes the following:
   i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
   ii. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Toolset

The notes below outline the setup and steps performed in the process of this audit.

Setup

Tool Setup:

- Slither v0.8.1

Steps taken to run the tools:

> Installed the Slither tool: `pip install slither-analyzer` Run Slither from the project directory: `slither .`
> Slither detected that the `poolAmountOut` variable of `getEquivalentPoolShares` is not used which turned out to be an issue where the return value is shadowed. All findings have been adopted in the report.

Findings

QSP-1 wrong `calcPoolInGivenSingleOut` calculation

Severity: *High Risk*

Status: Fixed

File(s) affected: `AMMMaths.sol`

Description: The `calcPoolInGivenSingleOut` function should work like Balancer's function, but instead of computing the `tokenAmountOutBeforeSwapFee` as a quotient, it multiplies the terms. This leads to a wrong return value:

```
// calcPoolInGivenSingleOut
uint256 tokenAmountOutBeforeSwapFee = _tokenAmountOut.mul(UNIT.sub(zar)).div(UNIT);

// in Balancer
uint tokenAmountOutBeforeSwapFee = bdiv(tokenAmountOut, bsub(BONE, zar));
```

Recommendation: Check the correctness of the `calcPoolInGivenSingleOut` function, whether it should be `tokenAmountOutBeforeSwapFee = UNIT.mul(tokenAmountOut).div( UNIT.sub(zar) )` instead.

Update: The issue still exists in the code.

QSP-2 wrong `calcSingleOutGivenPoolIn` calculation

Severity: *High Risk*

Status: Fixed

File(s) affected: `AMMMaths.sol`

Description: The `calcSingleOutGivenPoolIn` function should work like Balancer's function, but the `tokenOutRatio` is computed with an exponent with the wrong amount of precision.

```
// calcSingleOutGivenPoolIn
uint256 tokenOutRatio = pow(poolRatio, UNIT.div(normalizedWeight));

// in Balancer
uint tokenOutRatio = bpow(poolRatio, bdiv(BONE, normalizedWeight));
```

Recommendation: Check the correctness of the `calcSingleOutGivenPoolIn` function, whether `tokenOutRatio` should be `pow(poolRatio, UNIT.mul(UNIT).div(normalizedWeight))` instead.

Update: The issue still exists in the code.

QSP-3 Exiting pool does not pay out pool assets

Severity: *High Risk*

Status: Fixed

File(s) affected: `AMM.sol`

Description: The `exitswapPoolAmountIn` and `exitswapExternAmountOut` functions burn LP tokens and try to transfer tokens **from the user** instead of paying out the user's fair share of the pool.

```
// exitswapPoolAmountIn, should transfer to msg.sender instead
IERC20(_tokenOut).safeTransferFrom(msg.sender, address(this), tokenAmountOut);

// exitswapExternAmountOut, should transfer to msg.sender instead
IERC20(_tokenOut).safeTransferFrom(msg.sender, address(this), _tokenAmountOut);
```

Recommendation: Transfer the tokens to the user instead of trying to pull them from the user.

QSP-4 LP total supply excludes opt-out LP tokens

Severity: *High Risk*

Status: Mitigated

File(s) affected: `AMM.sol`

Description: The `totalSupply()` function adjusts the total supply of LP tokens dynamically, it is decreased by the outstanding opted-out LP tokens. This adjusted total supply is then used to mint and remove liquidity which leads to issues:

Exploit Scenario: Assume two users provide equal liquidity in the beginning, one opts-out of renewals, the other opts-in.

- User A: `joinPool(50.0 LP tokens, optOutRatio=0%)`
- User B: `joinPool(50.0 LP tokens, optOutRatio=100%) => _optOutAutoRenewal(50.0) => poolTokenToOut[currentPeriodIndex] += amount = 50.0`
- The total supply is now `totalSupply() = totalSupply = 100.0`
- `switchPeriod` is called and `poolTokenOut = poolTokenToOut[currentPeriodIndex] = 50.0` is set, changing `totalSupply() = totalSupply - poolTokenOut = 100.0 - 50.0 = 50.0`
- User A: `removeLiquidity(50.0)`, receives entire pool amount as the ratio of `_poolAmountIn / poolTotal = 1.0` due to `poolTotal = totalSupply() = 50.0`

User B's LP tokens are now valueless as the pool is empty.

Recommendation: Document how the entire opt-out / opt-in mechanism works, and the synergy of `poolTokenToOut`, `poolTokenToOut`, `totalSupply`, `balanceOf`, `_beforeTokenTransfer`, `_withdrawExpiredToken`, etc.

Update: No longer applicable after the changes.

QSP-5 `_scaleUnderlyingWeights` decimal issues

Severity: *High Risk*

Status: Mitigated

File(s) affected: `AMM.sol`

Description: The `_scaleUnderlyingWeights` function adds `newYieldRecorded` to the `_newSpot` value. The `_newSpot`'s *precision* depends on where the function is called from:

- when called from `switchPeriod` it is in underlying decimals: `10**underlyingOfIBT.decimals()`
- when called from `_scaleUnderlyingWeightsToGeneratedYield` it's in spot price decimals which have a precision of 18.

When the underlying decimals are different from 18, like is the case for USDC and USDT (which have 6 decimals), the function does not seem to work correctly as `_newSpot` does not always have the same precision as `newYieldRecorded`.

Recommendation: Make sure the precision is always correct for all calls to this function. Add tests for underlying tokens with different decimals.

Update: No longer applicable after the changes.

QSP-6 `getRedeemableExpiredTokens` decimal issues

Severity: *High Risk*

Status: Fixed

Description: The `getRedeemableExpiredTokens` function keeps multiplying the `redeemable` variable with the generated yield (which is an 18-decimal percentage value) but does not "normalize" it again by dividing by 1e18. The `redeemable` variable grows in orders of magnitude each time, which allows an attacker to get paid out multiples of their actual redeemable value.

Recommendation: Fix the multiplications in `getRedeemableExpiredTokens` and add tests for this function and the withdrawals.

Update: The issue still exists in the code.

QSP-7 Single-sided LP provisioning with weight readjustments is exploitable

Severity: *High Risk*

Status: Fixed

File(s) affected: `AMM.sol`

Description: A new `joinUpdatePoolAmountOut` function was added on top of Balancer that allows providing liquidity with a single token by rebalancing the pool weights only. The LP tokens minted only depend on the ratio of the deposited amount to the pool balance of this single asset.

- An attacker can first manipulate the pool draining a single asset of the pool (`_tokenIn`) by buying large amounts of it (for example with flashloans).
- They then only need to provide a small amount of `_tokenIn` tokens for `joinUpdatePoolAmountOut` but will mint a large amount of LP tokens based on the tiny `IERC20(_tokenIn).balanceOf(address(this))` balance.
- The `joinUpdatePoolAmountOut` might now drastically adjust the weights, but it doesn't matter as redeeming the LP tokens using `removeLiquidity` ignores weights. The attacker now receives a share of **all** three tokens in the pool.

Exploit Scenario: The attack should be profitable, assume we start with three token balances in the pool of `A`, `B`, `C` with equal initial weights of 33% (ignoring fees and `MAX_IN_RATIO` for simplicity).

- Trading `1.0 A` for roughly `0.5 B` in the pool using `swapExactAmountIn` would change the pool to `(2A, 0.5B, C)`.
- Joining the pool now using the single-sided liquidity provisioning function `joinUpdatePoolAmountOut(_tokenIn=B ,_poolAmountOut=poolTotal)` using `B` tokens such that we mint 100% new LP tokens, requires joining with `0.5B /0.33 = 1.5B`. Attacker mints 100% new LPs, pool balances are now `(2A, 2B, C)`. (Token weights change after `joinUpdatePoolAmountOut`.) The attacker's LP token share is now at 50% *of the total supply*.
- Redeeming 50% of the total supply using `removeLiquidity` leaves the pool at `(A, B, 0.5C)`.
- The attacker makes a profit of `0.5C` tokens.

Similar issues could happen with other weight rebalancing functions, like when sandwich attacking `switchPeriod`.

Recommendation: The core issue seems to be that Balancer's weights are static and therefore `removeLiquidity` can ignore the weights and redeem LP tokens based on a fair share of the total supply for all pool tokens. Perform further simulations to ensure that any new features added to Balancer (such as dynamic weight adjustments) don't lead to issues.

QSP-8 `removeLiquidity` should implement a minimum return check

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `AMM.sol`

Description: The `removeLiquidity` function checks whether the returned amounts of the three assets are *less or equal* than a *maximum* slippage amount. However, as this function is redeeming LP tokens for the pool assets, it should check if the received amounts are *greater* than a *minimum* slippage amount.
Currently, users could trade at a very bad execution price, far lower than their initial expected trade price, and the checks do not protect against this.

Recommendation: Use minimum amounts instead of maximum amounts as parameters and check against these.

QSP-9 `MAX_OUT_RATIO` not respected when exiting pool

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `AMM.sol`

Description: The `exitswapPoolAmountIn` and `exitswapExternAmountOut` functions are supposed to check that the `_tokenAmountOut` is below a certain `MAX_OUT_RATIO` percentage of the asset's entire pool balance. The implemented check is missing a division by `1e18` for it to work correctly. Currently, the allowed token amount is inflated by `10^18`.

```
// in exitswapPoolAmountIn and exitswapExternAmountOut
// should .div(UNIT) as MAX_OUT_RATIO is a percentage value where 1.0 = 1e18
require(_tokenAmountOut <= outTokenBalance.mul(MAX_OUT_RATIO), "ERR_MAX_OUT_RATIO");
```

Recommendation: Fix the `MAX_OUT_RATIO` check by dividing by `10**18`.

QSP-10 getEquivalentPoolShares always returns 0

Severity: *Low Risk*

Status: Mitigated

File(s) affected: `AMM.sol`

Description: The `AMM.getEquivalentPoolShares` function redeclares the `poolAmountOut` variable but does not return it. Therefore, the actual `poolAmountOut` return value is shadowed and never set to any other value than 0.
This function does not seem to be used by the contracts themselves but third-party contracts that rely on it will not work correctly.

Recommendation: Do not redeclare the variable, simply return the value.

Update: The issue is no longer applicable after the changes.

QSP-11 `initialize` functions can be frontrun

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `AMM.sol`, `MasterChef.sol`, `AMMRegistry.sol`

Description: The `initialize` function that initializes important contract state can be called by anyone. The attacker can initialize the contract before the legitimate deployer, hoping that the victim continues to use the same contract. In the best case for the victim, they notice it and have to redeploy their contract costing gas.

Recommendation: Use the constructor to initialize non-proxied contracts. For initializing proxy contracts deploy contracts using a factory contract that immediately calls `initialize` after deployment or make sure to call it immediately after deployment and verify the transaction succeeded.

Update: The response from the team:

> We acknowledge the issue and will implement a factory on a later stage. For the moment we will carefully review all our contract initialization

QSP-12 `PoolTokenDeployed` event not fired

Severity: *Low Risk*

Status: Fixed

Description: The `PoolTokenDeployed` event in `AMM` is not used. Unused code can hint at programming or architectural errors.

Recommendation: Use it or remove it if the backend does not require it.

Update: The issue is no longer applicable after the changes.

QSP-13 MasterChef pool rewards can be underfunded

Severity: *Low Risk*

Status: Mitigated

File(s) affected: `MasterChef.sol`

Description: Enough `APW` reward tokens need to be present in the MasterChef contract, otherwise payouts fail. As it is called before each `withdraw`, users might not be able to claim their already allocated and pending rewards.
The `updatePool` function transfers APW to itself when a mint would be expected. It's unclear why this should behave like a mint.

```
// updatePool
// should be mint?
apw.safeTransfer(address(this), apwReward);
```

Recommendation: Ensure there is always enough APW in the contract and clarify the intention of the self-transfer.

Update: The issue is no longer applicable after the changes.

QSP-14 MasterChef ignores `_withUpdate` parameter

Severity: *Low Risk*

Status: Fixed

Description: The `MasterChef.set` function ignores the `_withUpdate` parameter and always calls `massUpdatePools`.

Recommendation: Only call `massUpdatePools` when `_withUpdate` is true.

QSP-15 `calcUpdatedWeights` loss of precision

Severity: *Low Risk*

Status: Mitigated

File(s) affected: `AMMMaths.sol`

Description: When computing the updated weights in `calcUpdatedWeights` function first divides and then multiplies again. This leads to a loss of precision of the final result as the result of the integer division is truncated.

```
// inner cFYT.mul(Ra).div(UNIT) does an early division
uint256 newAPWIBTWeight = newFYTWeight.mul(cFYT.mul(rAPWIBT).div(UNIT)).div(UNIT);
// inner cFYT.mul(Ru).div(UNIT) does an early division
uint256 newUnderlyingWeight = newFYTWeight.mul(cFYT.mul(rUNDERLYING).div(UNIT)).div(UNIT);
```

Recommendation: Multiply first before diving, for example, `newAPWIBTWeight = newFYTWeight.mul(cFYT).mul(rAPWIBT).div(UNIT * UNIT)`

Update: The issue is no longer applicable after the changes.

QSP-16 Can use AMM with tokens that are unrelated to pool

Severity: *Low Risk*

Status: Fixed

File(s) affected: `AMM.sol`

Description: Some functions to swap or add/remove liquidity can be used with tokens unrelated to the pool as these functions take in a `_tokenIn` address parameter that is not further validated. For example, `joinswapExternAmountIn` or `joinswapPoolAmountOut` can be called with arbitrary tokens and they don't validate the parameters. No direct exploit was found as these tokens have a token weight of zero and fail either by a "division-by-zero" error or return zero pool tokens to mint. We still recommend validating token parameters to avoid any unforseen issues.

Recommendation: Validate the parameters, for example, by checking if `getTokenWeight(_tokenIn) > 0`.

Update: The issue still exists in the code.

QSP-17 Tokens to expire inequality

Severity: *Low Risk*

Status: Mitigated

File(s) affected: `AMM.sol`

Description: The `_optOutAutoRenewal` function does not allow fully opting out of the renewal as the `require(balanceOf(_user).sub(poolTokensRegistered) > _amount, "ERR_AMOUNT")` check performs a strict inequality check.

Recommendation: Allow to opt-out with the entire LP amount by using `>=` instead.

Update: The issue is no longer applicable after the changes.

QSP-18 Dangerous `delegatecall`

Severity: *Low Risk*

Status: Mitigated

File(s) affected: `AMM.sol`

Description: The `renewTokens` function performs a `delegatecall` on the `futureVault.getControllerAddress()` contract. Only admins can set this contract's address, but if the key is compromised, the attacker can change the controller contract to a malicious one and call `renewTokens` to `delegatecall` into it to steal the AMM funds by transfering them out.

Recommendation: Check if it's possible to avoid this `delegatecall` to reduce the attack surface on a compromised controller.

Update: The issue is no longer applicable after the changes.

QSP-19 Missing `LiquidityAdded` events

Severity: *Low Risk*

Status: Mitigated

File(s) affected: `AMM.sol`

Description: Functions like `joinMultipleUpdatePoolAmountIn` do not emit the `LiquidityAdded` event while similar functions like `joinMultipleUpdatePoolAmountIn` do.

Recommendation: Emit the event whenever liquidity is added

Update: The issue is no longer applicable after the changes.

QSP-20 Privileged Roles and Ownership

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `MasterChef.sol`

Description: Smart contracts will often have `owner` variables to designate the person with special privileges to make modifications to the smart contract.

- `MasterChef.withdrawAPW` can withdraw APW at any time
- `AMM.renewTokens` performs a `delegatecall` to an admin-controlled contract and can steal pool funds at any time

Recommendation: This centralization of power needs to be made clear to the users, especially depending on the level of privilege the contract allows to the owner.

Update: The response from the team:

> We acknowledge the extent of the privilege of the owner role. This will be clear in the user documentation. The role will be given to a multisig of chosen guardians, and passed to a more decentralized system in a later stage

QSP-21 Track already added LP tokens

Severity: *Low Risk*

Status: Mitigated

File(s) affected: `MasterChef.sol`

Description: In `add`, the comment says "DO NOT add the same LP token more than once. Rewards will be messed up if you do.", it's easy to avoid this error by keeping a map of already added LP tokens.

Recommendation: Keep a map of already added LP tokens and throw if trying to add the same LP token.

Update: The issue is no longer applicable after the changes.

QSP-22 Missing parameter validation

Severity: *Informational*

Status: Fixed

Description: Some parameters of functions are not checked for invalid values:

- `AMM.sol:constructor`: should check against zero
- `AMM.sol:99~101`: `_underlyingOfIBTAddress`, `_futureVault`, `_admin` should be checked against zero

- `MasterChef.sol:77~80`: `_apw`, `_apwPerBlock`, `_startBlock`, `_bonusEndBlock` should be checked against zero
- `MasterChef.sol:90~91`: `_allocPoint`, `_lpToken` should be checked against zero
- `MasterChef.sol:116`: `_to`,`_from` should be checked against zero
- `MasterChef.sol:167`: `_amount` should be greater than zero
- `MasterChef.sol:228`: `_recipient, _amount` should be greater than zero

Wrong user input or wallets defaulting to the zero addresses for a missing input can lead to the contract needing to redeploy or wasted gas.

Recommendation: Validate the parameters

Update: `AMM.sol:constructor` is removed, but others are not fixed.

QSP-23 Unbounded iteration in `massUpdatePools`

Severity: Informational

Status: Acknowledged

File(s) affected: `MasterChef.sol`

Description: In `massUpdatePools`, the `poolInfo.length` can be arbitrarily large. The transactions can fail if the arrays get too big and the transaction would consume more gas than the block limit. This will then result in a denial of service for the desired functionality and break core functionality.

Recommendation: Keep the number of pools small.

Update: The response from the team:

> We acknowledge the potential issue if the number of pools registered per contract becomes too high. As recommended, we will keep its number small.

QSP-24 Funds mistakenly sent to the AMM cannot be recovered

Severity: Informational

Status: Fixed

Description: Much of the calculations in the `AMM` contract use `balanceOf(this)` to count assets held by the contract. However, since funds may be sent to the contract by anyone for any reason (i.e., without calling a deposit function) and they are directly counted as the pool's reserve balances, these tokens cannot be recovered.

Recommendation: Consider adding a `rescueTokens` function to recover tokens unrelated to the three pool assets while ensuring correctness when changing future yield tokens.

QSP-25 No exit fee when exiting pool

Severity: Undetermined

Status: Fixed

File(s) affected: `AMM.sol`

Description: The `exitswapPoolAmountIn` and `exitswapExternAmountOut` functions don't charge an exit fee unlike Balancer.

Recommendation: Clarify if this is the intended behavior.

Update: The issue still exists in the code.

QSP-26 AMM pushes 0 to `periodSwitchBlock`

Severity: Undetermined

Status: Fixed

File(s) affected: `AMM.sol`

Description: It's unclear why the AMM pushes `0` to the `periodSwitchBlock` in `initialize`.

```
periodSwitchBlock.push();
```

Recommendation: The `periodSwitchBlock` value is never read in the contract, clarify what it is used for and how it's initialized.

QSP-27 Switchperiod's new underlying balance

Severity: Undetermined

Status: Mitigated

Description: It's unclear to the auditors why the `newVirtualUnderlyingBalance` is chosen as:

```
uint256 newVirtualUnderlyingBalance = oldUnderlyingOfIBTBalance.mul(currentSum).div(yieldPerUnderlying);
```

It is divided by `yieldPerUnderlying` which could be ~10% as the yield percentage per period. Should it be divided by `UNIT + yieldPerUnderlying`?

Recommendation: Explain how `newVirtualUnderlyingBalance` is chosen.

Update: The issue is no longer applicable after the changes.

QSP-28 `renewTokens` does not set `lastPeriodTokensRenewed`

Severity: Undetermined

Status: Mitigated

Description: The `renewTokens(user)` function does not set `lastPeriodTokensRenewed[user]`. It seems to be possible to repeatedly call `renewTokens` and claim the `IBT` and `LP` tokens several times.

Recommendation: Check that `renewTokens` works as intended.

Update: The issue is no longer applicable after the changes.

QSP-29 `removeLiquidity` does not lower LP total supply

Severity: *High Risk*

Status: Fixed

File(s) affected: `AMM.sol`

Description: The `removeLiquidity` function burns the tokens but does not decrease the internal tracking of LP total supply in `totalLPSupply[_pairID][lastPeriodRegistered]`.

Recommendation: Correctly decrease the total LP supply when tokens are burned. We recommend calling `_exitPool` instead which burns the tokens and correctly decreases the total supply.

QSP-30 `withdrawExpiredToken` does not lower LP total supply

Severity: *High Risk*

Status: Fixed

File(s) affected: `AMM.sol`

Description: The `withdrawExpiredToken` function burns the tokens but does not decrease the internal tracking of LP total supply in `totalLPSupply[_pairID][lastPeriodId]`. Note that this is still used to compute the `userUnderlyingAmount` for other users and must therefore be reduced.

Recommendation: Correctly decrease the total LP supply when tokens are burned.

QSP-31 [False Positive] `_getUpdatedUnderlyingWeightAndYield` uses inverse spot price

Severity: *High Risk*

Status: Fixed

Description: The `_getUpdatedUnderlyingWeightAndYield` function computes the underlying weight by dividing the underlying balance by the underlying balance plus the principal token balance times the new spot price:

```
uint256 newUnderlyingWeight = balances[1].mul(UNIT).div(
 balances[1].add(balances[0].mul(newSpotPrice).div(UNIT))
);
```

We believe the spot price should therefore be in underlying per principal token (`U / PT`). The inverseSpotPrice is in`U / PT` as `calcSpotPrice` divides the underlying by the `PT`, the `newSpotPrice` divides by `inverseSpotPrice` and is therefore in `PT / U`.

Recommendation: Ensure that the spot prices are correctly computed and add tests to simulate the spot prices without trades and compare them to the provided paper simulation. The underlying weight should decrease over time

Update: After close investigation we decided to mark this issue as false-positive.

QSP-32 `lastPeriodRegistered` is only set initially in `finalize`

Severity: *High Risk*

Status: Fixed

File(s) affected: `AMM.sol`

Description: The `lastPeriodRegistered` state variable should reflect the current period index of the future vault. It's used throughout the code to access the current total LP supply and other fields. Currently, the contract is minting/redeeming the same LP tokens for all periods which leads to issues.

Recommendation: Update `lastPeriodRegistered` to the new period at an appropriate time (in `createLiquidity`?).

QSP-33 `lastPeriodOfLP` is never set

Severity: *High Risk*

Status: Fixed

File(s) affected: `AMM.sol`

Description: The `lastPeriodOfLP` state variable is never set but assumed to be set throughout the code when withdrawing expired tokens, as well as in `getRedeemableExpiredTokens`.

Recommendation: Clarify how the code is supposed to work. Add more tests for these features.

QSP-34 `_renewFYTPool` does not set new fyToken

Severity: *High Risk*

Status: Fixed

File(s) affected: `AMM.sol`

Description: The `_renewFYTPool` function sets `pairs[1].tokenAddress = fytAddress;` where `fytAddress = futureVault.getFYTofPeriod(currentPeriodIndex);`. However, at this point, `currentPeriodIndex` is not yet set to the new period in `switchPeriod`. When advancing the period through `switchPeriod`, the wrong tokens are therefore set up for the pool.

Recommendation: Set the correct token for the pair when calling `switchPeriod`.

QSP-35 `MasterChef` does not include the `Multiplier` logic

Severity: *High Risk*

Status: Fixed

File(s) affected: `MasterChef.sol`

Description: `pendingAPW` and `updatePool` do not multiply the reward amount by the total amount of blocks since the last reward (`block.number - pool.lastRewardBlock`). This was handled as part of `getMultiplier` in the original Sushiswap implementation.

Recommendation: This needs to be added to the specification. The code should include comments as to why this decision was made.

Update: The multiplier logic was added back to the contract.

QSP-36 `WeightsUpdated` event not fired

Severity: *Low Risk*

Status: Fixed

File(s) affected: `AMM.sol`

Description: The `WeightsUpdated` event in AMM is not used. Unused code can hint at programming or architectural errors.

Recommendation: It should be emitted in `_updateWeightsFromYieldAtBlock`.

QSP-37 `PairCreated` event emitted twice

Severity: *Low Risk*

Status: Fixed

File(s) affected: `AMM.sol`

Description: The `PairCreated` event in `AMM` is emitted twice in `finalize` for the *same* pair. It is emitted once in the `_createPair` call and then another time in the function body with identical arguments.

Recommendation: Remove the second event and only use the one in `_createPair`.

QSP-38 `_upgradePoolRewardsIfNeeded` is underspecified and not tested

Severity: *Undetermined*

Status: Mitigated

File(s) affected: `MasterChef.sol`

Description: The `_upgradePoolRewardsIfNeeded` function is added to MasterChef. When `_upgradePoolRewardsIfNeeded` is invoked during `deposit`/`withdraw`, this sets the reward amount for the old period to zero. How does this affect users who deposited tokens for the previous period? Do they get zero rewards, or are their tokens somehow upgraded to the next period? If users of the previous period simply receive zero reward, this may introduce a transaction-ordering dependence issue, such that users that deposit old-period tokens must claim their rewards before the next period starts.

Recommendation: Add documentation explaining when such out of date periods could happen and what does that have to do with AMM. Include adequate tests to ensure the correctness of the logic.

Update: Migration logic is implemented to move pending rewards to current periods.

Automated Analyses

Slither

[UPDATE] Slither fails to analyze the latest commit with an exception.

Code Documentation

More comments explaining the intention of the code should be added.

Adherence to Best Practices

- `AMM.amm_paused` is in snake_case
- `AMM.joinMultipleUpdatePoolAmountIn` should use `getEquivalentPoolShares` to compute the `poolAmountOut`
- `AMM.sol:131`: initalized variable's value is checked before being assigned. Relying on compiler's default value is error-prone and can cause confusion. We recommend that you set the variable in the initialize method or right where you declare it.
- `MasterChef.sol`: Use of magic number throughout the contract like `1e12`. Consider defining constants with these values with meaningful names and using these throughout the contract
- `MasterChef.getMultiplier` could explicitly check that `from <= to`
- `MasterChef` deposit and withdraw functions do not follow the Checks-Effects-Interactions pattern

Test Results

Test Suite Results

One test case is failing (see below).
Consider adding test cases for all the logical issues listed in this report.

```
APWine AMM
  With future deployed [1]
    With funds deposited in future [2]
       ✓ Can get the current period index for the pool
       ✓ Can add initial liquidity (207ms)
       ✓ Cannot swap exact amount in yet (49ms)
       ✓ Cannot finalize it as the state is inactive
       ✓ Cannot swap exact amount out yet
       ✓ Cannot switch period yet
       ✓ Cannot add liquidity yet
       ✓ Cannot remove liquidity yet
       ✓ Cannot join pool yet
       ✓ Cannot exit pool yet
      With initial liquidity and another user in future [3]
         ✓ Can only pause the AMM as admin
         ✓ Can rescue funds mistakenly sent to the contract for token-underlying
         ✓ Can rescue funds mistakenly sent to the contract for token-ibt (40ms)
         ✓ Can only rescue funds as admin
         ✓ Cannot rescue funds if not necessary (42ms)
         ✓ Can get the last period of the future for which a user deposited liquidity on the amm
        1) Last period of LP is being updated correctly when joining an ongoing period
         ✓ Computes the correct spot price for PT/UNDERLYING and UNDERLYING/PT
         ✓ Computes the correct spot price for PT/FYT
         ✓ Anyone can add liquidity to PT/UNDERLYING pool (73ms)
         ✓ Anyone can add liquidity to PT/FYT pool (86ms)
         ✓ Cannot receive more LP tokens than liquidity provided (85ms)
         ✓ Cannot add liquidity for 0 LP Tokens
         ✓ Can remove liquidity (82ms)
         ✓ Cannot add liquidity if amount in is greater than max amount provided
         ✓ Transfers the right balances when removing liquidity from PT/FYT pool (97ms)
         ✓ Transfers the right balances when removing liquidity from PT/UNDERLYING pool (89ms)
         ✓ Cannot remove liquidity from PT/UNDERLYING pool by providing 0 pool tokens
         ✓ Cannot remove liquidity from PT/FYT pool by providing 0 pool tokens
         ✓ Can pull PT from the PT/UNDERLYING pool by specifying pool amount in (123ms)
         ✓ Can pull UNDERLYING from the PT/UNDERLYING pool by specifying pool amount in (102ms)
         ✓ exitSwapPoolAmountIn: Cannot provide tokenId greater or equal to 2
         ✓ Can pull PT from the PT/FYT pool by specifying pool amount in (125ms)
         ✓ Transfers the right balances when removing liquidity from PT/UNDERLYING pool (86ms)
         ✓ Can pull FYT from the PT/FYT pool by specifying pool amount in (111ms)
         ✓ Reverts if token out is less than min value provided (220ms)
         ✓ Can pull PT from the PT/UNDERLYING pool by specifying token amount out (149ms)
         ✓ exitSwapPoolAmountIn: Cannot provide tokenId greater or equal to 2
         ✓ Can pull UNDERLYING from the PT/UNDERLYING pool by specifying token amount out (137ms)
         ✓ Can pull PT from the PT/FYT pool by specifying token amount out (144ms)
         ✓ Can pull FYT from the PT/FYT pool by specifying token amount out (138ms)
         ✓ Reverts if MAX out Ratio is reached (150ms)
         ✓ Reverts if token in is more than max value provided (302ms)
         ✓ Can join pool by providing apwibt to PT/UNDERLYING pool (174ms)
         ✓ exitSwapExternAmountOut: Cannot provide tokenId greater or equal to 2
         ✓ Cannot provide more token than max in ratio to PT/UNDERLYING pool (43ms)
         ✓ Cannot join pool by expecting more pool tokens and providing less awpibt than needed to PT/UNDERLYING pool (81ms)
         ✓ Can join pool by providing apwibt to PT/FYT pool (186ms)
         ✓ Cannot provide more token than max in ratio to PT/FYT pool
         ✓ Cannot join pool by expecting more pool tokens and providing less awpibt than needed to PT/FYT pool (66ms)
         ✓ Can join pool by providing apwibt for PT/UNDERLYING pool (71ms)
         ✓ Cannot provide more token than max in ratio for PT/UNDERLYING pool
         ✓ Cannot provide more token than max in ratio for PT/UNDERLYING pool (70ms)
         ✓ Cannot join pool by if awpibt needed is more than given max amount for PT/UNDERLYING pool (41ms)
         ✓ Can join pool by providing apwibt for PT/FYT pool (67ms)
         ✓ Cannot provide more token than max in ratio for PT/FYT pool (45ms)
         ✓ joinSwapExternAmountIn: Cannot provide tokenId greater or equal to 2
         ✓ Cannot join pool by expecting more pool tokens and providing less awpibt than needed to PT/FYT pool (66ms)
         ✓ Cannot join pool by if awpibt needed is more than given max amount for PT/FYT pool (42ms)
         ✓ Can execute swapExactAmountIn between APWIBT and UNDERLYING (66ms)
         ✓ Can execute swapExactAmountIn in between APWIBT and FYT (70ms)
         ✓ Can execute swapExactAmountOut between APWIBT and UNDERLYING (67ms)
         ✓ Can execute swapExactAmountOut in between APWIBT and FYT (67ms)
         ✓ Pair 0 : Correctly perform swapExactAmountIn (96ms)
         ✓ Pair 1 : Correctly perform swapExactAmountIn (101ms)
         ✓ SwapExactAmountIn reverts if minAmountOut not reached (76ms)
         ✓ Pair 0 : Correctly perform swapExactAmountOut (93ms)
         ✓ Pair 1 : Correctly perform swapExactAmountOut (100ms)
         ✓ SwapExactAmountOut reverts if maxAmountIn reached (68ms)
         ✓ sets swapping fee
         ✓ reverts if swapping fee is greater than 1
         ✓ returns the amm state
         ✓ returns the future address
         ✓ returns the address of PT
         ✓ returns the address of underlying of IBT
         ✓ returns the address of IBT
         ✓ returns the address of FYT
         ✓ returns the pair ID for given token
         ✓ returns address of the pool tokens contract
         ✓ returns pairs by pair ID
         ✓ return the PT weight In pair
      With AMM paused [4]
         ✓ Can only resume the AMM as admin
         ✓ Cannot swap exact amount in
         ✓ Cannot swap exact amount out
         ✓ Cannot switch period
         ✓ Cannot create liquidity
         ✓ Cannot add liquidity
         ✓ Cannot remove liquidity
         ✓ Cannot join pool
         ✓ Cannot exit pool
      With 10% yield generated [5]
        With new period [6]
           ✓ The period index for the pool is being updated after switch period
           ✓ pair is renewed for Underlying and Fyt pool
           ✓ Withdraws expired tokens (140ms)
      With swapping fees set [7]
         ✓ Correctly performs swapExactAmountIn (105ms)
         ✓ Correctly perform swapExactAmountOut (89ms)
         ✓ Correctly perform joinSwapExternAmountIn (174ms)
         ✓ Correctly perform joinSwapPoolAmountOut (88ms)
         ✓ Correctly perform exitSwapExternAmountOut (136ms)
         ✓ Correctly perform calcSingleOutGivenPoolIn (109ms)
      Using AMM Router [8]
         ✓ SwapExactAmountIn works for PT -> FYT path (212ms)
    With initial liquidity at specific spot price and another user in future [9]
       ✓ Liquidity is already Initialized
       ✓ Liquidity can't be initilized with tokenAmount 0
       ✓ Weight scaling correctly computes the new price from generated yield (185ms)
       ✓ Weight scaling induced price should not exceed one (155ms)
       ✓ Weight scaling should make the price increase toward 1 without any trades (448ms)
  Negative cases for AMM Initialize function [10]
       ✓ Interface is not ERC1155_ERC165 (46ms)
       ✓ Underlying address is zero (45ms)
       ✓ FutureVault address is zero (44ms)
       ✓ Admin address is zero (44ms)
       ✓ FeesRecipient address is zero (46ms)
       ✓ Initialized AMM for togglePause (120ms)

APWine AMM Registry
  With Registry deployed
     ✓ Sets future vault AMM pool correctly
     ✓ Can override previously added future vault AMM pool
     ✓ Is empty by default
     ✓ Can only set future vault AMM pool as admin
     ✓ Can register AMM
     ✓ Only Admin can register AMM
     ✓ Cannot register same AMM again
     ✓ Can remove AMM from registry
     ✓ Cannot remove an AMM again
     ✓ Only an admin can remove an AMM
     ✓ Returns true for registered AMM
     ✓ Returns false for unregistered AMM

AMM LPTokens
  With Token Contract deployed
     ✓ Can predict lp tokens ID
     ✓ Can mint AMM pool tokens
     ✓ Only MINTER can mint AMM pool tokens
     ✓ Can burn AMM pool tokens
     ✓ Only MINTER can burn AMM pool tokens
     ✓ Cannot burn tokens more AMM pool tokens than balance
     ✓ Can toggle the paused state
     ✓ Only PAUSER can toggle pause
     ✓ Returns AMM Id from LPToken Id
     ✓ Returns Period Index from LPToken Id
     ✓ Returns Pair Id from LPToken Id

AMM MasterChef
```

```
  With future deployed [1]
    With funds deposited in future [2]
      Negative cases for MasterChef Initialize function [3]
         ✓ APW address is zero
         ✓ LPTOKEN address is zero
         ✓ APWPERBLOCK is zero
      With MasterChef initialized [4]
         ✓ Can't add a pool for a invalid lp token Id
         ✓ Can add a pool for lp token Id only once (73ms)
         ✓ Invalid Pool ID
         ✓ Update Pool for TokenId's  (41ms)
        With pool added for lp token [5]
           ✓ Owner can set allocPoint per block
           ✓ Only owner can set allocPoint per block
           ✓ Owner can set APW per block
           ✓ Only owner can set APW per block
           ✓ APW per block cannot be set to zero (47ms)
           ✓ Owner can remove APW from the contract
           ✓ User can get the registration state of a lp token Id
           ✓ User can get the lp token id at index
           ✓ User can get the lp tokens list length
           ✓ User can deposit in pool (137ms)
           ✓ User can update the pool (40ms)
           ✓ User can update all the pools (40ms)
          With lp tokens deposited in a pool [6]
             ✓ User can withdraw all token in case of emergency (38ms)
             ✓ User can withdraw from a pool (86ms)
             ✓ Withdrawing from the pool collect apw rewards as well (107ms)
             ✓ Depositing in the pool again collect apw rewards as well (136ms)
             ✓ User rewards accumulates in time while the pool is being updated (83ms)
             ✓ User rewards stops accumulating if allocation point set to 0 (166ms)
             ✓ Rewards can be updated while setting new allocation point (89ms)
             ✓ User Pending rewards for other pool are null  (92ms)
             ✓ User can get the list of LpToken Id it is registered to
             ✓ User cannot withdraw more lp token in the pool than initially deposited (60ms)
             ✓ User can withdraw less amount then deposit (62ms)
             ✓ User can get its pending reward after switching period without updating before (1070ms)
            With AMM switch period  [7]
               ✓ User deposit in pool get reverted for invalid Id (132ms)
               ✓ User can get pending APW from last period
               ✓ User can withdraw APW from last period (91ms)
               ✓ A new user can join the pool of the next period (1087ms)
              With fund deposited in the new period by another user [8]
                 ✓ User rewards from last period stop accumulating when period was switched by another user (87ms)


169 passing (27s)
1 failing

1) APWine AMM
     With future deployed [1]
       With funds deposited in future [2]
         With initial liquidity and another user in future [3]
           Last period of LP is being updated correctly when joining an ongoing period:
   AssertionError: Expected "0" to be equal 1
    at Context.<anonymous> (test/AMM.test.ts:321:93)
    at runMicrotasks (<anonymous>)
    at processTicksAndRejections (internal/process/task_queues.js:93:5)
    at runNextTicks (internal/process/task_queues.js:62:3)
    at listOnTimeout (internal/timers.js:523:9)
    at processTimers (internal/timers.js:497:7)
```

Code Coverage

The code coverage is high for most of the contracts. We still found several logical errors in the AMM and we recommend achieving 100% code coverage for the AMM contract and writing tests for all the issues found. We also recommend writing tests to ensure that the novel weight adjustments functionality matches the results of the simulation described in the paper.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	98.1	84.44	97.89	98.12	
AMM.sol	98.56	81.9	100	98.56	290,667,705,710
AMMRegistry.sol	100	83.33	100	100	
AMMRouter.sol	80	50	71.43	82.14	72,73,77,78,79
LPToken.sol	100	90	100	100	
MasterChef.sol	100	92.86	100	100	
RoleCheckable.sol	100	100	100	100	
contracts/interfaces/	100	100	100	100	
IAMM.sol	100	100	100	100	
IAMMRegistry.sol	100	100	100	100	
IAMMRouter.sol	100	100	100	100	
IAPWineIBT.sol	100	100	100	100	
IController.sol	100	100	100	100	
IERC1155.sol	100	100	100	100	
IERC20.sol	100	100	100	100	
IFutureVault.sol	100	100	100	100	
IFutureWallet.sol	100	100	100	100	
ILPToken.sol	100	100	100	100	
IProxyFactory.sol	100	100	100	100	
IRegistry.sol	100	100	100	100	
contracts/library/	100	69.23	100	100	
AMMMaths.sol	100	69.23	100	100	
contracts/test/mock/	100	100	100	100	
MockToken.sol	100	100	100	100	
All files	**98.44**	**82.52**	**98.21**	**98.44**	

Appendix

File Signatures

The following are the SHA-256 hashes of the reviewed files. A file with a different SHA-256 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different SHA-256 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review.

Contracts

```
8bf78038f6fe50c8609b0d18a0332731d40d073208a2b847cd8e854adf09f7fd ./contracts/AMM.sol
fcd4aefd27adfd610eaf2e934aff46412a62a134e30f3b856c3db8f501dc235a ./contracts/RoleCheckable.sol
be20337dd54a493d2990aad5fecf547d5a6d3f5e30a1a2916503e294c59e2440 ./contracts/LPToken.sol
0c4f1e8bc8bddce348787b0e1219bc1d1e9ceac45ef333fb3e693e4c2bde4ffb ./contracts/AMMRouter.sol
f013fcfbbebb7b69cefb4c28dcc935c547c2a445a585eb23dd6fca079d3454fa ./contracts/MasterChef.sol
6fe61e13d67ba6052ea9ef80f591dccb20298face855d622fa308f1ef4ef5ff5 ./contracts/AMMRegistry.sol
da7ff19a0f8583215139e3d39c11594b00390ade3dc04f93631542c516ef71d8 ./contracts/interfaces/IERC20.sol
b2fa51f3d6b00feb7f80dcc778bc0586a39a876ccd0c99e80ba0eb63cf68a3f0 ./contracts/interfaces/IERC1155.sol
b47e02b1105575e8f594dd4e2be759331a3e2fd7e52ccb87c4a67e2e2ff40224 ./contracts/interfaces/IFutureVault.sol
5708d6832ac816741003073d1dc577ee19b5878d24fc06242b26eab9912e6ea7 ./contracts/interfaces/IAPWineIBT.sol
07ad4168a835f82bf370e06464269a8f689acf39ac9abcc4e59c16b049ff750b ./contracts/interfaces/IRegistry.sol
06dfaa3eb41cdce2bee6d90b17d961fc5519a5e51b746936799635b9c6875408 ./contracts/interfaces/IProxyFactory.sol
b1679011d6a0f65c9b92c9999dc63ac216ff7a84bd8e3db0070a83e720774988 ./contracts/interfaces/IAMMRegistry.sol
e02df740ee33e34064f60cb688fd257738c6f6eb76f2c7f199259ecebca25d4d ./contracts/interfaces/IAMMRouter.sol
896edcc76ee7713c15e12b8538f87ecc4445785548a29bf2d2887cf17150b8c5 ./contracts/interfaces/ILPToken.sol
95d0661a54e05764fc2adf26e8bc1633bffa83cd7ef2215ab7ae3065b64d43c3 ./contracts/interfaces/IFutureWallet.sol
ec689508d5b17178cd83358874734cf73459a51fd43678e76392a0aa30178f83 ./contracts/interfaces/IAMM.sol
1fb18410453321a92a0b070743881ae566812676870b297a8db838df0d6344fb ./contracts/interfaces/IController.sol
ce395800d596f9eb4e7616ed4fb86a985808b942f3b9f3eae6b3b4931c03afb1 ./contracts/library/AMMMaths.sol
53a891db0250945ca16df29c8f65a6c926bc9188fd2556a7fd126c90aeea0cbc ./contracts/test/mock/MockToken.sol
```

Tests

```
7f5a5928efb2ab30208f540458f8795b9fb8a8714482f6376b97fc23c09cf918 ./test/MasterChef.test.ts
5744b84bd67b7fa845bb7d429c254537620f2c591a9e1659172b2016f85ecc9f ./test/AMMRegistry.test.ts
fc9d5e70df1252ab2b1957b63a977d35817ce1a2dc49d68e744a1b0057ddc0f1 ./test/AMM.test.ts
f1b87d12637cc07545a9707628cdcaabbf585ed4327eb86b7e4fa83a5d6f6d7e ./test/LPToken.test.ts
d19e816b3f32c73d3f91a1c1a0f1cadc176f1e73113b68362833bbf58ee05a85 ./test/util/poolMath.ts
3de4ed6aecadd311cdc483de0ee903fb31e919fac4fe0d16e79c8fe70a44c1be ./test/util/expectRevert.ts
cd0326e80ff19f71d473164c9365331fd3feb10e291365724b474bbd9c66d227 ./test/util/expectEvent.ts
40fd0304263759eb644f621a9108d6e088c73098e25fcc769125f5943b3497a4 ./test/util/skipBlocks.ts
82ce684c4c40f4c0debc510296cd9b4537468fe3cc29838bcd9eaac8e17d6bff ./test/util/mapValues.ts
f8ecff576dab06fb33a9559de7d9fed8acad824a999b4275668e1907ee42de5c ./test/util/expectEq.ts
2aab26261871d163a8db0af507b9212d07e59a2f9e1b8d500b7171b9c2dba817 ./test/util/deploy.ts
0c6a034b0b981e8099d5a33d2f6fba950dc100d29b5613b60179a7d2e85ccf42 ./test/util/deployMockToken.ts
6e1df630ced971bad2173f124d61becb436aba339db25061c310daf1b4ed3534 ./test/util/fixture.ts
```

Changelog

- 2021-09-21 - Initial report
- 2021-10-11 - Re-audit
- 2021-10-26 - Re-audit fixes

About Quantstamp

Quantstamp is a Y Combinator-backed company that helps to secure blockchain platforms at scale using computer-aided reasoning tools, with a mission to help boost the adoption of this exponentially growing technology.

With over 1000 Google scholar citations and numerous published papers, Quantstamp's team has decades of combined experience in formal verification, static analysis, and software verification. Quantstamp has also developed a protocol to help smart contract developers and projects worldwide to perform cost-effective smart contract security scans.

To date, Quantstamp has protected $5B in digital asset risk from hackers and assisted dozens of blockchain projects globally through its white glove security assessment services. As an evangelist of the blockchain ecosystem, Quantstamp assists core infrastructure projects and leading community initiatives such as the Ethereum Community Fund to expedite the adoption of blockchain technology.

Quantstamp's collaborations with leading academic institutions such as the National University of Singapore and MIT (Massachusetts Institute of Technology) reflect our commitment to research, development, and enabling world-class blockchain security.