Quantstamp Security Assessment Certificate

August 2nd 2021 — Quantstamp Verified

Siren Markets V2

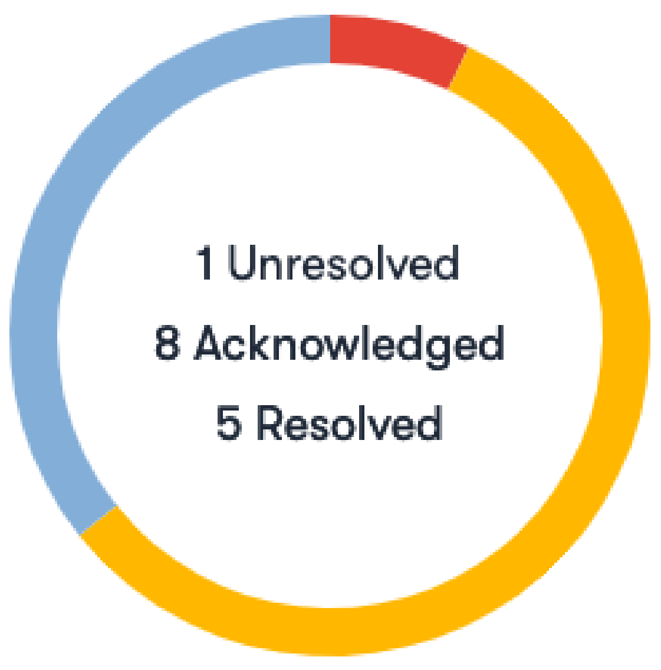
This smart contract audit was prepared by Quantstamp, the protocol for securing smart contracts.

Executive Summary

Type	DeFi protocol
Auditors	Kacper Bąk, Senior Research Engineer Fayçal Lalidji, Security Auditor Jose Ignacio Orlicki, Senior Engineer
Timeline	2021-06-16 through 2021-07-30
EVM	Muir Glacier
Languages	Solidity, Javascript
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	Pascal Club Core Smart Contracts
Documentation Quality	Medium
Test Quality	Medium
Source Code	

Repository	Commit
core	4f934d2

Goals	• Can funds get locked up in the contract? • Are the computations implemented correctly?
Total Issues	14 (5 Resolved)
High Risk Issues	2 (0 Resolved)
Medium Risk Issues	4 (2 Resolved)
Low Risk Issues	6 (3 Resolved)
Informational Risk Issues	1 (0 Resolved)
Undetermined Risk Issues	1 (0 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

During the review we have found a few issues. Notably, two of the issues are high severity, whereas four are medium severity. The high severity issues are related to the mechanics and pricing of options. Furthermore, we recommend improving the test coverage. We advise against deploying the contracts "as is".
Update: the team has addressed or acknowledged most of the issues as of commit `d110d71`.

ID	Description	Severity	Status
QSP-1	European Options	High	Acknowledged
QSP-2	Options Pricing Model is Inaccurate	High	Acknowledged
QSP-3	Potentially Stale Prices	Medium	Unresolved
QSP-4	Minter Role Not Removed	Medium	Fixed
QSP-5	`setSettlementPrice()` Must Be Called at a Very Specific Interval	Medium	Acknowledged
QSP-6	Unchecked Return Values	Medium	Mitigated
QSP-7	Missing Checks for Arguments of Type Address to Be Non-zero	Low	Fixed
QSP-8	Privileged Roles and Ownership	Low	Acknowledged
QSP-9	Interaction with External Contracts	Low	Acknowledged
QSP-10	Gas Usage / `for` Loop Concerns	Low	Fixed
QSP-11	Token Pair Prices	Low	Fixed
QSP-12	No Sweep Functionality	Low	Acknowledged
QSP-13	Chainlink Oracle May Be Manipulated	Informational	Acknowledged
QSP-14	`setVolatilityFactor()` Does Not Check The Upper Bound	? Undetermined	Acknowledged

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 v0.7.1

Steps taken to run the tools:

1. Installed the Slither tool: `pip install slither-analyzer`
2. Run Slither from the project directory: `slither .`

Findings

QSP-1 European Options

Severity: High Risk

Status: Acknowledged

File(s) affected: `MarketController.sol`

Description: By definition European options can only be exercised on the day before expiration; for further details see here. In the current implementation of `exerciseOption()`, European options are allowed to be executed only after expiration, as commented in L905: `We support only European so we exercise only after expiry`, which is incorrect. Strictly speaking, an option buyer should not be allowed to execute an expired option since the underlying asset spot price will be subject to changes after expiration, meaning that an option that expired out of the money can be executed while expired and be in the money.

Recommendation: Please clarify whether this is the intended behavior. We recommend aligning the implementation with the definition of European options.

Update: The team informed us that they will specify in the documentation for the `MarketController` that their European options mean the bTokens can be exercised any time after expiration, however the spot price (S in the equations for options on wikipedia) used to calculate the payoff will be the spot price at the expiration date. They will make note that this definition of European option is slightly different than in TradFi, and that the reason they do it this way is so that there is no longer a bounded time window where users need to exercise their options, like they had in v1 with American style options. Now users can exercise their options *anytime* after expiration (but it will always use a value of S set right at expiration).

QSP-2 Options Pricing Model is Inaccurate

Severity: High Risk

Status: Acknowledged

File(s) affected: `MinterAmm.sol`

Description: The options pricing formula approximation in `calcPrice()` is inaccurate, mainly because it does not include a discount rate and the standard probability measure. Also, the natspec documents that the volatility factor is `0.4`, but the factor itself is passed as an argument and it may be an arbitrary value. Finally, the time scaling factor `timeUntilExpiry` appears to be incorrect. Even if the discount rate is assumed to be 1, the Black-Scholes formula uses function `N(.)` that is the standard normal cumulative distribution.

Recommendation: Document why the implemented approximation is good enough, or provide an implementation that more closely corresponds to Black-Scholes formulation. Furthermore, document your assumptions on the discount factor (based on risk-free interest rate).

Update: The team informed us that they do not use the true Black-Scholes pricing formula onchain, because it is too computationally expensive to compute, instead they use an approximation for at-the-money options from Brennan-Subrahmanyam (here: https://www.researchgate.net/publication/245065192_A_Simple_Formula_to_Compute_the_Implied_Standard_Deviation). This approximation simplifies the standard cumulative normal distribution and removes the interest rate (r) term. They will provide a link to this in the documentation and code.

QSP-3 Potentially Stale Prices

Severity: Medium Risk

Status: Unresolved

File(s) affected: `ChainlinkEthUsdProxy.sol`, `PriceOracle.sol`

Description: Prices fetched by `latestRoundData()` are not checked to be up to date. Specifically, `PriceOracle.getCurrentPrice()` does not validate the answer, i.e., it does not check the values `startedAt` and `updatedAt`, meaning that the latest round may be obsolete since the Chainlink aggregators rely on external nodes to be updated once a request is submitted by one of the sponsors. The same issue occurs in `ChainlinkEthUsdProxy.latestRoundData()`.
It is important to note that a price that is stale by multiple hours or days may enable an out of the money option in the money or the opposite.

Recommendation: We recommend adding relevant checks. Furthermore, if the price is stale, we recommend implementing a fallback oracle (in-house or a decentralized oracle like Uniswap).

Update: The team informed us that the contract currently settles the option by using the latest price data provided by the Chainlink Oracle at 8 am UTC on expiration Friday. The price data returned by Chainlink also provides an `updatedAt` timestamp field. If the protocol needs to check the field, it would also need to define after how much delay a price is considered stale. Or, alternatively, it could wait for the first price update after expiration timestamp.
According to the development team both alternatives would delay the settlement of the market/series until the relevant price update has been pushed on-chain, and in theory this could happen well after the expiration timestamp. Moreover, there would be more uncertainty before the expiration timestamp at exactly what price the market will settle. The development team believes that the current logic strikes the right balance of functionality and simplicity for their users.

QSP-4 Minter Role Not Removed

Severity: Medium Risk

Status: Fixed

File(s) affected: `ERC1155Controller.sol`

Description: The function `ERC1155Controller.setMarketController()` adds a new controller address without removing `MINTER_ROLE` from the previously used address.

Recommendation: Remove minter upon setting the controller.

QSP-5 `setSettlementPrice()` Must Be Called at a Very Specific Interval

Severity: Medium Risk

Status: Acknowledged

File(s) affected: `PriceOracle.sol`

Description: The function `setSettlementPrice()` must be called at a very specific time interval. It may happen that the network gets jammed or the gas price is prohibitively expensive which would prevent function invocation. Consequently, the function may fail to set the settlement prices correctly for a given `underlyingToken` and `priceToken` pair due to the block gas limit error.
Another issue with `setSettlementPrice()` is that the price set for previous unset timestamp will not be accurate since it will use the actual return price to set spot prices of previous days or weeks depending on the contracts parameters.

Recommendation: We recommend informing users of such a possibility, and, perhaps designing a backup plan if the function cannot be executed. One possibility may be a function that can be called for a specific timestamp to allow setting unset spot prices.

Update: The team informed us that In order for a Market to settle, the `setSettlementPrice()` function has to be called by a transaction. The current implementation of `setSettlementPrice()` allows anyone to call the function to set settlement prices at market expiration.
Regarding the first issue (function calls at a very specific time interval), the development team believes that it will never be an issue for two reasons: firstly, the gas cost of executing the function is 50,000 gas units for each tokenPair. Therefore, even with extremely high gas costs (>500 Gwei), calling such function will cost at most 50-100$ per each pair, a price that they consider reasonable for them to bear. Secondly, outright censorship of their transactions regardless of gas prices cannot be sustained for more than a few blocks at most within a colluding majority of miners (i.e the chain undergoing a 51% attack).
Regarding the second issue (the `setSettlementPrice()` settles older unset market at the current price), they acknowledge this could be a very real problem, and for now they will rely on the timely inclusion of `setPriceSettlement()` transactions in the chain. If they observe that transactions are delayed (i.e more than 1 hour), they will upgrade the contract with a function that allows setting a specific date to a specific price, and is only callable by the owner contract.

QSP-6 Unchecked Return Values

Severity: *Medium Risk*

Status: Mitigated

File(s) affected: `Market.sol`, `MinterAmm.sol`

Description: Return value of `transfer()` is not checked in the following functions:

- `MinterAmm.withdrawCapital()`, (**Update:** fixed)
- `MintAmm._sellOrWithdrawActiveTokens()`,
- `MintAmm._sellOrWithdrawActiveTokens()`,
- `MintAmm.bTokenBuy()`,
- `Market.selfDestructMarket()`. (**Update:** fixed)

It may lead to moderate financial losses due to failed transaction that are assumed to be completed.

Recommendation: We recommend adding relevant checks.

Update: The issue is partially fixed.

QSP-7 Missing Checks for Arguments of Type Address to Be Non-zero

Severity: *Low Risk*

Status: Fixed

File(s) affected: `ChainlinkEthUsdProxy.sol`, `MarketController.sol`, `MinterAmm.sol`

Description: The following functions do not check if arguments of type address are non-zero:

- `ChainlinkEthUsdProxy.constructor()`,
- `MarketController.__MarketController_init()`,
- `MinterAmm.upgradeAmmForV2Markets()`.

Recommendation: We recommend adding the relevant checks.

QSP-8 Privileged Roles and Ownership

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `AmmFactory.sol`, `SimpleToken.sol`, `MarketVault.sol`

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

- `AmmFactory` is an ownable contract;
- `SimpleToken` allows the burner and minter roles to directly burn users' funds and mint any number of tokens. The same applies to `ERC1155Controller.mint()`, `ERC1155Controller.burn()`;
- `Controller` can withdraw all funds from the `MarketVault` via `MarketVault.setERC20ApprovalForController()` and `setERC1155ApprovalForController()`.

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.

QSP-9 Interaction with External Contracts

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `AmmFactory.sol`

Description: The protocol relies on functionalities of external contracts. Therefore, security of the project depends on on these contracts. While we are unaware of any immediate issues, it is important to note that defi protocols may be vulnerable to flash loan attacks, market manipulation, computational errors, etc. Furthermore, we also want to note that the project assumes constant supply of the external tokens, i.e., inflationary and deflationary tokens are not compatible.

Recommendation: We recommend reviewing external contracts to make sure they work as expected.

Update: The team informed us that the external contracts used by the `AmmFactory.sol` contract are all vetted by the Siren Team, and only a privileged owner user can call the functions which interact with external contracts (e.g. `AmmFactory.initialize()` and `AmmFactory.createAmm()`)

QSP-10 Gas Usage / `for` Loop Concerns

Severity: *Low Risk*

Status: Fixed

File(s) affected: `MinterAmm.sol`, `PriceOracle.sol`

Description: Gas usage is a main concern for smart contract developers and users, since high gas costs may prevent users from wanting to use the smart contract. Even worse, some gas usage issues may prevent the contract from providing services entirely. For example, if a `for` loop requires too much gas to exit, then it may prevent the contract from functioning correctly entirely. Specifically, we noticed the following loops:

- for loop in `MinterAmm.sol` in lines L383, 432, 525, 590,
- while loop in `PriceOracle.sol` in L98.

Recommendation: If possible, we recommend breaking the loops into individual functions or enabling a way of iterating in batches. Furthermore, we recommend performing a gas analysis to find out the upper limits.

QSP-11 Token Pair Prices

Severity: Low Risk

Status: Fixed

File(s) affected: `PriceOracle.sol`

Description: Chainlink aggregators do not implement all the possible tokens pair prices. However, the implementation of `PriceOracle.getCurrentPrice()` suggest otherwise.

Recommendation: When adding the oracle address, the token pair should be validated against the oracle to confirm that it returns a correct price.

Update: The team informed us that the settlement price comes from the on-chain aggregator. Multiple oracles could collude to manipulate prices and make them drift from the actual spot price, therefore settling the market at an unfair price. Alternative oracle systems were considered, such as Uniswap V3, which provides a TWAP price oracle. Although having their contracts refer to DEX prices is appealing to the team, as it would be more censorship resistant and decentralized than relying on Chain Link, but there are issues. Firstly, the TWAP price the market settles is not exactly the spot price and traders would need to be aware of that. Secondly, DEX prices can also be manipulated by large trades or from changes in the liquidity. This is especially important in the current scenario of greater liquidity fragmentation (UniSwap V3, Uniswap V2, Uniswap V3 on different Layer-2 solutions such as Optimism, Arbitrum). Although in the future the team may reconsider their choice, currently they believe Chainlink oracle strikes the best balance between risks and usability.

QSP-12 No Sweep Functionality

Severity: Low Risk

Status: Acknowledged

File(s) affected: `SimpleToken.sol`, `Market.sol`

Description: Contracts `SimpleToken` and `Market` have self-destruct features but they lack the sweep functionality to transfer any ERC20 tokens that might have been sent to the contracts' addresses.

Recommendation: Add sweep functionality so that the owner can transfer out any ERC20 tokens before destroying the contracts.

Update: The team removed the self destruct functionality from `SimpleToken.sol`, however, it is important to note that funds can still be sent by mistake, and a sweep function may be useful in this case.

QSP-13 Chainlink Oracle May Be Manipulated

Severity: Informational

Status: Acknowledged

File(s) affected: `PriceOracle.sol`, `ChainlinkEthUsdProxy.sol`

Description: The protocol uses Chainlink as a price oracle. Although unlikely, it is possible that the reported price is manipulated.

Recommendation: Consider using other oracles in case Chainlink price gets manipulated or is unavailable.

QSP-14 `setVolatilityFactor()` Does Not Check The Upper Bound

Severity: Undetermined

Status: Acknowledged

File(s) affected: `MinterAmm.sol`

Description: The function checks only lower bound for the volatility factor, but does not check the upper bound.

Recommendation: Unless there is a good reason not to, we recommend adding an upper bound check.

Update: The team acknowledged there is no upper bound. They do not know how high individual tokens' volatilities may rise, so they do not want to put an upper bound.

Automated Analyses

Slither

Slither did not report any issues.

Adherence to Specification

Inline comments in `Market` inform that `collateralDecimals` must be used because `wToken` and `bToken` will be denominated in the same decimals as the collateral. In the implementation, however, `underlyingDecimals` is used for `bToken` and `wToken`. We recommend aligning specification with the implementation.

Adherence to Best Practices

Update: acknowledged.

1. TODO items:
   1. `MinterAmm.sol`, L115, 502,
   2. `ERC1155Controller.sol` L21, 35,
   3. `PriceOracle.sol` L32, 150,
   4. `MarketController.sol`, 84, 123, 1120.
2. The syntax `.value(..)` was deprecated (in July 2020, for version 0.7.0) in favor of `{value: ..}`. We recommend updating the syntax in `GovernorAlpha.execute()` and `Timelock.executeTransaction()`.

Test Results

Test Suite Results

All the tests executed successfully.

```
  Contract: Chainlink ETH/USD Proxy
     ✓ Calculates price correctly (225ms)

  Contract: AMM Call Verification
     ✓ Provides capital without trading (783ms)
     ✓ Provides and immediately withdraws capital without trading (396ms)
     ✓ Provides capital with trading (1411ms)
     ✓ Buys and sells bTokens (1974ms)
     ✓ Sells wTokens (1292ms)
     ✓ Withdraws large share of LP tokens (1215ms)
     ✓ Sells more bTokens than wTokens in the pool (1052ms)
     ✓ Enforces minimum trade size (43ms)
     ✓ Works in initial state (99ms)
     ✓ should calculate correct total pool value with multiple expired  ITM markets (1904ms)

  Contract: Minter AMM Expired
     ✓ Expired OTM with constant price (1304ms)
     ✓ Expired ITM with exercise (2247ms)
     ✓ should claimExpiredTokens succeed (1217ms)
     ✓ claimAllExpiredTokens should succeed (1238ms)

  Contract: AMM Markets Limit Verification
     ✓ Expect revert when adding more than 100 markets (10079ms)
     ✓ Add 100 markets successfully (9816ms)

  Contract: AMM Put Verification
     ✓ Provides capital without trading (626ms)
     ✓ Provides and immediately withdraws capital without trading (410ms)
     ✓ Provides capital with trading (1675ms)
     ✓ Enforces minimum trade size (51ms)
     ✓ Works in initial state (121ms)

  Contract: Minter AMM Remove expired markets
     ✓ Add markets to amm (182ms)
     ✓ All markets expired (265ms)
     ✓ 3 open markets 2 expired markets (486ms)
     ✓ 1 open & 1 market expired(last one added to the open markets) (239ms)
seed: 0.2262358971655094
     ✓ fuzz test searching through many different open/expired permutations (6547ms)

  Contract: AMM Upgradeability
     ✓ should fail to deploy AMM when an AMM with that token triplet has already been deployed (68ms)
     ✓ Fail to upgrade from non-owner account
     ✓ should upgrade and be able to call function on upgraded contract (249ms)
     ✓ should fail to call function when upgrading to a non-MinterAmm implementation contract
     ✓ Should fail to initialize twice

  Contract: AMM Pricing
     ✓ should calculate correctly when USDC is the collateral token (2104ms)
     ✓ should calculate correctly when WBTC is the collateral token (1816ms)

  Contract: Volatility Factor
     ✓ Enforces Limits (80ms)

  Contract: Proxy ERC1155Controller Verification
     ✓ Cannot initialize twice (712ms)
     ✓ should upgrade correctly (844ms)

  Contract: Governance Verification
     ✓ Initializes governance and creates a market (773ms)

  Contract: Cash Settlement
     ✓ claim before any exercises (341ms)
seed: 0.16615905679542187
     ✓ fuzz exercising options and claiming collateral with multiple users (7630ms)
     ✓ should use the current oracle price when no settlement price is set (400ms)
     ✓ should not be able to claimCollateral if market not expired (304ms)

  Contract: MarketController close
     ✓ Can close out a Call market (1129ms)
     ✓ Can close out a Put market (1075ms)

  Contract: Market Fees
     ✓ Sends fees to the owner account (958ms)

  Contract: Market Scenarios
     ✓ Calculates call option decimals for wBTC and USDC at $20k strike (1298ms)
     ✓ Calculates put option decimals for USDC and wBTC at $12k strike (1176ms)

  Contract: PriceOracle verification
    Failures
       ✓ should fail to set the same token oracle twice
       ✓ should fail to set settlement price before setting an oracle (79ms)
       ✓ should fail if the oracle returns a negative value (44ms)
       ✓ should fail to update logic contract to 0 address
       ✓ should fail to update logic contract with non-owner (84ms)
       ✓ should fail to get current price before setting an oracle (84ms)
       ✓ should fail to get settlement price before setting an oracle (76ms)
       ✓ should fail to set a price for a specific settlement date before setting the oracle (103ms)
       ✓ should fail to set a price for a specific settlement date for a non-aligned date
       ✓ should fail to set a price for a settlement date in the future
       ✓ should fail to set a price for a settlement date such that there would be gaps in the dates that have had their prices set (71ms)
    Successes
       ✓ should set settlement price (58ms)
       ✓ should not change price when setting the same token + settlementDate multiple times (115ms)
       ✓ should be able to set the price for the same settlement date but different token (192ms)
       ✓ should return isSet == false when calling getSettlementPrice for a date where the settlement price hasn't been set
       ✓ should be able to set multiple settlement dates with a single call to PriceOracle.setSettlementDate (69ms)
       ✓ should upgrade correctly (846ms)
       ✓ should set settlement price for a specific date (126ms)

  Contract: Proxy Market Verification
     ✓ Cannot initialize twice (825ms)
     ✓ Creates Market (335ms)
     ✓ Creates multiple Markets (976ms)
     ✓ fails to initialise Market with any tokens equal to the 0x0 address (657ms)
     ✓ fails to initialise Market with empty restrictedMinters arg (569ms)
     ✓ Calculates open and expired state (40ms)
     ✓ Mints option wTokens and bTokens (593ms)
     ✓ allows exercise if expired (263ms)
     ✓ Blocks redeem if prior to start of the exercise window (146ms)
     ✓ Allows exercise for European-style option (279ms)
     ✓ Allows claiming after expiration with no redemptions (560ms)
     ✓ Allows claiming after expiration with full redemptions (380ms)
     ✓ Allows claiming after expiration with partial redemptions (443ms)
     ✓ Allows closing a position while open (423ms)
     ✓ should upgrade correctly (570ms)

  Contract: Proxy Vault Verification
     ✓ Cannot initialize twice (833ms)
     ✓ should only allow the admin to set the vault's controller (784ms)
     ✓ setERC20ApprovalForController should succeed and fail as expected (852ms)
     ✓ setERC1155ApprovalForController should succeed and fail as expected (833ms)
     ✓ should upgrade correctly (965ms)

  Contract: Proxy Token Verification
     ✓ Initializes (70ms)
     ✓ Mints (71ms)
     ✓ Transfers (70ms)
     ✓ Admin Burns (56ms)
     ✓ Minting and burning should be gated to role (96ms)


  90 passing (2m)
```

Code Coverage

We recommend improving the coverage to achieve at least 80% in each category for each contract.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
amm/	92.83	65	88.89	93.17	
AmmFactory.sol	71.43	40.91	50	71.43	… 117,127,132
ChainlinkEthUsdProxy.sol	87.5	42.86	66.67	87.5	64,74
IAddMarketToAmm.sol	100	100	100	100	
ISirenTradeAMM.sol	100	100	100	100	
InitializeableAmm.sol	100	100	100	100	
MinterAmm.sol	95.74	73.08	100	96.15	… 06,707,1030
governance/	54.77	32.88	57.69	52.59	
GovernorAlpha.sol	66.98	39.71	66.67	62.89	… 324,325,326
SirenToken.sol	31.91	19.57	42.11	31.91	… 429,430,433
Timelock.sol	75.61	37.5	66.67	75.61	… 76,77,79,97
libraries/	81.82	50	100	88.89	
Math.sol	81.82	50	100	88.89	18
market/	94.72	78.03	92.65	94.79	
ERC1155Controller.sol	90	75	90	90.91	107,111
IERC1155Controller.sol	100	100	100	100	
IMarketController.sol	100	100	100	100	
IMarketVault.sol	100	100	100	100	
IPriceOracle.sol	100	100	100	100	
MarketController.sol	94.06	73.33	90.48	94.09	… 4,1131,1136
MarketLibrary.sol	100	100	100	100	
MarketVault.sol	100	75	100	100	
PriceOracle.sol	97.87	93.33	100	97.87	296
oz/	64.71	50	50	65.71	
EnumerableSet.sol	64.71	50	50	65.71	… 254,261,279
proxy/	100	50	100	100	
Proxiable.sol	100	50	100	100	
Proxy.sol	100	50	100	100	
token/	100	100	100	100	
IERC20Lib.sol	100	100	100	100	
ISimpleToken.sol	100	100	100	100	
SimpleToken.sol	100	100	100	100	
All files	**81.84**	**57.99**	**78.53**	**81.89**	

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

```
6f84d2fe384216c0473d6779b0ab0be3268d65eccc0cd2818dca569bdd8604c4 ./contracts/oz/EnumerableSet.sol
2b4346643ade537eabd1221ca4cbf0f6e4ad4a038c83e11f24cbf44dc88a0986 ./contracts/market/ERC1155Controller.sol
e8a17fc174403c6bbe29808b3e76d8458d4823634d68c6fefdd0d8392a11f1d2 ./contracts/market/IMarketController.sol
bf5f0950010ec68c4d24e395adedb1902cf1b34f84b490088ab96f058b671318 ./contracts/market/PriceOracle.sol
fa70d6abb135d771f215196ffb91d34fb911147a61180a72bace25eecbec9ce1 ./contracts/market/IPriceOracle.sol
3ba317a100e0dfac17c5fec591302b8d264c29a6aad05438b587b4b3a4400199 ./contracts/market/IERC1155Controller.sol
c9dc09cd95e17f727e867da0a6f5ef7445b637e6579fa599be698ea9c3898d68 ./contracts/market/IMarketVault.sol
765d0e1ceee9028be1bc39d59a5bcc497030237da8f8113999e1e28ac2850faf ./contracts/market/MarketController.sol
59c94d28a84083d273e983ca90b208582398305b12d0fff7034510f98a9caf52 ./contracts/market/MarketVault.sol
e9f08376aa0ed4ff2b16060d91e26ffa39b92958d58d2acb6f32e011c6c5b70e ./contracts/market/MarketLibrary.sol
3d7c983c84497374153193cb78ef35c8fbde6c1acec8e4d22a12be78481de77b ./contracts/governance/SirenToken.sol
618d304ffafd37e35e5dd2d980ae309ff677f8dd0931e1179a85a4d3facf125e ./contracts/governance/GovernorAlpha.sol
4fc2e550d274d1636424f5a17755c4a6b6fbf6d2e2addef7ffaf9571c51a0260 ./contracts/governance/Timelock.sol
ed098ffd2e843048c209f6eb6e1a6a8f46cb33a3034231cf9afe350992aedf0f ./contracts/amm/AmmFactory.sol
5695d6996a56fd7ed361c6f3dd3e047503b490289fc3caf5b7dda08fa3a72ac9 ./contracts/amm/ChainlinkEthUsdProxy.sol
a140403b7313b8da843d617b9f76c632fa146314a329cffa88c0d148ca1e8c7b ./contracts/amm/IAddMarketToAmm.sol
b295ec147384420b215e88f04de09a89988d2176ac612b46b2f53424eaaa2478 ./contracts/amm/MinterAmm.sol
207971a6549406304037f65158cd46b3ce7ab738c3c095e21259868b205c43b7 ./contracts/amm/ISirenTradeAMM.sol
baae6b28c7a929ba65071680c2ad93d9d46deef0239ad89e90644cbf4e6b14af ./contracts/amm/InitializeableAmm.sol
265a0b66639092cdc3921a54e207599f7982a7f1f787bef5bf6b38b9592b0f75 ./contracts/libraries/Math.sol
082e09c467191c8d81b33342c56b5386cb4e469ffd3b2969a10e6bc47e7818ec ./contracts/test/MockPriceOracle.sol
e3f52e24f037d3980d42a9a311e5104f5ecb916cb29a812242488f97a17042fe ./contracts/proxy/Proxy.sol
067300d342b73ad56f0988f57c4f31bb83de16cd1f50105fc44728263211c5f1 ./contracts/proxy/Proxiable.sol
7553567a9480a3b96189e0d7ac1e38916d9d1faeeadb118b4e68fdf15dd91b33 ./contracts/token/ISimpleToken.sol
e6a46f657c2b85f3e3c6dc5c80c3a55e6ce81210259039f75d4a1fddb968b764 ./contracts/token/IERC20Lib.sol
29fbc9e38de97bde9f065e10e439289323f29a61ccb62f111e57ff579d9e0257 ./contracts/token/SimpleToken.sol
```

Tests

```
082e09c467191c8d81b33342c56b5386cb4e469ffd3b2969a10e6bc47e7818ec ./test/MockPriceOracle.sol
```

Changelog

- 2021-07-20 - Initial report
- 2021-07-30 - Revised report based on commit `d110d71`.

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.