Quantstamp Security Assessment Certificate

February 24th 2021 — Quantstamp Verified

Pods Finance V2

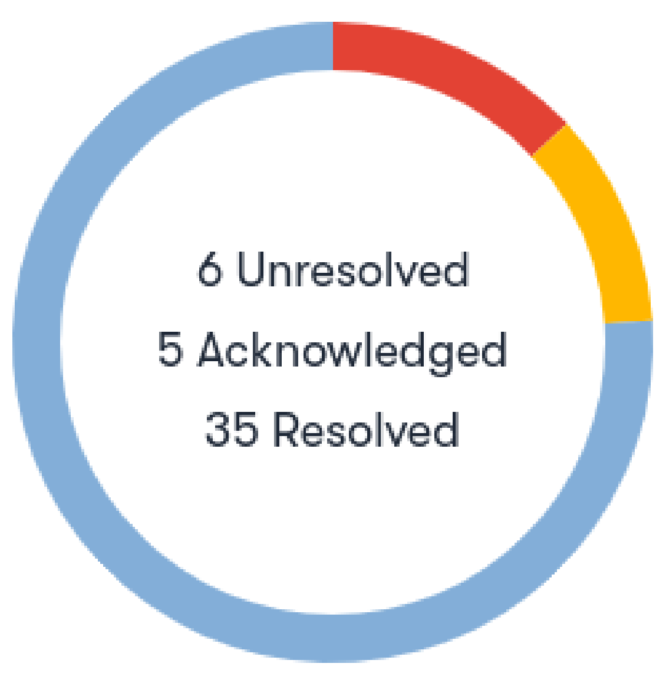
This security assessment was prepared by Quantstamp, the leader in blockchain security

Executive Summary

Type	Defi
Auditors	Leonardo Passos, Senior Research Engineer Fayçal Lalidji, Security Auditor Luís Fernando Schultz Xavier da Silveira, Security Consultant
Timeline	2020-10-20 through 2021-02-23
EVM	Muir Glacier
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	GitBook Documentation (private)
Documentation Quality	Medium
Test Quality	Medium
Source Code	

Repository	Commit
contracts (v2)	143d9ee
contracts (v2)	bbccd8e
contracts (v2)	80116eb

Total Issues	46 (35 Resolved)
High Risk Issues	10 (10 Resolved)
Medium Risk Issues	6 (4 Resolved)
Low Risk Issues	12 (9 Resolved)
Informational Risk Issues	10 (7 Resolved)
Undetermined Risk Issues	8 (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

This audit reports the findings of two engagements with the Pods team made between October'20 and February'21.

Our first audit iteration found a high volume of issues in the code (26), including seven high severity issues and five other medium ones. Poor documentation in the code as well as in the project's public domain proved to be an issue, as auditors could not fully grasp the implemented mechanism from the code alone.

In the second iteration, the Pods team re-worked their entire documentation site, while also enhancing the code documentation. At the same time, they addressed all high severity issues pointed in the first iteration. Nonetheless, new issues were found, while others remained unfixed. At present, there are 6 unresolved vulnerabilities. In total there are 46 issues, an increase of about 77% in the number of issues in comparison to the previous iteration.

As part of the second iteration, the Pods team also implemented mechanisms to perform emergency stops in the AMM contracts and to cap the amount of value handled by the contracts. These mechanisms were intended for a test phase. We have audited their code and included the relevant issues in this report.

This project has a large code base, whose complexity cannot be underestimated. While the documentation has greatly improved, and many changes have been done to improve the code, certain parts of the code still need to be better documented. Given its size, the code is subject to carrying bugs to production. We encourage a mitigation plan in place in case contracts need to be redeployed.

Disclaimer: The mathematics supporting the code was NOT subjected to analysis and validation, nor any economic attacks that could result from it. Under the assumption that the underlying mathematics is correct, this audit focused solely on the implemented code, its adherence to the provided documentation, and if one could exploit any of the smart contracts in ways unforeseen by the developers (excluded economic attacks).

Disclaimer: The Pods team performed several significant code changes between the audit and the submission of the code fixes, especially regarding the signedness of integer variables. These changes are out of scope and we cannot make sure they are correct.

ID	Description	Severity	Status
QSP-1	Poorly Documented Logic With Little Connection With White Paper	High	Mitigated
QSP-2	Loss Of Funds On Some Usage Patterns of `PodPut`	High	Fixed
QSP-3	Short Exercise Window May Prevent Option Exercise	High	Fixed
QSP-4	Withdraw Logic Is Not Weighted By Minters' Locked Balance	High	Fixed
QSP-5	Owner May Not Be The Target Recipient When Minting Pods	High	Fixed
QSP-6	Code Implements European Options Instead Of American	High	Fixed
QSP-7	Call Options Lack Explicit Contract Representation	High	Fixed
QSP-8	AMM Is Subject To Denial of Service	High	Mitigated
QSP-9	Fee Distribution Loss Due to Truncation	High	Fixed
QSP-10	Uncompliant European Option Implementation	High	Fixed
QSP-11	Transfers Could Occur Even After Options Expire	Medium	Acknowledged
QSP-12	Function `Blacksholes._getProbabilities` Does Not Match Formulation	Medium	Fixed
QSP-13	Function `Sigma.getCallSigma` Is Undefined	Medium	Fixed
QSP-14	Functions `BlackScholes.getCallPrice`and `getPutPrice` Do Not Match Formulation	Medium	Acknowledged
QSP-15	Incorrect Fee Calculation	Medium	Fixed
QSP-16	Incorrect Pricing Model Usage	Medium	Fixed
QSP-17	Potential Stale Oracle Price	Low	Fixed
QSP-18	Integer Overflow / Underflow	Low	Fixed
QSP-19	The `decimals` Function Is Optional In The ERC20 Standard	Low	Fixed
QSP-20	Option Exchange Liquidity Addition Misses Some ERC20 Calls	Low	Fixed
QSP-21	AMM Provider Liquidity Addition Misses Some ERC20 Calls	Low	Fixed
QSP-22	Input Validation Missing (1)	Low	Fixed
QSP-23	Input Validation Missing (2)	Low	Unresolved
QSP-24	Unchecked ERC20 `approve` Return Value	Low	Fixed
QSP-25	Integer Overflow / Underflow	Low	Unresolved
QSP-26	Use of Unassigned Value	Low	Fixed
QSP-27	Incomplete OptionExchange Mint Implementation	Low	Fixed
QSP-28	Input Validation Missing (3)	Low	Unresolved
QSP-29	Documentation Is Not In Natspec Format	Informational	Fixed
QSP-30	Copied Code Amongst Options	Informational	Mitigated
QSP-31	Fixidity Is A Clone	Informational	Acknowledged
QSP-32	Tokens Are Assumed To Be Implemented With OpenZeppelin's `ERC20`	Informational	Fixed
QSP-33	Unlocked Pragma	Informational	Fixed
QSP-34	Allowance Double-Spend Exploit	Informational	Mitigated
QSP-35	Incorrect Handling ETH Deposits Made by Mistake	Informational	Fixed
QSP-36	American Options Should Not Require An Exercise Window	Informational	Fixed
QSP-37	Wrong Variable and Method Names	Informational	Unresolved
QSP-38	Option Unmint Restrictions Can Be Avoided	Informational	Acknowledged
QSP-39	`mAB` Could Be A Zero Multiplier	Undetermined	Acknowledged
QSP-40	`wPodPut` and `waPodPut` May Not Have WETH As The Underlying Token	Undetermined	Mitigated
QSP-41	Use of (Third-Party) Untested Function	Undetermined	Fixed
QSP-42	Incorrect Multiplication Implementation	Undetermined	Fixed
QSP-43	Incorrect Shares Calculation	Undetermined	Fixed
QSP-44	Return Statements Found in Functions With No Expected Return	Undetermined	Fixed
QSP-45	AMM Has Implementation Oversights	Undetermined	Unresolved
QSP-46	Incorrect Implementation of the `uintToInt` Function	Undetermined	Unresolved

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.

Findings

QSP-1 Poorly Documented Logic With Little Connection With White Paper

Severity: High Risk

Status: Mitigated

File(s) affected: `contracts/amm/*`

Description: The AMM logic is poorly documented in the code, which impairs external auditing. Furthermore, the AMM implementation lacks proper links with the white paper. The latter, in turn, does not provide readers with high-level insights on the design choices and overall solutions, making it very hard to be read and understood. Hence, one cannot state with confidence whether the code is secure, nor state whether it adheres to the white paper.

Scope: audit iteration 1 (143d9ee)

Recommendation: Invest in high quality documentation as a means to foster independent auditing. The latter is crucial for users to gain confidence in using the platform. Specifically, we recommend the following:

- Provide high-level documentation that explains the internal algorithms and overall design
- Provide inline comments, tracing the code back to the public facing documentation

Update: The Pods team has greatly improved their documentation in comparison to the first auditing iteration. Nonetheless, it still falls short on certain aspects, which led us to mark this issue as mitigated instead of fixed. For instance, `FeePool.sol`, `Sigma.sol`, and `OptionAMMPool.sol` could be aided with comments in the code. The Gitbook still lacks intuitive explanations for most of the underlying math and why and if it works.

QSP-2 Loss Of Funds On Some Usage Patterns of `PodPut`

Severity: High Risk

Status: Fixed

File(s) affected: `contracts/options/PodPut.sol`

Description: When “fractional” amounts of tokens are put into contracts, they generally should be rounded up and, when such amounts are taken from contracts, they generally should be rounded down. Failure to do so, as in `mint` of `PodPut`, can have disastrous consequences.

Scope: audit iteration 1 (143d9ee)

Exploit Scenario: Example illustrating the issue

- Suppose a put option `BRL:USDC`, where both coins use two decimal places. The strike price is 0.21 USDC per BRL unit
- Alice mints 1000099 pod tokens (10,000.99 BRL equivalent), locking 1000099 units in the contract (lockedBalance) and adds exactly `trunc(0.21 * 1000099 = 210020.79) = 210020` of strike asset to the contract (2,100.20 USDC)
- Alice repeats the operation, so now she has 2000198 in locked balance and the contract has 420040 in strike asset
- Alice does not trade her option tokens
- Alice then proceeds to withdraw the liquidity she had previously provided. She has 2000198 in locked balance; the contract attempts to withdraw `trunc(0.21 * 2000198 = 420041.58) = 420041` of strike asset, but it has only 420040 in that asset. Therefore, the remaining 1 of strike asset (0.01 USDC) gets converted to 0.04 BRL (4 in underlying asset). However, the contract doesn't have any underlying assets and so the transaction reverts.
- Hence, Alice cannot withdraw her liquidity; in this case, she just lost 20,001.98 BRL.

Recommendation: Make sure that, in every contract receiving or sending funds, including the ones in `amm/` and `exchanges/`, the funds received are rounded up and the funds sent are rounded down. This is in order to make sure the contract always has enough funds to send.

Update: This appears fixed in the second audit iteration. The code now keeps track of the amount of shares of the reserve any given user has. Any withdrawal accounts for the proportion (%) of shares a user has, using that as a multiplier to the user provided funds. The resulting value is rounded down.

QSP-3 Short Exercise Window May Prevent Option Exercise

Severity: High Risk

Status: Fixed

File(s) affected: `contracts/options/.*PodPut.sol`, `contracts/options/PodOption.sol`

Description: Options do not enforce a minimum (and reasonable) exercise window length. Hence, a malicious actor could create options with a very short exercise window; pod holders, in turn, would not be able to exercise their options.

Scope: audit iteration 1 (143d9ee)

Exploit Scenario: Example illustrating the issue

- On Nov. 1st/2020, Bob creates a Put Option `wETH:USDC`, with a striking price of 400 USDC, expiration date on Nov. 30th/2020, 23:00:00pm, and an exercise window of 1 second
- Bob then mints 100 of such put options, locking 40,000 USDC. In this case, Bob operates as an insurer
- On Nov. 1st/2020, Alice buys 30 ETH at 400 USDC, paying a total of 12,000 USDC
- To insure her investment, Alice buys the Put Options issued by Bob, paying a premium
- During the month of November, ETH suddenly drops in price, e.g., it drops to 320 USDC
- Alice now has a loss of 12,000 - 9,600 = 2,400 USDC
- To alleviate her loss, Alices chooses to exercise her Put Options. However, as the average time to mine a block is 15s, Alice does not succeed, as she could only exercise her options within 1s past the expiration date
- Bob, the insurer, who was supposed to pay Alice 12,000 USDC to compensate for her losses, ends up paying nothing. Alice, in turn, is left with a loss of 2,400 USDC, in addition to the premium previously paid.

Recommendation: Define a minimum and reasonable amount of time for option holders to exercise their options if they so choose.

Update: A window of 24 hours has been added so that users can exercise their options if they so choose. Such a window applies to both call and put options.

QSP-4 Withdraw Logic Is Not Weighted By Minters' Locked Balance

Severity: High Risk

Status: Fixed

File(s) affected: `contracts/options/PodPut.sol`, `contracts/options/wPodPut.sol`

Description: The `withdraw` logic in `PodPut` and `wPodPut` do not distribute the strike balance according to the weight of each minter's locked balance, which is important in case some users had previously exercised part of the minted options. As is, minters could potentially suffer financial losses. Generally speaking, this can occur whenever put options are of the form `u1:s`, `u1:u2`, `s1:s2`, where `u1` and `u2` are underlying unstable coins and `s1` and `s2` are striking stable coins representing different fiat currencies.

Scope: audit iteration 1 (143d9ee)

Exploit Scenario: Example illustrating the issue

- Contract for `wETH:USDC` put options is created, with the striking price set to 400 USDC
- Bob mints 10 `wETH:USDC` put options
- Around the same time, Joe mints 20 `wETH:USDC` put options. Contract's strike balance is 30 * 400 USDC
- Alice buys 5 of those 30 minted options as a means to insure an investment she had made
- Eventually, the price of ETH drops to 100 USDC
- After the `wETH:USDC` put options expire, Alice exercises all her 5 options. Hence, the contract has an updated strike balance of 25 * 400 USDC, and underlying balance of 5 wETH
- Bob withdraws his 10 minted options, receiving 10 * 400 USDC (no loss)
- Joe attempts to withdraw his 20 options, but can only withdraw a maximum of 15. He receives 5 wETH to compensate for the lack of funds in the option contract. In total, Joe receives 15 * 400 + 5 * 100, which is equal to 6500 USDC. Given his initial investment of 20 * 400 (8,000 USDC), Bob has lost ~20% upon withdrawal. In comparison to Bob, Joe was the largest liquidity provider, but accrued all losses, while Bob had none

Recommendation: Change withdrawal logic to account for the weight put in by liquidity providers whenever they mint options, distributing strike tokens fairly.

Update: This issue has been fixed by distributing the funds in the options contract proportionally against minters' locked balance.

QSP-5 Owner May Not Be The Target Recipient When Minting Pods

Severity: High Risk

Status: Fixed

File(s) affected: `contracts/options/PodPut.sol`

Description: In the `mint` function, the `owner` parameter documentation states that it denotes *"which address will be the owner of the options"*. This matches the online specs: *"The seller has to lock collateral in the contract, mint PodPut and sell them in Uniswap at market rate"*, which leads to conclude that the address who provides the collateral is the same as the one who mints. However, in `mint`, tokens are minted having `msg.sender` as the recipient, which may not be the same as input `owner` address. If one invokes `mint` passing an owner that is different from `msg.sender`, he could potentially lose all the resulting minted tokens.

Scope: audit iteration 1 (143d9ee)

Recommendation: Remove the `owner` parameter; take it to be `msg.sender`.

Update: The Pods team stated that this is indeed an intentional behavior, and therefore, should not be considered an issue. The owner may not be the same as `msg.sender`, which brings the flexibility to mint options on someone's behalf.

QSP-6 Code Implements European Options Instead Of American

Severity: High Risk

Status: Fixed

File(s) affected: `contracts/options/.+PodPut.sol`, `contracts/options/PodOption.sol`

Description: Following the Pods online documentation, *"Pods options are American and therefore, can be exercised at any moment until maturity"*. However, in the given implementation, pods are European, i.e., they are a point in time instrument; buyers exercise them only when the expiration is reached. Following the current documentation, users may be misled when using the platform, expecting to buy/sell American options when in fact they are buying/selling European ones.

Scope: audit iteration 1 (143d9ee)

Recommendation: Update your public facing documentation s.t. it reflects the implementation. Alternatively, change the current code to match what the documentation states.

Update: After the changes since the first audit round, both European and American options are supported in the code, although the implemented pricing logic is only suited for European options.

QSP-7 Call Options Lack Explicit Contract Representation

Severity: High Risk

Status: Fixed

Description: Following Pods' online documentation, there are two types of options: `PodCall` and `PodPut`. However, the code provides a single contract (`PodPut` and its subtypes) to represent both option types (a call option is essentially a put option whose underlying and strike assets are swapped in comparison to the former). This is extremely confusing; using the current put contracts, one can create call options by essentially passing in the `PodOption.CALL` parameter. However, no documentation exists w.r.t the latter. This can mislead users when they interface with pods; i.e, one could incorrectly create a put option where a call was needed, or vice-versa.

Scope: audit iteration 1 (143d9ee)

Recommendation: Create a contract to explicitly represent `PodCall`. Then, every contract that is a subtype of `PodPut` should now be a subtype of `PodCall`. With the suggested fix, proceed to drop the `_optionType` in the constructor of `PodPut` and pass `PodOption.PUT` as the corresponding value to `PodOption` (`PodPut.sol`, L49).

Update: Call options are now captured explicitly; developers added a `PodCall.sol` contract.

QSP-8 AMM Is Subject To Denial of Service

Severity: High Risk

Status: Mitigated

File(s) affected: `contracts/amm/AMM.sol`, `contracts/amm/OptionAMMPool.sol`

Description: The pool balances of the AMM are tracked through the `balanceOf` method of its ERC20 tokens. One can do a denial of service on new pools by simply adding funds to the pool contract for either of the tokens in the pool, which would break the underlying assumptions of the initial balance state. Manipulating the pool factor is also possible, which could result in locked, lost or stolen funds or even in problematic contract logic. Further vulnerabilities cannot be discarded.

Scope: audit iteration 2 (bbccd8e)

Update (`80116ebc`): The Pods team did not follow our recommendation of tracking the AMM balances manually. Rather, they simply changed the code that detects an out-of-liquidity condition. As a result, the contract is apparently no longer vulnerable to the DoS attack described, but the pool factor can still be manipulated, especially if the option price is close to zero. Furthermore, one more place now needs to be fixed: the modifier `capped` in contract `CappedPool`.

Exploit Scenario:

- A pool for a pair of tokens A;B is created - initially it has no liquidity.
- Mallory, a malicious actor, transfer 1 unit of A to the pool's address. Hence, `A.balanceOf(pool)`is one.
- Bob, a liquidity provider, attempts to add liquidity to the pool. When doing so, the conditional in `AMM.sol` L245 will be false and the `_getFImpOpening` function will revert due to division by zero, since the deamortized balances are both zero. Bob will not be able to add any liquidity, nor will anybody else.

Recommendation: Instead of consulting the balance on the ERC20 tokens, keep track of the balances yourself.

QSP-9 Fee Distribution Loss Due to Truncation

Severity: High Risk

Status: Fixed

File(s) affected: `contracts/amm/FeePool.sol`

Description: When `FeePool` mints new shares, if the minted share value is higher than the fee to be distributed, the division in L90 can be truncated to a level where the result is zero. Similarly, the operation in L63 can also truncate results.

Scope: audit iteration 2 (bbccd8e)

Recommendation: Multiply the numerator by the base token unit before performing a division.

QSP-10 Uncompliant European Option Implementation

Severity: High Risk

Status: Fixed

File(s) affected: `contracts/options/*.sol`

Description: By definition European options can only be exercised on the day of expiration---see further details https://www.investopedia.com/terms/e/europeanoption.asp#:~:text=A%20European%20option%20is%20a,of%20or%20sell%20the%20shares. In the current implementation, however, European options are allowed to be executed only after expiration and within a predefined window after the expiration timestamp.
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 during the execution window in the money.

Scope: audit iteration 2 (bbccd8e)

Recommendation: To adhere to the strict definition of an European option, we suggest moving the exercise window right before the expiration timestamp (instead of right after, as presently implemented).

QSP-11 Transfers Could Occur Even After Options Expire

Severity: Medium Risk

Status: Acknowledged

File(s) affected: `contracts/options/.*PodPut.sol`

Description: The comments in the code state the following:

```
* Represents a tokenized european put option series for some
 * long/short token pair.

 * It is fungible and it is meant to be freely tradable until its
 * expiration time, when its transfer functions will be blocked
 * and the only available operation will be for the option writers
 * to unlock their collateral.
```

Transfers, however, are still enabled after expiration time, which contradicts the comment. Concretely, this allows transferring worthless pod tokens (they cannot be exercised) to recipients who could be led to believe they are worth something.

Scope: audit iteration 1 (143d9ee)

Update (`80116ebc`): The Pods team confirmed their intention is to only have the AMM pools stop once options expire. This is now implemented.

Recommendation: Following the comment, transfers should be blocked after the option expires. As such, we recommend overriding transfer-related functions in `PodPut` and subcontracts alike. However, we also recommend caution when blocking transfers, as integration with third-party DApps could break.

Update: The Pods team decided to not have a custom transfer function in their release. To mitigate the issue, they will warn users at the UI level, and adapt the documentation accordingly.

QSP-12 Function `Blacksholes._getProbabilities` Does Not Match Formulation

Severity: Medium Risk

Status: Fixed

File(s) affected: `contracts/amm/BlackScholes.sol`

Description: `_getProbabilities` does not match the specified formulation as per comments in the code:

```
* Get probabilities d1 and d2
     *
     ***********************************************************************************************
     * So = spotPrice
     * X  = strikePrice                                  t ( r + (  ✓² / 2 ) )
     *  ✓  = sigma              d1 = ln( So / X ) + --------------------
     * t  = time                                               ✓ ( sqrt(t) )
     * r  = riskFree
     * d2 = d1 -  ✓ ( sqrt(t) )
     ***********************************************************************************************
```

Specifically, `ln(So/X)` is not divided by `✓(sqrt(t))` as stated in the formulation (as given in the comments), whereas in the implementation it is.

Scope: audit iteration 1 (143d9ee)

Recommendation: Change the code so as to match the formulation in the comments; otherwise, it the formulation is incorrect, fix it accordingly.

Update: The code comment has been fixed to reflect the implementation.

QSP-13 Function `Sigma.getCallSigma` Is Undefined

Severity: Medium Risk

Status: Fixed

File(s) affected: `contracts/amm/Sigma.sol`, `contracts/interfaces/ISigma.sol`

Description: The `ISigma` interface defines two functions: `getPutSigma` and `getCallSigma`. However, in `Sigma.sol`, only `getPutSigma` is implemented. Please note that `OptionAMMPool` uses both functions, meaning that a transaction that makes a call to `getCallSigma` will throw.

Scope: audit iteration 1 (143d9ee)

Recommendation: Implement `getCallSigma` or remove all functions related to call option if it is not intended to be used in this version.

Update: Both functions are now defined.

QSP-14 Functions `BlackScholes.getCallPrice`and `getPutPrice` Do Not Match Formulation

Severity: Medium Risk

Status: Acknowledged

File(s) affected: `contracts/amm/BlackScholes.sol`

Description: The implementation of `getCallPrice` does not match the expected mathematical formula, as given by:

```
C =  St * N(d1) - K * e^(-r * t) * N(d2)
```

The current implementation lacks the `e^(-r * t)` multiplication term. A similar issue occurs with `getPutPrice` and the corresponding expected formula.

Scope: audit iteration 1 (143d9ee)

Recommendation: Fix the implementation to match the corresponding mathematical Black-Scholes formulae.

QSP-15 Incorrect Fee Calculation

Severity: Medium Risk

Status: Fixed

File(s) affected: `contracts/amm/FeePool.sol`

Description: `FeePool._getCollectable` computes the given amount minus the fees. Hence, it returns the net amount. However, `FeePool.collect` invokes `_getCollectable`, using its return value as the fees to be collected. Following the comments in `FeePool.collect`, the function should "*calculate and collect the fees from an amount*". Instead, it is collecting the net amount, which is incorrect. Hence, more fees are being collected than it should.

Scope: audit iteration 1 (143d9ee)

Recommendation: Enhance the existing documentation to reflect the actual intended behavior, making sure it matches the code. In case the current comments are incorrect, proceed to fix the implementation accordingly.

Update: Fee calculation has been fixed.

QSP-16 Incorrect Pricing Model Usage

Severity: Medium Risk

Status: Fixed

File(s) affected: `contracts/options/OptionAMMFactory.sol`, `contracts/options/OptionAMMPool.sol`

Description: Following Pods documentation, the Black-Scholes formula "*applies only to European options (and to American calls on non-dividend paying assets)*". However, the `OptionAMMFactory` and `OptionAMMPool` contracts allow the creation of an AMM for any option type, including American put and call options. The Pods team does acknowledge the latter in their Gitbook, warning users accordingly. We note, however, that allowing to have an AMM for any option type could lead to user funds being lost in the case of American options, which could negatively impact the project.

Scope: audit iteration 2 (bbccd8e)

Recommendation: Although the existing documentation does warn users against listing American options, we recommend adding a requirement statement to `OptionAMMFactory` and `OptionAMMPool` constructors preventing such a listing from ever occurring.

QSP-17 Potential Stale Oracle Price

Severity: Low Risk

Status: Fixed

File(s) affected: `contracts/oracle/PriceProvider.sol`

Description: `PriceProvider.getAssetPrice` calls `ChainlinkPriceFeed.getLatestPrice`; however `getLatestPrice` does not take into consideration if the price is stale, meaning that the latest returned round can be obsolete. This occurs due to the fact that Chainlink aggregators relies on external nodes to be updated once a request is submitted by one of the sponsors.

Scope: audit iteration 1 (143d9ee)

Recommendation: The value returned by the Chainlink aggregator must be validated; `updatedAt` should be verified, instead of being ignored.

QSP-18 Integer Overflow / Underflow

Severity: Low Risk

Status: Fixed

File(s) affected: `contracts/options/PodOption.sol`

Description: A potential integer overflow could occur in `PodOption` (L77), as there are no upper bounds on `_expiration` and `_exerciseWindowSize`.

Scope: audit iteration 1 (143d9ee)

Recommendation: Either limit the upper values of `_expiration` and `_exerciseWindowSize` s.t. one can ensure an integer overflow can never occur, or if left unbounded, use SafeMath's `add()` function.

Update: The issue has been fixed by using SafeMath's operations.

QSP-19 The `decimals` Function Is Optional In The ERC20 Standard

Severity: Low Risk

Status: Fixed

File(s) affected: `contracts/options/PodOption.sol`

Description: Some code assumes the token it is handling implements the `decimals` function, which may not be the case. Hence, the code could fail for certain tokens.

Scope: audit iteration 1 (143d9ee)

Recommendation: Have a default value or document this requirement.

Update: This has been resolved by means of a try-like operation that fails in case the `decimals` function is not available.

QSP-20 Option Exchange Liquidity Addition Misses Some ERC20 Calls

Severity: Low Risk

Status: Fixed

File(s) affected: `contracts/exchanges/OptionExchange.sol`

Description: When adding liquidity by means of calling `OptionExchange.addLiquidity`, the token address is not used in a `transferFrom(msg.sender, ...)` call; also, `tokenAmount` is not approved to be spent by the exchange.

Scope: audit iteration 1 (143d9ee)

Recommendation: Add the required ERC20 function calls.

Update: The recommended calls were added to the code.

QSP-21 AMM Provider Liquidity Addition Misses Some ERC20 Calls

Severity: Low Risk

Status: Fixed

File(s) affected: `contracts/exchanges/AMMProvider.sol`

Description: In `AMMProvider.addLiquidity` the A and B tokens amounts are not approved to be spent by the pool; consequently, all calls to that function will throw.

Scope: audit iteration 1 (143d9ee)

Recommendation: Add the required ERC20 function calls.

Update: Transfer uses the corresponding address of token A and token B; both must be approved prior to calling `addLiquidity` (already the case in `OptionExchange.sol`).

QSP-22 Input Validation Missing (1)

Severity: Low Risk

Status: Fixed

File(s) affected: `contracts/options/PodOption.sol`, `contracts/amm/AMM.sol`

Description: In the constructor of the `PodOption` contract, input parameters are not validated. Specifically:

- No check that `_underlyingAsset` and `strikeAsset` are different from `0x0` (fixed)
- No check that `_underlyingAsset` and `strikeAsset` are different from each other (fixed)
- No check that `_underlyingAsset` and `strikeAsset` are indeed addresses of a contract (fixed)
- No check that `_expiration` is greater than `block.timestamp`(fixed)
- No check that `_exerciseWindowSize` is greater than zero (fixed)
- No check that `_strikePrice` is greater than zero (fixed)
- No check on potential limits (if any) on the maximum allowed expiration and exercise window values. If there are no limits, then the code on L77 is subject to integer overflow (fixed).

In `AMM.sol`:

- No check that `_tokenA` and `tokenB` are different from `0x0` (fixed)
- No check that `_tokenA` and `tokenB` are indeed contracts (fixed)

Lacking sanity checks on input parameters could lead to incorrect pod contracts being created.

Scope: audit iteration 1 (143d9ee)

Recommendation: Add `require` statements as a means to sanity check the given input parameters.

Update: Corresponding require statements have been added to sanitize the aforementioned inputs.

QSP-23 Input Validation Missing (2)

Severity: Low Risk

Status: Unresolved

File(s) affected: `contracts/options/PodOption.sol`, `contracts/options/WPodOption.sol`, `contracts/amm/AMM.sol`, `contracts/amm/BlackSholes.sol`, `contracts/amm/FeePool.sol`, `contracts/amm/OptionAMMPool.sol`, `contracts/amm/Sigma.sol`, `contracts/oracle/ChainlinkPriceFeed.sol`, `contracts-v2/contracts/amm/OptionAMMFactory.sol`

Description: Consider checking that:

- In `NormalizedDistribution.sol`, the check in L334 should be that decimals >= 4 (fixed)
- `owner != address(0)` in `PodCall.mint` and `WPodCall.mint`**(not fixed)**

- `_tokenA != _tokenB` in the constructor of `AMM`(fixed)
- The input amounts to the trade fuctions in contract `AMM` are non-zero **(partially fixed, we meant both amounts)**
- `owner != address(0)` in the trade functions in contract `AMM`(fixed)
- `normalDistribution != address(0)` in the constructor of contract `BlackScholes`(fixed)
- `token != address(0)` in the constructor of contract `FeePool`(fixed)
- `feeDecimals <= 77` and `feeValue <= uint256(10)**feeDecimals` in the constructor of contract `FeePool`(fixed)
- The 7 input addresses are non-zero in contract `OptionAMMPool`(fixed)
- The input addresses to `OptionAMMFactory.createPool` are non-zero (fixed)
- `blackScholes != address(0)` in the constructor of `Sigma`(fixed)
- `_source != address(0)` in the constructor of `ChainlinkPriceFeed` (fixed)
- `_factory != address(0)` in the constructor of `OptionExchange` (fixed)
- `token != address(0)` in `OptionExchange.mintAndSellOptions` (fixed)
- `token != address(0)` in `OptionExchange.mintAndAddLiquidity` (fixed)
- `token != address(0)` in `OptionExchange.buyExactOptions` (fixed)
- `token != address(0)` in `OptionExchange.buyOptionsWithExactTokens` (fixed).

Also, for the `AMM`: According to the GitBook (https://app.gitbook.com/@pods-finance-1/s/teste/v/master/options-amm-overview/optionamm/functions/remove-liquidity, Section 1.1), a price of zero isn't acceptable, but this check is missing, unlike in the image on the GitBook **(not fixed)**. Additionally, it might be appropriate to check that there are funds in the contract, i.e., that totalTokenA > 0 || totalTokenB > 0 **(not fixed)**.
Last, but not least, the functions `OptionExchange.mintAndSellOptions`, `OptionExchange.mintAndAddLiquidity`, `OptionExchange.buyExactOptions` and `OptionExchange.buyOptionsWithExactTokens` must check that token is the option token of the pool, but they don't (fixed).
Lacking sanity checks on input parameters could lead to incorrect pod contracts being created.

Scope: audit iteration 2 (bbccd8e)

Recommendation: Add `require` statements as a means to sanity check the given input parameters.

QSP-24 Unchecked ERC20 `approve` Return Value

Severity: Low Risk

Status: Fixed

File(s) affected: `contracts/exchange/OptionsExchange.sol`

Description: The ERC20 `approve` calls made in `OptionsExchange` have their return value ignored; tokens that return `false` to signal an error would operate as no error occurred.

Scope: audit iteration 2 (bbccd8e)

Recommendation: Wrap `approve` calls inside a `require` statement.

QSP-25 Integer Overflow / Underflow

Severity: Low Risk

Status: Unresolved

File(s) affected: `contracts/options/PodOption.sol`, `contracts/amm/NormalDistribution.sol`, `contracts/amm/BlackSholes/sol`, `contracts/amm/OptionAMMPool.sol`, `contracts/amm/Sigma.sol`

Description: Tokens that have more than 77 decimals will cause the `_strikeToTransfer` function to always perform the unchecked overflow in L299, 316, 320 of `PodOption.sol` (fixed).
Similarly, if the number of decimals passed to `NormalDistribution.getProbability` exceeds 76, an unchecked integer overflow will occur (fixed).
Also, `NormalDistribution.sol` has unchecked integer arithmetic in L335, which could overflow for large absolute values of `z` (fixed), and in L350, which would overflow when a is the minimum value of int256 (fixed).

In `BlackScholes.sol` L61, if, maybe due to rounding problems, pay becomes more than get, an unchecked overflow will occur (fixed).

The constructor of `OptionAMMPool` should check that `tokenBDecimals <= BS_RES_DECIMALS` for otherwise unchecked integer underflow could happen, e.g., in L403 **(not fixed)**.
Other places with unchecked integer overflow/underflow are:

- `BlackScholes.sol` L82, 83 **(not fixed, the overflow is in the unary minus operator)**
- `BlackScholes.sol` L120, 123, 130, 131 (fixed)
- `BlackScholes.sol` L123 (fixed)
- `BlackScholes.sol` L143, 147 (fixed)
- `BlackScholes.sol` L181, 188 (fixed)
- `OptionAMMPool.sol` L385, 386, 389, 393, 394, 397 (fixed)
- `OptionAMMPool.sol` L414 **(not fixed)**
- `OptionAMMPool.sol` L442,445 **(not fixed)**
- `Sigma.sol` L184, 185, 188, 192, 193, 196 (fixed)

Scope: audit iteration 2 (bbccd8e)

Recommendation: Consider checking that `decimals <= 77` in function `tryDecimals` of the `RequiredDecimals` contract and that `decimals <= 76` in `NormalDistribution.getProbability`.
Implement the fixes in the "Best Practices" section of this document regarding values being multiples of 100 in `NormalDistribution.sol` and insert a check in function `_mod` (which should be renamed `_abs`).

QSP-26 Use of Unassigned Value

Severity: Low Risk

Status: Fixed

File(s) affected: `contracts/options/PodOption.sol`

Description: The check in `require` statement in L86 always succeeds, as `_exerciseType` hasn't been assigned yet (it holds the default value of `EUROPEAN`).

Scope: audit iteration 2 (bbccd8e)

Recommendation: In L86, replace `_exerciseType` with `exerciseType`.

QSP-27 Incomplete OptionExchange Mint Implementation

Severity: Low Risk

Status: Fixed

File(s) affected: `contracts/exchange/OptionExchange.sol`

Description: `OptionExchange.mint` does not handle call options since it only approve strike asset and does not check the options type to approve the underlying asset in the case of a call option.

Scope: audit iteration 2 (bbccd8e)

Recommendation: Verify the option type and calculate the asset amount to transfer and approve accordingly.

QSP-28 Input Validation Missing (3)

Severity: Low Risk

Status: Unresolved

File(s) affected: `EmergencyStop.sol, ModuleStorage.sol`

Description: Consider making the following checks:

1. The address is not stopped in the `stop` method of contract `EmergencyStop`;
2. `module != address(0)` in `ModuleStorage._setModule`;
3. `!_addresses[name]` in `ModuleStorage._setModule`;
4. `target != address(0)` in `CapProvider.setCap`;
5. The option pool itself is not stopped in function `_emergencyStopCheck()` of contract `OptionAMMPool`;
6. `configurationManager != address(0)` in the constructor of `CappedOption`.

QSP-29 Documentation Is Not In Natspec Format

Severity: Informational

Status: Fixed

File(s) affected: `contracts/*`

Description: Solidity contracts can use a special form of comments to provide rich documentation for functions, return variables and more. This special form is named the Ethereum Natural Language Specification Format (NatSpec) - see Solidity's official documentation. According to the latter, "It is recommended that Solidity contracts are fully annotated using NatSpec for all public interfaces (everything in the ABI)". However, the existing contracts in Pod's audited contract are not in Natspec.

Scope: audit iteration 1 (143d9ee)

Recommendation: Document all contracts according to the NatSpec format.

Update: Public and external functions have been documented to a satisfactory level.

QSP-30 Copied Code Amongst Options

Severity: Informational

Status: Mitigated

File(s) affected: `contracts/options/*.sol`

Description: Many functions are duplicated across option contracts. Rather, we suggest removing duplicate code (e.g., child contract function overrides parent function with the same code), or by fostering further reuse by re-organizing your inheritance chain.

Scope: audit iteration 1 (143d9ee)

Recommendation: Share common code in supertype contracts; only override functions that will have a different behavior.

Update: While similar code still exists across options, they no longer are plain copies. Nonetheless, further refactorings are possible.

QSP-31 Fixidity Is A Clone

Severity: Informational

Status: Acknowledged

File(s) affected: `contracts/amm/lib/*`

Description: Files under the `lib` folder seem to be copied from Fixidity - see https://github.com/CementDAO/Fixidity/blob/master/contracts. Copying code from a repository requires developers to keep track of bugs in the original code, fetching patches, and merging them. Not only this is a time consuming task, but also error prone. If not done correctly or frequently enough, bugs in the original code could remain in the copied one.

Recommendation: Since `yarn` is being used for managing dependencies, add Fixidity as a source code dependency.

Update: The Pods development team attempted to add Fixidity as source code dependency, but it did not work out of the box. They decided to acknowlegde the issue and not fix it for the time being.

QSP-32 Tokens Are Assumed To Be Implemented With OpenZeppelin's `ERC20`

Severity: Informational

Status: Fixed

File(s) affected: `contracts/amm/AMM.sol`, `contracts/amm/FeePool.sol`, `contracts/amm/OptionAMMPool.sol`, `contracts/exchanges/AMMProvider.sol`, `contracts/exchanges/BalancerProvider.sol`, `contracts/exchanges/OptionExchange.sol`, `contracts/exchanges/UniswapV1Provider.sol`, `contracts/options/*.PodPut.sol`, `contracts/options/PodOption.sol`,

Description: When invoking a method of the `IERC20` interface, the code casts an address to `ERC20` instead of `IERC20`, which assumes it is a contract deriving from OpenZeppelin's `ERC20` contract. However, there is no indication the tokens manipulated by the code satisfy this requirement.

Scope: audit iteration 1 (143d9ee), audit iteration 2 (bbccd8e)

Recommendation: Replace all instances of `ERC20(...).` with `IERC20(...).`

Update: The issue has been fixed as per recommendation in the files that we initially pointed out in the first audit iteration. However, in the second audit iteration, the issue has been re-introduced in `contracts/options/WPodCall.sol` (a new file).

QSP-33 Unlocked Pragma

Severity: Informational

Status: Fixed

File(s) affected: `contracts/*`

Description: Every Solidity file specifies in the header a version number of the format `pragma solidity (^)0.6.*`. The caret (^) before the version number implies an unlocked pragma, meaning that the compiler will use the specified version *and above*, hence the term "unlocked".

Recommendation: For consistency and to prevent unexpected behavior in the future, it is recommended to remove the caret to lock the file onto a specific Solidity version.

Update: All files have been locked to the0.6.12 version.

QSP-34 Allowance Double-Spend Exploit

Severity: Informational

Status: Mitigated

File(s) affected: `contracts/options/*.sol`,

Description: As it presently is constructed, all options are vulnerable to the allowance double-spend exploit, as with any other ERC20 tokens. An example of an exploit goes as follows:

1. Alice allows Bob to transfer `N` amount of Alice's tokens (`N>0`) by calling the `approve()` method on `Token` smart contract (passing Bob's address and `N` as method arguments)
2. After some time, Alice decides to change from `N` to `M` (`M>0`) the number of Alice's tokens Bob is allowed to transfer, so she calls the `approve()` method again, this time passing Bob's address and `M` as method arguments
3. Bob notices Alice's second transaction before it was mined and quickly sends another transaction that calls the `transferFrom()` method to transfer `N` Alice's tokens somewhere
4. If Bob's transaction will be executed before Alice's transaction, then Bob will successfully transfer `N` Alice's tokens and will gain an ability to transfer another `M` tokens
5. Before Alice notices any irregularities, Bob calls `transferFrom()` method again, this time to transfer `M` Alice's tokens. The exploit (as described above) is mitigated through use of functions that increase/decrease the allowance relative to its current value, such as `increaseAllowance` and `decreaseAllowance`.

Pending community agreement on an ERC standard that would protect against this exploit, we recommend that developers of applications dependent on `approve()` / `transferFrom()` should keep in mind that they have to set allowance to 0 first and verify if it was used before setting the new value. Teams who decide to wait for such a standard should make these recommendations to app developers who work with their token contract.

Scope: audit iteration 1 (143d9ee)

Recommendation: As a means to mitigate the issue, provide functions to increase and decrease allowance (already the case).

QSP-35 Incorrect Handling ETH Deposits Made by Mistake

Severity: Informational

Status: Fixed

File(s) affected: `contracts/options/WPodCall.sol`, `contracts/options/WPodPut.sol`

Description: Both `WPodCall` and `WPodPut` implement `receive` to handle ETH deposits without executing any logic. Any ETH sent by mistake will be deposited to the contact balance without any implemented feature to withdraw it.

Scope: audit iteration 2 (bbccd8e)

Recommendation: Only allow WETH address to deposit ETH.

QSP-36 American Options Should Not Require An Exercise Window

Severity: Informational

Status: Fixed

File(s) affected: `contracts/options/PodOption.sol`

Description: Following the `exerciseWindow` modifier, American options do not depend on `exerciseWindowSize`. Hence, one would expect it to be zero when creating an American option. However, the `PodOption` constructor requires an `exerciseWindow` always greater than zero, which should only be the case for European options (see QSP-20, audit iteration 1).

Scope: audit iteration 2 (bbccd8e)

Recommendation: In the `PodOption`constructor, require `exerciseWindowSize` to be greater than zero if and only if the option type is European.

QSP-37 Wrong Variable and Method Names

Severity: Informational

Status: Unresolved

File(s) affected: `PodOption.sol, OptionAMMPool.sol`

Description: In commit `80116ebca5f2a369c15649cafcc96f3c75954591`, a new naming convention in file `PodOption.sol` appeared: `_endOfExerciseWindow` became `_startOfExerciseWindow` and the associated getters were also renamed. However, this naming convention is incorrect for American option series, since they depict an exercise window of size zero while the real exercise window starts at the beginning of the option series. The corresponding function calls in `OptionAMMPool.sol` became thus confusing.

QSP-38 Option Unmint Restrictions Can Be Avoided

Severity: Informational

Status: Acknowledged

File(s) affected: `contracts/options/PodCall.sol`, `contracts/options/PodPut.sol`

Description: In both `PodPut` and `PodCall` contracts, if the buyers execute their options before expiration (American options), the sellers still holding their options will get both underlying and strike assets (following how much was executed) when they use `unmint`. However, the implemented `unmint` logic can be worked around by the sellers if they are still in hold of their ITM (in the money) options. The sellers can execute their options and wait for the expiration to withdraw and minimize their losses.

Recommendation: It should be clearly noted that the implemented `unmint` logic can be avoided when the options are in the money.

QSP-39 `mAB` Could Be A Zero Multiplier

Severity: Undetermined

Status: Acknowledged

File(s) affected: `contracts/amm/AMM.sol`

Description: When users remove liquidity, their returns are adjusted by a multiplier. Specifically, the `mAB` multiplier defines how much B tokens can a user remove for each A token initially deposited by the same user. In `_getMultipliers`, if `mAA` is set as in L596 (say `mAA > 1` => there is more A tokens that what the pool owes), `mBB` is set to

```
totalTokenBWithPrecision.div(deamortizedTokenBBalance
```

in L602, and say `mBB > 1`. All indicators are healthy, as there are more tokens than debt. Still, in 607, `mAB` will be zero (this can be verified if one expand the whole equation).

Scope: audit iteration 2 (bbccd8e)

Recommendation: Verify whether the described behavior is indeed intended, and if so, document if properly in the code. If not, adjust the mathematical model and its implementation.

QSP-40 `wPodPut` and `waPodPut` May Not Have WETH As The Underlying Token

Severity: Undetermined

Status: Mitigated

File(s) affected: `contracts/options/wPodPut.sol`, `contracts/options/waPodPut.sol`

Description: From the Solidity code alone, there is no guarantee that the underlying token in `wPodPut`and `waPodPut` is indeed WETH, even if the former two contracts are created through the option factory. If the underlying token is expected to be WEth, but turns out to be a different token, then the result could be catastrophic (e.g., funds could be potentially locked) and undetermined.

Recommendation: Hardcode the WETH address in all contracts that refer to its address.

Update: The team guarantees the correctness of the WETH address through their deployment scripts. Having it parameterized allows one to inject different addresses according to the target network (e.g., mainnet vs testnet). The validation of deployment scripts, however, is outside the scope of this audit. Based on the provided explanation, we are marking this issue as mitigated.

QSP-41 Use of (Third-Party) Untested Function

Severity: Undetermined

Status: Fixed

File(s) affected: `contracts/lib/ExponentLib.sol`

Description: The `powerE` function in `ExponentLib.sol` (a code clone) hasn't been fully tested. See comments in L12-14.

Scope: audit iteration 2 (bbccd8e)

Update (`80116ebc`)**:** the affected file has been removed from the codebase.

Recommendation: Check if tests were introduced in newer commits on the original repository where `ExponentLib` was cloned from; if so, incorporate them. Also, create your own tests so as to cover untested parts of the original code.

QSP-42 Incorrect Multiplication Implementation

Severity: Undetermined

Status: Fixed

File(s) affected: `contracts/amm/BlackSholes.sol`

Description: The BlackSholes implementation provides its own versions of operations found in `SafeMath`, from OpenZeppelin. It is unclear why SafeMath was not used. Re-implementing things could lead to bugs. In fact, the implemented multiplication operation (`_mul`) fails when the first operand is zero - L169 results to a division by zero:

```
L.167.  function _mul(uint256 a, uint256 b) internal pure returns (uint256) {
L.168.     uint256 c = a * b;
L.169.     require(c / a == b, "Multiplication overflow"); <==
           ...
        }
```

Scope: audit iteration 2 (bbccd8e)

Recommendation: Following OpenZeppelin's implementation of SafeMath, add the following condition to the `_mul` function:

```
L167.  function _mul(uint256 a, uint256 b) internal pure returns (uint256) {
L168.     if (a == 0) return 0; <===
          ...
       }
```

QSP-43 Incorrect Shares Calculation

Severity: Undetermined

Status: Fixed

Description: The `_calculateShares` function calculates the denominator of `CALL` option as:

```
denominator = _underlyingReserves.add(
   _strikeReserves.mul((uint256(10)**_underlyingAssetDecimals).div(_strikePrice))
);
```

This could lead to a denominator of zero if the underlying asset has fewer decimal places (e.g., 3) than the `strikePrice` (say 2, as in 4758 => 47.58) - in the example, the integer division of `1000/4758` is zero. With a denominator of zero, L323 will throw an exception.

Scope: audit iteration 2 (bbccd8e)

Recommendation: To prevent truncation that could lead to a denominator of zero, consider first changing the calculation to:

```
denominator = _underlyingReserves.add(
   _strikeReserves.mul(uint256(10)**_underlyingAssetDecimals).div(_strikePrice)
);
```

It is worth mentioning that this change could still lead to the transaction being reverted - as the multiplication could throw an overflow.

QSP-44 Return Statements Found in Functions With No Expected Return

Severity: Undetermined

Status: Fixed

File(s) affected: `contracts/amm/OptionsAMMPool.sol`

Description: Functions `addLiquidity` and `removeLiquidity` do not have return values, as stated by the `IAMM` interface. Their implementation in `OptionAMMPool.sol`, however, are defined as `return _addLiquidity` and `return _removeLiquidity`, respectively. Moreover, `_addLiquidity` and `_removeLiquidity` do not return anything. While this appears to be working, developers should fix the code to convey what they expect to be the correct semantics in the contract, i.e., whether a return value should occur or not.

Scope: audit iteration 2 (bbccd8e)

Exploit Scenario: Fix the code s.t. no value is returned, or if a return is expected, change the signatures in `IAMM` and `OptionAMMPool.sol` to include a return value.

QSP-45 AMM Has Implementation Oversights

Severity: Undetermined

Status: Unresolved

File(s) affected: `contracts/amm/AMM.sol`

Description: There seems to be some coding oversights that require prompt attention from developers. For instance,

- In AMM.sol L483, it should be `_onTradeExactBOutput` **(not fixed)**
- The fourth parameter in AMM.sol L391 is incorrect (fixed)
- The last two parameters in AMM.sol L425 are swapped (fixed)
- The last two parameters in AMM.sol L459 are swapped (fixed)
- In `AMM.sol`, the emission of events `AddLiquidity` and `RemoveLiquidity` swapped the `amountOfStable` and `amountOfOptions` parameters. **(fixed by reordering event parameters: make sure the code that depended on this order has been updated accordingly)**

Scope: audit iteration 2 (bbccd8e)

Exploit Scenario: In addition to fixing the oversights herein listed, we urge the Pods team to do a thorough code review and make sure oversights are captured and eliminated.

QSP-46 Incorrect Implementation of the `uintToInt` Function

Severity: Undetermined

Status: Unresolved

File(s) affected: `BlackScholes.sol`

Description: The `uintToInt` function will revert when the argument is zero.

Recommendation: Evaulate if this behavior is intended. If it is, give a more descriptive name to the function. If it is not, change the comparison to a non-strict inequality.

Adherence to Specification

- The formulae in function `_getAvailableForRescueAmounts` do not seem to have any counterpart in the GitBook documentation. Furthermore, they seem odd given that `_balanceTokenB` is multiplied by `m.AB` (how much of B you get for A) and `_balanceTokenA` is multiplied by `m.BA` (how much of A you get for B). Please check that this code is working as intended.

- The multiplier formulae in the documentation (Section 1.3 of https://app.gitbook.com/@pods-finance-1/s/teste/v/master/options-amm-overview/optionamm/functions/remove-liquidity) differ from the ones in the code of function `_getMultipliers` of contract `AMM` in the if statements in `AMM.sol` L595,601, 606, 610. Make sure this difference is intended and that the behavior of this function is correct.

- Should function `OptionExchange.buyExactOptions` reset the pool's ERC20 allowance? Please clarify. See, for instance, `AMM.sol` L288.

- It is not clear from the documentation or the code comments whether the inequality in `PodOption.sol` L291 should be strict or not.

Code Documentation

Audit iteration 2 (bbccd8e)

- Both the GitBook documentation and the code comments are in dire need of careful proofreading by skilled English users. The errors are too many to list here.
- Throughout the file `BlackScholes.sol`, the values `d1` and `d2` are referred to as probabilities, even though they may not be between 0 and 1. The function `_getProbabilities` should also be renamed.
- `AMM.sol` L420,454: it's token B.
- `AMM.sol` L423,457: it's token A.
- `AMM.sol` L467: it's `_onTradeExactBOutput`.
- `AMM.sol` L485: it's "in" rather than "out".
- `AMM.sol` L692,693: it's not the original deposit but the current one.
- `AMM.sol` L696,697: they will be stored, not rescued.
- `OptionAMMPool.sol` L307: it's `amountAOut` rather than `amountBIn`.
- `OptionExchange.sol` L89: it's to the pool.
- In `IPodOption.sol` L24-25, it's the other way around. In L27-28, it's the collateral token, not necessarily the strike token.
- In `PodCall.sol` L22, `WPodCall.sol` L23, `PodPut.sol` L21 and `WPodPut.sol` L24, the claim that the ratio of tokens is "based on its initial position" seems incorrect, as it is based on the current ratio of underlying/strike tokens to collateral.
- In `IPodOption.sol` L32-37, `PodCall.sol` L117, `WPodCall.sol` L117 and `PodPut.sol` L115: the description is confusing because the options are sent to `msg.sender`, who is the clear owner of the options. However, the word "owner" also refers to who can withdraw the collateral later.
- Consider indicating which repositories and commits the following files were cloned from, together with which changes have been made to them.
  - `FixidityLib.sol`;
  - `ExponentLib.sol`;
  - `LogarithmLib.sol`.
- Give a reference to where the `BlackScholes._sqrt` implementation came from.
- `NormalDistribution.sol`: in English, the function name should be `_abs` rather than `_mod`; in L347, it should read "absolute value" rather than "module".
- `RequiredDecimals` L18: isn't it "RequiredDecimals" rather than "OptionalDecimals"?
- `PodCall.sol` L176, `WPodCall.sol` L173, `AMM.sol` L237, `NormalDistribution.sol` L331: bad comment.
- `NormalDistribution.sol` L334, `PriceProvider.sol` L66: bad error message.
- `AMM.sol` L389,423,457,491: it is not necessarily sent to the caller, but to `owner`.
- Revise the comments in the following files carefully:
  - `PodCallBuilder.sol`;
  - `WPodCallBuilder.sol`;
  - `PodPutBuilder.sol`;
  - `WPodPutBuilder.sol`.
- Replace "UNIX timestamp" by "seconds" in the code comments for the five option builders.
- `PodPut.sol` L36: should be "sell" rather than "buy".
- `PodCall.sol` L55, `WPodCall.sol` L56, `PodPut.sol` L55, `WPodPut.sol` L57: if the option type is American, they may receive a mix of underlying and strike asset.
- `WPodCall.sol` L12: the token pair is not arbitrary; does the option really need to be European? Likewise, in `WPodPut.sol` L12, does the option really need to be American?
- `AMM.sol` L40-41,43-44: this doesn't make much sense; consider rewriting it.
- `WPodPut.sol` L44: it should be ETH rather than DAI.
- `AMM.sol` L134: the name of the mapping should be `userSnapshots`.
- `WPodPut.sol` L54: the function name is `unmint` rather than `burn`.
- `PodCall.sol` L183: "WPodPut" should be "PodCall" instead.
- `PodCall.sol` L236, `WPodCall.sol` L228, `PodPut.sol` L235: it's only after series expiration for American options; European options can be withdrawn only after the end of the exercise window.
- `PodPut.sol` L182: this error message does not follow the standard set by the others.
- `PodOption.sol` L82: consider replacing "in a future timestamp" by "in the future".
- Be careful abbreviating "Black Scholes" as "BS" as that also means "bullshit".
- Make sure the APACHE license of the cloned files has no encumbrances that the MIT license doesn't. Otherwise check the claims about the MIT license in the

- documentation.

Update (80116ebc): There are new documentation issues:

- CapProvider.sol L8: wrong title.
- Document that a value of zero for the cap in contracts CappedOption and CappedPool mean there is no cap.

Adherence to Best Practices

Audit iteration 1 (143d9ee)

- Lock dependencies in packages.json.
- There are instances of variable shadowing in the contracts. Please refer to the Slither's output, fixing all items pointed by the tool.
- In PodPut.sol (L166), aPodPut.sol (L90), waPodPut.sol (L137), and wPodPut.sol (L94), the error message says "Null amount", which is non-intuitive error message. We suggest changing the error message to "Amount should be greater than zero".
- In aPodPut.sol (L93, L145, L160), PodPut.sol (L111, L136, LL169), waPodPut.sol (L96, L111, L140), and wPodPut.sol (L97), the error message says ""Amount too low"", which is non-intuitive error message. We suggest changing the error message to "Amount should be greater than zero".
- In PodPut.sol, the _strikeToTransfer function can be simplified as follows. The expected strike to transfer corresponds to n = amount * strikePrice, a number with underlyingAssetDecimals + strikePriceDecimals decimal digits. Since the desired precision is strikePriceDecimals, one has to truncate n to strikePriceDecimals precision, which means removing the extra decimals added by underlyingAssetDecimals. As such, it suffices to divide n by 10^underlyingAssetDecimals. This simplification saves gas and it is simple to understand.
- In PodPut.sol, the strikeToTransfer and _strikeToTransfer functions provide unnecessary redundancy. Rather, delete strikeToTransfer, renaming _strikeToTransfer to strikeToTransfer, adjusting its visibility to external. A similar situation occurs with the functions amountOfMintedOptions and _underlyingToTransfer.
- Many typos in the code. For instance: PodPut.sol L7 ("european" => "European"), contracts inheriting from PodPut ("american" => "American"), AMM.sol L112 (propotion` => proportion), among others. Consider using a spellchecker.
- In PodPut.sol, strictly speaking, the check on L166 is not required; if amount is zero, then amountStrikeToTransfer will also be zero and the condition on L169 would cause the transaction to revert anyways. If the intent is to save upfront gas in case of an incorrect amount, the require statement is indeed correct. In that latter case, we also suggest adding something similar to the mint function.
- The filenames aOptionFactory.sol, aPutOption.sol, waPodPut.sol, and wPodPut.sol and the contract declarations therein do not adhere to upper camel case as in other files. We recommend standardizing names; in this case, adhering to contract naming as upper camel case. Also, make better use of identifiers; prefixes such as w and a convey little to no information.
- In wPodPut.sol, the Received event is not indexed. We suggest indexing it by the sender's address.
- The result of transfer and transferFrom of IERC20 are ignored in files BalancerProvider.sol and UniswapV1Provider.sol. Instead of ignoring their values, wrap these calls in corresponding require statements.
- In file PodOption.sol: modifier beforeExerciseWindow should have been named something similar to beforeEndOfExerciseWindow, as the former name refers to the beginning of the window.
- Make the contracts inherit from the corresponding interfaces in interfaces/.
- In AMM.sol: the visibility of INITIAL_FIMP and FIMP_PRECISION is not explicit. Make it explicit.
- In AMM.sol, lines 64 and 65: Usage of uint32 for decimals. Use uint8 instead.
- In BlackScholes.sol: why not use SafeMath.mul rather than _mul?
- TODOS in file BalancerProvider.sol (line 105) and UniswapV1Provider.sol (line 136). Consider linking them to corresponding Github issues.
- In PodPut.sol, function mint: consider checking that owner != address(0).
- In PodOption.sol, lines 175 to 177, 186 to 188, 197 to 199, 208 to 210: why not just use require?

Audit iteration 2 (bbccd8e)

- Remove extra dot from the filename WPodCallBuilder..sol.
- Bad file name: WPodCallBuilder..sol.
- The probabilities returned by NormalDistribution.getProbability should arguably be of type uint256.
- Write a better (almost comprehensive) test set for NormalDistribution.getProbability.
- Don't redefine OptionType in contract Sigma.
- Because strikeAsset() is a trusted contract but msg.sender is not a trusted address, making the Address.sendValue call after the strike assets have been transferred in WPodCall.unmint and WPodCall.withdraw, even if slightly, decreases the odds of a reentrancy vulnerability.
- The following storage items can be marked immutable:
  - . All the storage items of contract PodOption, except for shares, mintedOptions and totalShares;
  - . tokenA, tokenADecimals, tokenB and tokenBDecimals of contract AMM;
  - . normalDistribution in contract BlackScholes;
  - . token in contract FeePool;
  - . priceProvider, priceMethod, impliedVolatility, feePoolA, feePoolB in contract OptionAMMPool.

- Consider marking `PodOption.getSellerWithdrawAmounts` `public` and using it in the `withdraw` function of `PodCall` and `WPodCall`.

- The storage items `_underlyingAssetDecimals` and `_strikePriceDecimals` are redundant. The former can be obtained by consulting the `ERC20` `decimals` method and the latter is the same as `_strikeAssetDecimals`.

- The storage items `weth` in contracts `WPodCall` and `WPodPut` are redundant, as they are always the same address as `_underlyingAsset`.

- Instead of the low-level `staticcall` Solidity primitive, consider using the `try` construct instead.

- Why not just assign `0` in `PodPut.sol` L251?

- The constant `BS_RES_DECIMALS` should arguably be in the interface `IBlackScholes`.

- Consider giving `usersSnapshot[msg.sender].fImp` a variable name in `AMM._removeLiquidity`.

- The variable names in L308,309 of `PodOption.sol` disrespects the variable naming convention of the project.

- The constant `PERCENT_PRECISION` has the same naming convention as `FIMP_PRECISION`, but they are very different because `FIMP_PRECISION` is a number of decimals while `PERCENT_PRECISION` is a denominator. Verify the naming conventions in place and make sure they are consistent throughout the code.

- The parameters of the following functions have a bad naming convention. Not only it is different from that of the constructor of `PodOption`, some names clash with those of storage items.

  - The constructor of `PodCall`
  - The constructor of `PodPut`
  - The constructor of `WPodCall`
  - The constructor of `WPodPut`
  - `IOptionBuilder.buildOption`
  - `PodCallBuilder.buildOption`
  - `WPodCallBuilder.buildOption`
  - `PodPutBuilder.buildOption`
  - `WPodPutBuilder.buildOption`
  - All functions in file `OptionFactory.sol`.

- It's better to call the internal version of the functions in L308, 309 of `PodOption.sol`.

- The following return statements are not necessary:

  - `PodOption.sol` L132, 324
  - `OptionFactory.sol` L113
  - `AMM.sol` L528
  - `AMM.sol` L54
  - `AMM.sol` L567
  - `AMM.sol` L615
  - `AMM.sol` L657
  - `AMM.sol` L718
  - `BlackScholes.sol` L133.

- There is an extraneous pair of parentheses in L316 of `PodOption.sol`.

- The `using SafeMath for uint8` statements in `PodOption.sol` L29 and `PodCall.sol` L72 are not necessary.

- Consider using the same code to perform the exponentiations in L299, 316, 320 of `PodOption.sol`.

- Consider giving the enumerals of enums `IPodOption.OptionType` and `IPodOption.ExerciseType` explicit values because those are expected according to the code comments in the option builders (e.g., `OptionFactory.sol` L51, 52.

- The following functions should be marked `external`:

  - `PodOption.optionType`
  - `PodOption.exerciseType`
  - `PodOption.underlyingAsset`
  - `PodOption.underlyingAssetDecimals`
  - `PodOption.strikeAsset`
  - `PodOption.strikeAssetDecimals`
  - `PodOption.strikePrice`
  - `PodOption.strikePriceDecimals`

-
    - `PodOption.expiration`
    - `PodOption.endOfExerciseWindow`
    - `PodCallBuilder.buildOption`
    - `WPodCallBuilder.buildOption`
    - `PodPutBuilder.buildOption`
    - `WPodPutBuilder.buildOption`.

- On our last audit with Pods Finance, we mentioned that it is a good idea for contracts to receive fractional funds rounded up and send fractional funds rounded down. We also described a scenario in which violating this principle could result in a loss of funds. While that scenario is no longer valid for the current code, we believe this is a principle worth following.

Update (`80116ebc`): There are new issues:

- Instead of relying on central contracts for the emergency stop and the cap, you could have made the contracts using this feature inherit from a simpler base contract. The `Pausable` contract from OpenZeppelin could have been used instead of the `EmergencyStop` contract.

- Since they do not hold addresses, the mappings `_addresses` in contracts `EmergencyStop` and `CapProvider` could use better variable names.

- The following functions should be marked `external`: `ModuleStorage.getModule`.

- Since `PodOption` inherits from OpenZeppelin's `ERC20` contract, you might wish to inherit `CappedOption` from that as well so there wouldn't be a need for an external call at `CappedOption.sol` L30.

Test Results

Test Suite Results

Test suite seems extensive, with 477 tests cases (all passing).

```
AMM.sol - TKN-A/TKN-B

    ✓ should not be able to trade in zero-address behalf (176ms)

    ✓ should not be able to trade inputting zero-amount (135ms)

   Constructor/Initialization checks

      ✓ should have correct initial parameters

      ✓ should not allow tokens with 0x0 address (62ms)

      ✓ should not allow tokens that are not contracts (64ms)

      ✓ should not allow the same token as tokenA and tokenB (42ms)

   Add Liquidity

      ✓ should revert if user dont supply liquidity of both assets

      ✓ should revert if user ask more assets than the user s balance

      ✓ should match balances accordingly (85ms)

      ✓ should not be able to add liquidity in zero-address behalf (46ms)

   Remove Liquidity

      ✓ should remove liquidity completely (127ms)

      ✓ should remove liquidity completely even if some address sent tokens directly to the contract before the initial liquidity (151ms)

      ✓ should show the position of users that did not add liquidity (90ms)

   Scenario group APR - Add Liquidity / Price change / Remove Liquidity

      ✓ should match balances accordingly - 3 adds / 1 price change / 3 remove un order (289ms)

      ✓ should match balances accordingly - 3 adds / 1 price change higher / 3 remove unorder (331ms)

      ✓ should match balances accordingly - 3 adds / 1 price change lower / 3 remove un order (307ms)

   Scenario group ATR - Add Liquidity / Trade / Remove Liquidity

      ✓ should match balances accordingly - 3 adds / 1 trade(buy) / 3 remove un-order (333ms)

      ✓ Sum of initial and final value should be equal - 3 adds / 1 trade(sell) / 3 remove un-order

   Scenario group ATPR - Add Liquidity / Trade / Price change / Remove Liquidity

      ✓ Impermanent Loss should be equal between participants - 3 adds / 1 trade(buy) / 1 price(up) / 3 removed un-order (361ms)

      ✓ Impermanent Loss should be equal between participants - 3 adds / 1 trade(buy) / 1 price(down) / 3 removed un-order (320ms)

      ✓ Impermanent Loss should be equal between participants - 3 adds / 1 trade(buy) / 1 price(up) / 3 removed un-order

      ✓ Impermanent Loss should be equal between participants - 3 adds / 1 trade(sell) / 1 price(down) / 3 removed un-order

      ✓ Impermanent Loss should be equal between participants - 3 adds / 1 trade(sell) / 1 price(up) / 3 removed un-order

   Re-Add Liquidity - The sum of withdraws in value for a combined deposit should be equal to a two separate ones

      ✓ APR (692ms)


  BlackScholes

    ✓ cannot create a pool with a zero-address normalDistribution

    ✓ should revert if number multiplication overflow

    ✓ should revert if multInt overflows (76ms)

    ✓ should revert if casting uint to int overflow (65ms)

   Put price: 428800000000000000

   scenario.expectedPrice: 399197219100000000

    ✓ Calculates the put price correctly (97ms)

   Put price: 1253750000000000000000

   scenario.expectedPrice: 1275126573000000000000

    ✓ Calculates the put price correctly (89ms)

   Put price: 0

   scenario.expectedPrice: 0

    ✓ Calculates the put price correctly (86ms)

Call price: 3997510000000000000000
```

```
      ✓ Calculates the call price correctly (91ms)
Call price: 4314200000000000000
      ✓ Calculates the call price correctly (92ms)
Call price: 0
      ✓ Calculates the call price correctly (86ms)

  FeePool
      ✓ cannot charge more than 100% fees
      ✓ cannot create a pool with a zero-address token
    Fee parameters
        ✓ sets the contract with initial params
        ✓ updates the contract parameters
    Fee collection
        ✓ calculates the fee correctly
    Fee shares
        ✓ triggers the events when minting and withdrawing (179ms)
        ✓ should be able to withdraw all fees if its the only share owner (161ms)
        ✓ should mint shares proportionally to their participation (218ms)
        ✓ should not allow to withdraw without enough share balance

  NormalDistribution
      ✓ fails when the decimals is less than 4
      ✓ gets cached normal distribution
      ✓ gets negative normal distribution
      ✓ should revert if decimals input probabilty is higher than 76
      ✓ gets concentrated probabilities (60ms)

  OptionAMMFactory
      ✓ should create new pool (55ms)
      ✓ should not create the same pool twice (43ms)
      ✓ return an existent pool (49ms)

  OptionAMMPool.sol - WBTC/USDC
    Constructor/Initialization checks
        ✓ should have correct option data (strikePrice, expiration, strikeAsset) (39ms)
        ✓ should return spotPrice accordingly
        ✓ should not allow trade after option expiration
        ✓ should revert when trying to deploy a Pool with strikeAsset decimals > BS_RES_DECIMALS (607ms)
        ✓ should revert when trying to deploy a Pool with tokenB decimals > BS_RES_DECIMALS (552ms)
        ✓ should not allow add liquidity after option expiration
        ✓ should not create a pool with fee pools that are non-contracts
        ✓ should not create a pool with American options (522ms)
    Reading functions
        ✓ should return the ABPrice (101ms)
    Add Liquidity
        ✓ should revert if user dont supply liquidity of both assets
        ✓ should revert if user ask more assets to it has in balance (62ms)
        ✓ should revert if user do not approved one of assets to be spent by OptionAMMPool (99ms)
        ✓ should not be able to add more liquidity than the cap (43ms)
        ✓ should revert if any dependency contract is stopped (149ms)
        ✓ should revert if add liquidity when the option price is zero (977ms)
    Remove Liquidity
        ✓ should remove all amount after simple addition (297ms)
        ✓ should revert if any dependency contract is stopped (179ms)
        ✓ should remove liquidity when option price is rounded to zero (470ms)
        ✓ should remove liquidity after expiration (368ms)
        ✓ should remove liquidity single-sided (451ms)
    Price too low
        ✓ should revert transactions if the option price is too low (1209ms)
    tradeExactAOutput
        ✓ should match values accordingly (539ms)
        ✓ should revert if any dependency contract is stopped (281ms)
    tradeExactAInput
        ✓ should match values accordingly (650ms)
        ✓ should revert if any dependency contract is stopped (297ms)
    tradeExactBOutput
        ✓ should match values accordingly (541ms)
        ✓ should revert if any dependency contract is stopped (293ms)
    tradeExactBInput
        ✓ should match values accordingly (372ms)
        ✓ should revert if any dependency contract is stopped (247ms)

  Sigma
      ✓ should return the assigned sigma
      ✓ cannot be deployed with a zero-address BlackScholes
    FindNextSigma
        ✓ Should return the next sigma value correctly
    FindNewSigma
newPrice: 1253750000000000000000
targetPrice: 1275126573000000000000
error 1%
        ✓ Should find the new sigma using pre-calculated initial guess (86ms)
newPrice: 1378900000000000000000
```

```
targetPrice: 1275126573000000000000
error 8%
      ✓ Should find the new sigma using initial guess > target price (1 iteration) (160ms)
newPrice: 1295750000000000000000
targetPrice: 1275126573000000000000
error 1%
      ✓ Should find the new sigma using initial guess > target price (n+1 iterations) (245ms)
newPrice: 1253750000000000000000
targetPrice: 1275126573000000000000
error 1%
      ✓ Should find the new sigma using initial guess < target price (1 iteration) (229ms)
newPrice: 1170400000000000000000
targetPrice: 1275126573000000000000
error 8%
      ✓ Should find the new sigma using initial guess < target price (n+1 iterations) (621ms)
      ✓ Should revert if initial sigma is 0

  EmergencyStop
    ✓ should return false to contracts that were not stopped
    ✓ signals the stoppage of contracts
    ✓ cannot resume not previously stopped contracts
    ✓ signals the resume of contracts

  ModuleStorage
    ✓ returns no address zero when module is not set
    ✓ sets a module address correctly

  OptionExchange
    ✓ assigns the factory address correctly
    ✓ cannot be deployed with a zero-address factory
    Mint
      ✓ mints the exact amount of options (72ms)
      ✓ mints both puts and call options (1136ms)
    Mint and Add Liquidity
      ✓ mints and add the options to the pool as liquidity (230ms)
      ✓ fails to add liquidity when the pool do not exist
    Mint and Sell
      ✓ mints and sells the exact amount of options (457ms)
      ✓ fails when the deadline has passed (208ms)
      ✓ fails to sell when the pool do not exist (214ms)
    Buy
      ✓ buys the exact amount of options (456ms)
      ✓ buy options with an exact amount of tokens (604ms)
      ✓ fails to buy when the pool do not exist (205ms)
      ✓ fails when the deadline has passed (195ms)

  Logarithm
    ✓ calculates ln for 1000000000000000000000000
    ✓ calculates ln for 2000000000000000000000000 (55ms)
    ✓ calculates ln for 2718281828459045000000000 (55ms)
    ✓ calculates ln for 500000000000000000000000 (74ms)
    ✓ calculates ln for 50000000000000000000000000 (60ms)

  OptionFactory
    ✓ Should create a new PodPut Option correctly and emit event (38ms)
    ✓ Should create a new WPodPut Option correctly and emit event
    ✓ Should create a new PodCall Option correctly and emit event
    ✓ Should create a new WPodCall Option correctly and emit event

  PodCall.sol - WETH/aUSDC
    Constructor/Initialization checks
      ✓ should have correct number of decimals for underlying and strike asset
      ✓ should have correct exercise type
      ✓ should have equal number of decimals podPut and underlyingAsset
      ✓ should have equal number of decimals StrikePrice and strikeAsset
      ✓ should not allow underlyingAsset/strikeAsset with 0x0 address (58ms)
      ✓ should not allow underlyingAsset/strikeAsset that are not contracts (49ms)
      ✓ should not allow for underlyingAsset and strikeAsset too be the same address
      ✓ should only allow expiration in the future
      ✓ should not allow strikePrice lesser than or equal 0
      ✓ should not allow exercise windows shorter than 24 hours if EUROPEAN exercise type
      ✓ should not allow exercise windows equal to 0 if AMERICAN exercise type
    Minting options
      ✓ should revert if user dont have enough collateral (38ms)
      ✓ should revert if user do not approve collateral to be spent by PodCall (43ms)
      ✓ should revert if user tries to mint 0 options
      ✓ should mint, and have right number when checking for users balances (61ms)
      ✓ should mint, increase senders option balance and decrease sender underlying balance (60ms)
      ✓ should not be able to mint more than the cap (78ms)
      ✓ should revert if user try to mint after expiration - European (44ms)
      ✓ should not mint for the zero address behalf
    Exercising options
      ✓ should revert if amount of options asked is zero
      ✓ should revert if user try to exercise before expiration (76ms)
      ✓ should revert if user have strike approved, but do not have enough options (42ms)
      ✓ should revert if sender not have enough strike balance (89ms)
      ✓ should revert if not approved strike balance (88ms)
      ✓ should revert if transfer fail from ERC20 (434ms)
      ✓ should exercise and have all final balances matched (176ms)
      ✓ should revert if user try to exercise after exercise window - European (89ms)
    Unminting options
      ✓ should revert if try to unmint without amount
      ✓ should revert if try to unmint amount higher than possible (52ms)
      ✓ should revert if transfer fail from ERC20 - underlying (371ms)
      ✓ should unmint, destroy sender option, reduce its balance and send underlying back (411ms)
      ✓ should unmint, destroy seller option, reduce its balance and send underlying back counting interests (Ma-Mb-UNa) (185ms)
      ✓ should unmint, destroy seller option, reduce its balance and send underlying back counting interests (Ma-Mb-UNa-UNb) (193ms)
      ✓ should revert if user try to unmint after expiration - European
      ✓ should revert if user try to unmint after start of exercise window - European
    Withdrawing options
      ✓ should revert if user try to withdraw before expiration
      ✓ should revert if user try to withdraw without balance after expiration
      ✓ should get withdraw amounts correctly in a mixed amount of Strike Asset and Underlying Asset (164ms)
      ✓ should withdraw Underlying Asset balance plus interest earned (116ms)
      ✓ should withdraw Underlying Asset balance plus interest earned proportional (Ma-Mb-Wa-Wb) (199ms)
      ✓ should withdraw mixed amount of Strike Asset and Underlying Asset (Ma-Mb-Ec-Wa-Wb) (269ms)
      ✓ should revert if transfer fail from ERC20 - underlying (390ms)
      ✓ should revert if transfer fail from ERC20 - strike (442ms)
    American Options
      ✓ should unmint american options partially (112ms)
      ✓ Unmint - should revert if underlyingToSend is 0 (option amount too low) (108ms)
      ✓ Unmint - should revert if strikeToSend is 0 (option amount too low) (109ms)
      ✓ Unmint - should revert if transfer fail from ERC20 (418ms)


  PodCall.sol - WBTC/aDAI
    Constructor/Initialization checks
      ✓ should have correct number of decimals for underlying and strike asset
      ✓ should have correct exercise type
      ✓ should have equal number of decimals podPut and underlyingAsset
      ✓ should have equal number of decimals StrikePrice and strikeAsset
      ✓ should not allow underlyingAsset/strikeAsset with 0x0 address (58ms)
      ✓ should not allow underlyingAsset/strikeAsset that are not contracts (50ms)
      ✓ should not allow for underlyingAsset and strikeAsset too be the same address
      ✓ should only allow expiration in the future
      ✓ should not allow strikePrice lesser than or equal 0
      ✓ should not allow exercise windows shorter than 24 hours if EUROPEAN exercise type
      ✓ should not allow exercise windows equal to 0 if AMERICAN exercise type
    Minting options
      ✓ should revert if user dont have enough collateral (45ms)
      ✓ should revert if user do not approve collateral to be spent by PodCall (40ms)
      ✓ should revert if user tries to mint 0 options
      ✓ should mint, and have right number when checking for users balances (63ms)
      ✓ should mint, increase senders option balance and decrease sender underlying balance (62ms)
      ✓ should not be able to mint more than the cap (76ms)
      ✓ should revert if user try to mint after expiration - European (48ms)
      ✓ should not mint for the zero address behalf
    Exercising options
      ✓ should revert if amount of options asked is zero
      ✓ should revert if user try to exercise before expiration (76ms)
      ✓ should revert if user have strike approved, but do not have enough options
      ✓ should revert if sender not have enough strike balance (93ms)
      ✓ should revert if not approved strike balance (87ms)
      ✓ should revert if transfer fail from ERC20 (409ms)
      ✓ should exercise and have all final balances matched (163ms)
      ✓ should revert if user try to exercise after exercise window - European (84ms)
    Unminting options
      ✓ should revert if try to unmint without amount
      ✓ should revert if try to unmint amount higher than possible (51ms)
      ✓ should revert if transfer fail from ERC20 - underlying (382ms)
      ✓ should unmint, destroy sender option, reduce its balance and send underlying back (133ms)
      ✓ should unmint, destroy seller option, reduce its balance and send underlying back counting interests (Ma-Mb-UNa) (179ms)
      ✓ should unmint, destroy seller option, reduce its balance and send underlying back counting interests (Ma-Mb-UNa-UNb) (182ms)
      ✓ should revert if user try to unmint after expiration - European
      ✓ should revert if user try to unmint after start of exercise window - European
    Withdrawing options
      ✓ should revert if user try to withdraw before expiration
      ✓ should revert if user try to withdraw without balance after expiration
      ✓ should get withdraw amounts correctly in a mixed amount of Strike Asset and Underlying Asset (158ms)
      ✓ should withdraw Underlying Asset balance plus interest earned (111ms)
      ✓ should withdraw Underlying Asset balance plus interest earned proportional (Ma-Mb-Wa-Wb) (188ms)
      ✓ should withdraw mixed amount of Strike Asset and Underlying Asset (Ma-Mb-Ec-Wa-Wb) (239ms)
      ✓ should revert if transfer fail from ERC20 - underlying (370ms)
      ✓ should revert if transfer fail from ERC20 - strike (383ms)
    American Options
      ✓ should unmint american options partially (105ms)
      ✓ Unmint - should revert if underlyingToSend is 0 (option amount too low) (93ms)
      ✓ Unmint - should revert if strikeToSend is 0 (option amount too low)
      ✓ Unmint - should revert if transfer fail from ERC20 (401ms)

  PodCallBuilder
    ✓ Should create a new PodPut Option correctly and not revert

  PodPut.sol - WETH/USDC
    Constructor/Initialization checks
      ✓ should have correct number of decimals for underlying and strike asset
      ✓ should have correct exercise type
      ✓ should have equal number of decimals podPut and underlyingAsset
      ✓ should have equal number of decimals StrikePrice and strikeAsset
      ✓ should not allow underlyingAsset/strikeAsset with 0x0 address (209ms)
      ✓ should not allow underlyingAsset/strikeAsset that are not contracts (57ms)
      ✓ should not allow for underlyingAsset and strikeAsset to be the same address
      ✓ should only allow expiration in the future
      ✓ should not allow strikePrice lesser than or equal 0
      ✓ should not allow exercise windows shorter than 24 hours if EUROPEAN option
      ✓ should not allow exercise window different than 0 if AMERICAN option
      ✓ should return right booleans if the option is expired or not
      ✓ should not allow underlyingAsset or strikeAsset decimals higher than 76 (155ms)
    Minting options
      ✓ should revert if user do not have enough collateral (43ms)
      ✓ should revert if user do not approve collateral to be spent by podPut (39ms)
      ✓ should revert if asked amount is too low (40ms)
      ✓ should revert if amount of options asked is zero
      ✓ should mint, and have right number when checking for users balances (64ms)
      ✓ should not be able to mint more than the cap (81ms)
      ✓ should mint, increase senders option balance and decrease sender strike balance (54ms)
      ✓ should mint rounding up the value to receive, and round down the value sent, in order to avoid locked funds - multiple users (308ms)
      ✓ should mint rounding up the value to receive, and round down the value sent, in order to avoid locked funds (380ms)
      ✓ should revert if user try to mint after expiration (38ms)
      ✓ should not mint for the zero address behalf
    Exercising options
      ✓ should revert if amount of options asked is zero
      ✓ should revert if transfer fail from ERC20 (385ms)
      ✓ should revert if user try to exercise before start of exercise window (96ms)
      ✓ should revert if user have underlying approved, but do not have enough options
      ✓ should revert if sender not have enough strike balance (119ms)
      ✓ should revert if not approved strike balance (110ms)
      ✓ should exercise and have all final balances matched (152ms)
      ✓ should revert if user try to exercise after exercise window (102ms)
    Unminting options
      ✓ should revert if try to unmint without amount
      ✓ should revert if try to unmint amount higher than possible (79ms)
      ✓ should revert if unmint amount is too low (94ms)
      ✓ should revert if transfer fail from ERC20 (377ms)
      ✓ should unmint, destroy sender option, reduce its balance and send strike back (150ms)
      ✓ should unmint, destroy seller option, reduce its balance and send strike back counting interests (Ma-Mb-UNa) (227ms)
      ✓ should unmint, destroy seller option, reduce its balance and send strike back counting interests (Ma-Mb-UNa-UNb) (232ms)
      ✓ should revert if user try to unmint after expiration
    Withdrawing options
      ✓ should revert if user try to withdraw before expiration
      ✓ should revert if user try to withdraw without balance after expiration
      ✓ should get withdraw amounts correctly in a mixed amount of Strike Asset and Underlying Asset (Ma-Mb-Ec-Wa-Wb) (202ms)
      ✓ should revert if transfer fail from ERC20 (379ms)
      ✓ should withdraw Strike Asset balance plus interest earned (132ms)
      ✓ should withdraw Strike Asset balance plus interest earned proportional (Ma-Mb-Wa-Wb) (251ms)
      ✓ should withdraw mixed amount of Strike Asset and Underlying Asset (Ma-Mb-Ec-Wa-Wb) (323ms)
      ✓ Withdraw - should revert if transfer fail from ERC20 (394ms)
    American Options
      ✓ should mint american options correctly (60ms)
      ✓ should unmint american options partially (104ms)
      ✓ should revert if trying to exercise after expiration
      ✓ should withdraw american options correctly (84ms)
      ✓ should revert if trying to withdraw before expiration (46ms)
      ✓ Unmint - should revert if transfer fail from ERC20 (402ms)
      ✓ should revert if unmint amount is too low - underlying

  PodPut.sol - WBTC/aDAI
    Constructor/Initialization checks
      ✓ should have correct number of decimals for underlying and strike asset
      ✓ should have correct exercise type
      ✓ should have equal number of decimals podPut and underlyingAsset
      ✓ should have equal number of decimals StrikePrice and strikeAsset
      ✓ should not allow underlyingAsset/strikeAsset with 0x0 address (59ms)
      ✓ should not allow underlyingAsset/strikeAsset that are not contracts (63ms)
      ✓ should not allow for underlyingAsset and strikeAsset to be the same address
      ✓ should only allow expiration in the future
      ✓ should not allow strikePrice lesser than or equal 0
      ✓ should not allow exercise windows shorter than 24 hours if EUROPEAN option
      ✓ should not allow exercise window different than 0 if AMERICAN option
      ✓ should return right booleans if the option is expired or not (41ms)
      ✓ should not allow underlyingAsset or strikeAsset decimals higher than 76 (155ms)
    Minting options
      ✓ should revert if user do not have enough collateral
      ✓ should revert if user do not approve collateral to be spent by podPut (38ms)
      ✓ should revert if asked amount is too low
      ✓ should revert if amount of options asked is zero
      ✓ should mint, and have right number when checking for users balances (58ms)
      ✓ should not be able to mint more than the cap (76ms)
      ✓ should mint, increase senders option balance and decrease sender strike balance (55ms)
      ✓ should mint rounding up the value to receive, and round down the value sent, in order to avoid locked funds - multiple users (304ms)
      ✓ should mint rounding up the value to receive, and round down the value sent, in order to avoid locked funds (408ms)
      ✓ should revert if user try to mint after expiration
      ✓ should not mint for the zero address behalf
    Exercising options
      ✓ should revert if amount of options asked is zero
      ✓ should revert if transfer fail from ERC20 (369ms)
      ✓ should revert if user try to exercise before start of exercise window (91ms)
      ✓ should revert if user have underlying approved, but do not have enough options
      ✓ should revert if sender not have enough strike balance (108ms)
      ✓ should revert if not approved strike balance (111ms)
      ✓ should exercise and have all final balances matched (143ms)
      ✓ should revert if user try to exercise after exercise window (97ms)
    Unminting options
      ✓ should revert if try to unmint without amount
      ✓ should revert if try to unmint amount higher than possible (74ms)
      ✓ should revert if unmint amount is too low (77ms)
      ✓ should revert if transfer fail from ERC20 (367ms)
      ✓ should unmint, destroy sender option, reduce its balance and send strike back (137ms)
      ✓ should unmint, destroy seller option, reduce its balance and send strike back counting interests (Ma-Mb-UNa) (214ms)
      ✓ should unmint, destroy seller option, reduce its balance and send strike back counting interests (Ma-Mb-UNa-UNb) (523ms)
      ✓ should revert if user try to unmint after expiration
    Withdrawing options
      ✓ should revert if user try to withdraw before expiration
      ✓ should revert if user try to withdraw without balance after expiration
      ✓ should get withdraw amounts correctly in a mixed amount of Strike Asset and Underlying Asset (Ma-Mb-Ec-Wa-Wb) (215ms)
      ✓ should revert if transfer fail from ERC20 (391ms)
      ✓ should withdraw Strike Asset balance plus interest earned (135ms)
      ✓ should withdraw Strike Asset balance plus interest earned proportional (Ma-Mb-Wa-Wb) (265ms)
      ✓ should withdraw mixed amount of Strike Asset and Underlying Asset (Ma-Mb-Ec-Wa-Wb) (331ms)
      ✓ Withdraw - should revert if transfer fail from ERC20 (404ms)
    American Options
      ✓ should mint american options correctly (72ms)
      ✓ should unmint american options partially (108ms)
      ✓ should revert if trying to exercise after expiration
      ✓ should withdraw american options correctly (81ms)
      ✓ should revert if trying to withdraw before expiration (47ms)
      ✓ Unmint - should revert if transfer fail from ERC20 (412ms)
      ✓ should revert if unmint amount is too low - underlying (119ms)


  PodPutBuilder
    ✓ Should create a new PodPut Option correctly and not revert


  WPodCall.sol - ETH/USDC
    Constructor/Initialization checks
      ✓ should have correct number of decimals for underlying and strike asset
      ✓ should have equal number of decimals PodCall and underlyingAsset
      ✓ should have equal number of decimals StrikePrice and strikeAsset
      ✓ should not be able to send ETH directly
    Minting options
      ✓ should revert if user send 0 value to mint function
      ✓ should mint, increase senders option balance and decrease sender ETH balance
      ✓ should not be able to mint more than the cap (51ms)
      ✓ should revert if user try to mint after expiration
      ✓ should revert if user try to mint after start of exercise window
      ✓ should not mint for the zero address behalf
    Exercising options
      ✓ should revert if user try to exercise before expiration (55ms)
      ✓ should revert if transfer fail from ERC20 (362ms)
      ✓ should revert if user have underlying enough (ETH), but do not have enough options
      ✓ should exercise and have all final balances matched (126ms)
      ✓ should revert if user try to exercise after exercise window closed (42ms)
      ✓ should not be able to exercise zero options
    Unminting options
      ✓ should revert if try to unmint without amount
      ✓ should revert if try to unmint amount higher than possible
      ✓ should unmint, destroy sender option, reduce its balance and send underlying back - European (117ms)
      ✓ should unmint, destroy seller option, reduce its balance and send strike back counting interests (Ma-Mb-UNa) (137ms)
      ✓ should not unmint if there is not enough options (42ms)
    Withdrawing options
      ✓ should revert if user try to withdraw before expiration
      ✓ should revert if user try to withdraw without balance after expiration
      ✓ should seller withdraw Underlying Asset balance (129ms)
      ✓ should withdraw Strike Asset balance plus interest earned proportional (Ma-Mb-Wa-Wb) (156ms)
      ✓ should withdraw mixed amount of Strike Asset and Underlying Asset (Ma-Mb-Ec-Wa-Wb) (200ms)
      ✓ should revert if transfer fail from ERC20 - strike (408ms)
    American Options
      ✓ should unmint american options partially (83ms)
      ✓ Unmint - should revert if underlyingToSend is 0 (option amount too low) (80ms)
      ✓ Unmint - should revert if strikeToSend is 0 (option amount too low) (87ms)
      ✓ Unmint - should revert if transfer fail from ERC20 (404ms)

  WPodCall.sol - ETH/DAI
    Constructor/Initialization checks
      ✓ should have correct number of decimals for underlying and strike asset
      ✓ should have equal number of decimals PodCall and underlyingAsset
      ✓ should have equal number of decimals StrikePrice and strikeAsset
      ✓ should not be able to send ETH directly
    Minting options
      ✓ should revert if user send 0 value to mint function
      ✓ should mint, increase senders option balance and decrease sender ETH balance
      ✓ should not be able to mint more than the cap (52ms)
      ✓ should revert if user try to mint after expiration
      ✓ should revert if user try to mint after start of exercise window
      ✓ should not mint for the zero address behalf
    Exercising options
      ✓ should revert if user try to exercise before expiration (51ms)
      ✓ should revert if transfer fail from ERC20 (333ms)
      ✓ should revert if user have underlying enough (ETH), but do not have enough options
      ✓ should exercise and have all final balances matched (121ms)
      ✓ should revert if user try to exercise after exercise window closed
      ✓ should not be able to exercise zero options
    Unminting options
      ✓ should revert if try to unmint without amount
      ✓ should revert if try to unmint amount higher than possible
      ✓ should unmint, destroy sender option, reduce its balance and send underlying back - European (109ms)
      ✓ should unmint, destroy seller option, reduce its balance and send strike back counting interests (Ma-Mb-UNa) (129ms)
      ✓ should not unmint if there is not enough options
    Withdrawing options
      ✓ should revert if user try to withdraw before expiration
      ✓ should revert if user try to withdraw without balance after expiration
      ✓ should seller withdraw Underlying Asset balance (116ms)
      ✓ should withdraw Strike Asset balance plus interest earned proportional (Ma-Mb-Wa-Wb) (152ms)
      ✓ should withdraw mixed amount of Strike Asset and Underlying Asset (Ma-Mb-Ec-Wa-Wb) (192ms)
      ✓ should revert if transfer fail from ERC20 - strike (757ms)
    American Options
      ✓ should unmint american options partially (81ms)
      ✓ Unmint - should revert if underlyingToSend is 0 (option amount too low) (80ms)
      ✓ Unmint - should revert if strikeToSend is 0 (option amount too low)
      ✓ Unmint - should revert if transfer fail from ERC20 (384ms)

  WPodCallBuilder
    ✓ Should create a new WPodPut Option correctly and not revert

  WPodPut.sol - ETH/USDC
    Constructor/Initialization checks
      ✓ should have correct number of decimals for underlying and strike asset
      ✓ should have equal number of decimals aPodPut and underlyingAsset
      ✓ should have equal number of decimals StrikePrice and strikeAsset
      ✓ should not be able to send ETH directly
    Minting options
      ✓ should revert if user dont have enough collateral
      ✓ should revert if user do not approve collateral to be spent by WPodPut (38ms)
      ✓ should revert if asked amount is too low (39ms)
      ✓ should mint, increase senders option balance and decrease sender strike balance (53ms)
      ✓ should not be able to mint more than the cap (56ms)
      ✓ should revert if user try to mint after expiration (48ms)
      ✓ should not mint for the zero address behalf
    Exercising options
      ✓ should revert if user try to exercise before start of exercise window (77ms)
      ✓ should revert if user have underlying enough, but dont have enough options
      ✓ should exercise and have all final balances matched (148ms)
      ✓ should revert if user try to exercise after exercise window closed (83ms)
      ✓ should not be able to exercise zero options
      ✓ should revert if transfer fail from ERC20 (375ms)
    Unminting options
      ✓ should revert if try to unmint without amount
      ✓ should revert if try to unmint amount higher than possible (71ms)
      ✓ should revert if unmint amount is too low (84ms)
      ✓ should unmint, destroy sender option, reduce its balance and send strike back (Without Exercise Scenario) (136ms)
      ✓ should unmint, destroy seller option, reduce its balance and send strike back counting interests (Ma-Mb-UNa) (238ms)
      ✓ should unmint, destroy seller option, reduce its balance and send strike back counting interests (Ma-Mb-UNa-UNb) (224ms)
      ✓ should not unmint if there is not enough options (65ms)
      ✓ should revert if user try to unmint after expiration
      ✓ should revert if user try to unmint after start of exercise window
      ✓ should revert if transfer fail from ERC20 (407ms)
    Withdrawing options
    ✓ should revert if user try to withdraw before expiration
    ✓ should revert if user try to withdraw without balance after expiration
    ✓ should withdraw Strike Asset balance plus interest earned (134ms)
    ✓ should withdraw Strike Asset balance plus interest earned proportional (Ma-Mb-Wa-Wb) (250ms)
    ✓ should withdraw mixed amount of Strike Asset and Underlying Asset (Ma-Mb-Ec-Wa-Wb) (305ms)
    ✓ Withdraw - should revert if transfer fail from ERC20 (419ms)
  American Options
    ✓ should unmint american options partially (83ms)
    ✓ should revert if unmint amount is too low - underlying (249ms)

WPodPut.sol - ETH/DAI
  Constructor/Initialization checks
    ✓ should have correct number of decimals for underlying and strike asset
    ✓ should have equal number of decimals aPodPut and underlyingAsset
    ✓ should have equal number of decimals StrikePrice and strikeAsset
    ✓ should not be able to send ETH directly
  Minting options
    ✓ should revert if user dont have enough collateral (39ms)
    ✓ should revert if user do not approve collateral to be spent by WPodPut
    ✓ should revert if asked amount is too low
    ✓ should mint, increase senders option balance and decrease sender strike balance (54ms)
    ✓ should not be able to mint more than the cap (52ms)
    ✓ should revert if user try to mint after expiration (48ms)
    ✓ should not mint for the zero address behalf
  Exercising options
    ✓ should revert if user try to exercise before start of exercise window (74ms)
    ✓ should revert if user have underlying enough, but dont have enough options
    ✓ should exercise and have all final balances matched (144ms)
    ✓ should revert if user try to exercise after exercise window closed (83ms)
    ✓ should not be able to exercise zero options
    ✓ should revert if transfer fail from ERC20 (355ms)
  Unminting options
    ✓ should revert if try to unmint without amount
    ✓ should revert if try to unmint amount higher than possible (65ms)
    ✓ should revert if unmint amount is too low
    ✓ should unmint, destroy sender option, reduce its balance and send strike back (Without Exercise Scenario) (139ms)
    ✓ should unmint, destroy seller option, reduce its balance and send strike back counting interests (Ma-Mb-UNa) (228ms)
    ✓ should unmint, destroy seller option, reduce its balance and send strike back counting interests (Ma-Mb-UNa-UNb) (204ms)
    ✓ should not unmint if there is not enough options (68ms)
    ✓ should revert if user try to unmint after expiration
    ✓ should revert if user try to unmint after start of exercise window
    ✓ should revert if transfer fail from ERC20 (387ms)
  Withdrawing options
    ✓ should revert if user try to withdraw before expiration
    ✓ should revert if user try to withdraw without balance after expiration
    ✓ should withdraw Strike Asset balance plus interest earned (125ms)
    ✓ should withdraw Strike Asset balance plus interest earned proportional (Ma-Mb-Wa-Wb) (239ms)
    ✓ should withdraw mixed amount of Strike Asset and Underlying Asset (Ma-Mb-Ec-Wa-Wb) (294ms)
    ✓ Withdraw - should revert if transfer fail from ERC20 (757ms)
  American Options
    ✓ should unmint american options partially (98ms)
    ✓ should revert if unmint amount is too low - underlying (277ms)

WPodPutBuilder
  ✓ Should create a new WPodPut Option correctly and not revert

ChainlinkPriceFeed
  ✓ cannot be deployed with a zero-address source
  ✓ returns the correct decimals
  ✓ returns the correct price
  ✓ returns the correct round data

PriceProvider
  PriceFeed management
    ✓ assigns the asset on construction correctly
    ✓ assigns the decimals on construction correctly
  setAssetFeeds
    ✓ should set a new feed (131ms)
    ✓ should revert if assets and feeds are with different lengths
    ✓ should revert if create with invalid PriceFeed
    ✓ should revert if Price Feed not started (125ms)
    ✓ should revert if stale price feed (134ms)
  removeAssetFeeds
    ✓ should remove a feed (134ms)
    ✓ should revert if try to remove a nonexistent feed
  getAssetPrice
    ✓ should fetches the price correctly
    ✓ should revert if stale price (298ms)
    ✓ should revert if fetches the price of nonexistent asset
    ✓ should revert if fetches the asset decimals of nonexistent asset
    ✓ should revert when the price is negative (236ms)
  latestRoundData
    ✓ fetches the round data
```

```
      ✓ should revert if fetches unregistered Feed

  477 passing (4m)
```

Code Coverage

The test suite provides overall high coverage on statement and line coverage metrics (both at ~98%); branch coverage has satisfactory coverage (~91%). However, branch coverage is particularly low on some critical files in the AMM module, such as the `AMM.sol` and the `OptionAMMFactory.sol` , both at 75% coverage. Consequently, there are portions of the execution flow in those two files that are not covered by the test suite, which could lead to uncovered bugs if not found by this audit.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
amm/	97.92	84.62	95.1	97.91	
AMM.sol	95.06	75	92	95.12	… 717,719,720
BlackScholes.sol	100	100	100	100	
FeePool.sol	100	91.67	100	100	
NormalDistribution.sol	100	100	100	100	
OptionAMMFactory.sol	100	75	100	100	
OptionAMMPool.sol	96.7	85.71	95	96.7	… 442,469,736
Sigma.sol	94	91.67	90	94	59,68,186
configuration/	100	100	100	100	
CapProvider.sol	100	100	100	100	
ConfigurationManager.sol	100	100	100	100	
EmergencyStop.sol	100	100	100	100	
ModuleStorage.sol	100	100	100	100	
exchange/	100	83.33	100	100	
OptionExchange.sol	100	83.33	100	100	
options/	100	99.38	100	100	
OptionFactory.sol	100	100	100	100	
PodCall.sol	100	100	100	100	
PodCallBuilder.sol	100	100	100	100	
PodOption.sol	100	97.62	100	100	
PodPut.sol	100	100	100	100	
PodPutBuilder.sol	100	100	100	100	
WPodCall.sol	100	100	100	100	
WPodCallBuilder..sol	100	100	100	100	
WPodPut.sol	100	100	100	100	
WPodPutBuilder.sol	100	100	100	100	
oracle/	100	100	100	100	
ChainlinkPriceFeed.sol	100	100	100	100	
PriceProvider.sol	100	100	100	100	
All files	**98.57**	**91.84**	**97.38**	**98.58**	

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

```
bd50f0ad70abaead03ababcfc6000350c19b9d237400e203ca866d07b44cf2e6 ./contracts/configuration/CapProvider.sol
844d9d7ca7bbe6df958ca60159558bd7cfbaae597337c9a0b37af4139a87a5c0 ./contracts/configuration/ConfigurationManager.sol
735702f8e6fb542c317fa71ba58486d7bf32bde18ca19f066ca67649e7cf5f8c ./contracts/configuration/EmergencyStop.sol
61ea0d42dabd9ef1ecf2768c6a1f6562b58ca2d7ac03cd4d5c8f2a403e9d8a35 ./contracts/configuration/ModuleStorage.sol
17e8189c12ebe95830199ca70993e3c896d1d742d9fec9e1157f5eb822648d21 ./contracts/lib/FixidityLib.sol
9a27a5a4c08eb42cd8142ed366c07139f7846d15dff57f91758a236b1d2a99b5 ./contracts/lib/LogarithmLib.sol
511a695221ac31c019cfba3f34385f921de93463c62d4c028e3c81178506bde0 ./contracts/lib/CappedOption.sol
```

```
44c84dd8b79b95e3ce567de5ae5aad16ba46e1a31b1e52b473dfadc0dfcf4ad7 ./contracts/lib/CappedPool.sol
9ccace4be4013950854ef94add66d2502dd3fb3acfc960579175c4f92ca8ac91 ./contracts/lib/RequiredDecimals.sol
85188f76f4d2be274f94ac68a3de224ae0bb2dd228f357e9b4b94a2e34b9f4bc ./contracts/exchange/OptionExchange.sol
52ff0b2faf3ffb6de58f123c71230acedb6f0c6e3d1e1b643a795004a8ebff4b ./contracts/oracle/ChainlinkPriceFeed.sol
c0f6a50da21357d14b12ddbcc1e24fd53cfba63b7a4c351dd03d183a9081b79d ./contracts/oracle/PriceProvider.sol
eb7774c53ae102f30eb85608c377bf801350e4c2782f9a6956d8468b6a0bc16b ./contracts/options/OptionFactory.sol
ff2d1b983fe93394caf5096929c89a5dbdc80ea8e06c860c838ca3c1c5270f1a ./contracts/options/PodCall.sol
b027429052a79d235214232a90f2141af00ef9ed846b053b8b328fbe49374a1f ./contracts/options/PodCallBuilder.sol
2ae29e8cae5734cfce52569753228bb01d6db3cf58f73d3d01d07e08a8270711 ./contracts/options/PodOption.sol
1651277c1cc3a38fa74aa18c527ee8c7e23984cce4bbe2c17442ffd0533277b5 ./contracts/options/PodPut.sol
80726cc25080ff534db534d5988420975dd9146dd1e126224087dc97c5d8e58d ./contracts/options/PodPutBuilder.sol
5371c24863ba137401b9de5d44bfa2f392ddbffa450261e752e15a6174bc1e4d ./contracts/options/WPodCall.sol
993711db6152234f1360d9b89cb0062b6f016be99df3b6571df9b5e3c495ab55 ./contracts/options/WPodCallBuilder..sol
77db4b84d69e1ae33cf06c29fb5a151133f25c47119e30d8a6714f6ad2314932 ./contracts/options/WPodPut.sol
1dba7668df33c98eba48103d9d20285f97aa5cf23b7d616f10015e38a1863885 ./contracts/options/WPodPutBuilder.sol
b68c9f2f8a555af12f09fc4a0af23592d133237128351e14b9364f040c825021 ./contracts/interfaces/IChainlinkPriceFeed.sol
fdded64943453392450a892524592c0aff0b5a044eea0bb729adeaf6a43e8de0 ./contracts/interfaces/IERC20Mintable.sol
42af245bc4c6bdb60ae0cb7eb334d9b73b6d20ed0956cf263d8fa25cdaab7f92 ./contracts/interfaces/IFeePool.sol
2afebbbbafcc9daff47a1ed059f1c3cbb4fb88ebb82ca31cd7592c72619e5070 ./contracts/interfaces/IOptionAMMPool.sol
65a3c76ff5c14c1b4918a7d0cb44a60ae6ed66234b103ab74699c002776ce04e ./contracts/interfaces/IPriceProvider.sol
e9d5df3021698438a7043e7c7fa426e0b17f65b556d60f3187c34b0bda383457 ./contracts/interfaces/IWETH.sol
4b71f085991e93c6323f1cef4db4d05979aedbaa4505e49b6a73b7932eeda15f ./contracts/interfaces/IAMM.sol
d5195f6599a6946b6f76189090090069f784edd74fc9ed887b809d60aac1150f ./contracts/interfaces/IBlackScholes.sol
0ad46c506085c1c6a7bd8fcc0cd22006c0012910a8509d54a89fbca59150e736 ./contracts/interfaces/ICapProvider.sol
67131903b05a6c7b5724dc4821132a11a5ea4a8686b6a4bd67df9e4eb7955208 ./contracts/interfaces/IConfigurationManager.sol
223aa7062e9774ce8322fec750bb27d4781f73a7b6a2ff470036281b56785730 ./contracts/interfaces/IEmergencyStop.sol
b8bfe0b8c598d0f49b17d6fba4f6f6b32568dc951d2e3247b8699cff0e5b9f34 ./contracts/interfaces/INormalDistribution.sol
85cd99a99e17417b02dd8cb5468bba0cdf41d0e035b772206b13669d028ea940 ./contracts/interfaces/IOptionAMMFactory.sol
4ae0ed32a0822480ded17de245ee705124fe0892313b916ad67e720af3d515ca ./contracts/interfaces/IOptionBuilder.sol
3dd9279b9bdb524987d6f668a8fcc61f06680d0715e9f64f2663df8a23881b3b ./contracts/interfaces/IPodOption.sol
eebd5d1ee3d489103f14a6cde25907ef049a86f269f30602d12affaceb6dade9 ./contracts/interfaces/IPriceFeed.sol
702de86776ec02dbf809f3e68aa4a78fd70c77c97a8b6568840686800b0fe8b0 ./contracts/interfaces/ISigma.sol
33d6e9cacb1ec5a8e92cd55ea39568029b8b17dc1831736c68480a01cc64b207 ./contracts/amm/AMM.sol
13e558b161e1737e0299e000b57b54baa92b19f7dd85ed4c18b6ffb2538d6c07 ./contracts/amm/BlackScholes.sol
10c017d01a82f2d806330b399aa613116adae19f5616d15569ec2f13d0983fad ./contracts/amm/FeePool.sol
f565d6f213f8b1501cb5a04a9c00897248836f427c6abb7215c6b2e2330b6c59 ./contracts/amm/NormalDistribution.sol
f7ff6e174ad7d37c56bc931ab4dd01374ee1ad22007e4a71929e98ca25fa739e ./contracts/amm/OptionAMMFactory.sol
796cb3c1eb20e3246d6f933754582ac25ecaa724c1964b490fb706ef0d89429d ./contracts/amm/OptionAMMPool.sol
8f835b1eabe8b9a9dc6f6e448a9cc3ff81bdd196443381133e9957237d71e01a ./contracts/amm/Sigma.sol
```

Tests

```
b9039725233633b2eb6ac1dc782a6d68fd57ab6876e6285794a4f013e52b3d3d ./test/lib/Logarithm.test.js
f7f1e1206bdd410545a45f410a62565a4b9fd44e3329c5ec3fb39a161f379a19 ./test/configuration/EmergencyStop.js
edcc0574c1767ba53fd54e44d0be27276bf4f5f89b8a990c7c4dee10a5c8e0b1 ./test/configuration/ModuleStorage.js
90ea7407f1da2295fba0574e96ccabedb6adc52720a78edd278b684d0d34168e ./test/oracle/ChainlinkPriceFeed.test.js
68a10470e9bdaa696269fcd5520822c12b5a267517a2959321f9d6708d8ee1e6 ./test/oracle/PriceProvider.test.js
c11825614ff86c7bb2f753ede57fc27ac62ef6881bfe64e060d1bae3025db656 ./test/options/OptionFactory.test.js
3e07a0c2425e44f4386d83587834b4f13d7fecf461602f797dc5412026157e63 ./test/options/PodCall.test.js
d2454b551c42ae39dcd44a269ec38d2a06216ce44b60dbd09b9ef21194de601e ./test/options/PodCallBuilder.test.js
0c6d2b2f8463de70267ebe5fc5d5060f264c5301b327da33174e3f5367c0c92c ./test/options/PodPut.test.js
31c86902a37fdf4495e87ff2cd7f25c573598d8fc65ce4bb5010f1382efea200 ./test/options/PodPutBuilder.test.js
16f8b6aaeeb114c6c712b55e46aa7274e82049ea700459e9fba955ce396cf359 ./test/options/WPodCall.test.js
3b19de9d78df51e5a8e0063f6a6419b5156b26764b192aac346ab19a4a528d00 ./test/options/WPodCallBuilder.test.js
2312dfd9464e10727dec647c9cfdf8003b9f123b11327559035de3db4cb9a305 ./test/options/WPodPut.test.js
961aa8688ac71ff83f41ef5cf957d07938508034a10bb829d36ebccf8c519ec6 ./test/options/WPodPutBuilder.test.js
08664cfe27ab54aaabf5d782db84de067b039329595488972 9b294e16e352890 ./test/exchange/OptionExchange.test.js
e6a80ed936b28f5c0c5132b8f6dd591af165cca3aa818f4dc7207cad95886bba ./test/amm/AMM.test.js
82461497b8842c2a110c6e9d820fd4a9a66eb6dae1ffd56b7d7878fbcea5cd51 ./test/amm/BlackScholes.test.js
d26742a2e3c32be89b1d9640115293943020f335eac01f0a5f0e8adbd2dd79b6 ./test/amm/FeePool.test.js
a45912d356ebb4a7bb2a4961ead656e8d9170d1b85c59f50be0115da4dee93d5 ./test/amm/NormalDistribution.test.js
074f41bd5a3894eb323491f144022ee82078d929f60e8b18ab8efd54f63bff5d ./test/amm/OptionAMMFactory.test.js
```

```
2e8480d4bf402162e8cd629ea877fdd3bdc71cfa7d444b755ed2d0b0c976416f ./test/amm/OptionAMMPool.test.js
4f2fa651127f4d3f82439f88d7b6ef60988b41c31f577a801f851b1a648ab6a3 ./test/amm/Sigma.test.js
9466b2ceba5bb305a61f02e9af5d7e350e72fbcb9f248609346ffe21c24142c5 ./test/util/getTimestamp.js
08a8ff7355e13820d19b3cf669929dd3b289561533216260577c44853de653c2 ./test/util/getTxCost.js
be6caf6d26076befe98919b85ab08dfe6de7906f4fc2951d656f029c70b27f73 ./test/util/linkBytecode.js
7b2f81034829865698c1cd5b7298b41c9ead0e9144b4dc48717c4001c20f024c ./test/util/forceExpiration.js
9f33e96921647ea9f4712bf6a822f827f813d4edc36476a94569316da930fa96 ./test/util/snapshot.js
c6c8c4c2540d581e879c1b6799c3e763828e55773489aaeb77f25358c3b60134 ./test/util/addLiquidity.js
9326543fef6cddff6d172b567861c91d742f998da8224cfcc947b28fc2a4e277 ./test/util/createBlackScholes.js
16da2e1ffdcc0107b5f3239ad314eec9dc9672db55943a9ddb9bf4e10db3bf8d ./test/util/createConfigurationManager.js
d84fd7fcc1def93ad57c9b96972e6bf34f9652f16513f869d604d795b3af7a6b ./test/util/createMockOption.js
c1acdaa9ae4edf24e9befcf927c615f403aa06489fcf69d3a6483ea479353bd8 ./test/util/createNewOption.js
a280303183f4c4e34c5e9915a42c2993b2a467cef574aa57080afcc7a296b613 ./test/util/createNewPool.js
34f46c5c90895adb54dbe009c9d78a7ef9932ecb3204829ac0020c9e701cdffd ./test/util/createOptionFactory.js
946934ee8a3f2937e7ea2ed1321b053b894656f3b9e825da732b3c9f3cb2da9c ./test/util/createPriceFeedMock.js
abcde97c6691200eb5a4c75ca3b8b2c3980e926300f9e5a6dfda504a708a735a ./test/util/forceStartOfExerciseWindow.js
7d0ff1d2cc1738266df8d6a710effe50d2c131c77a195d03468f4513426b083d ./test/util/getPriceProviderMock.js
3fea0876b206ace189f90e26f7a05201f9d37477a46af645c66f3e30abe3f7ae ./test/util/mintOptions.js
```

Changelog

- 2020-11-11 - Initial report (Audit iteration 1)
- 2021-02-10 - Re-audit (Audit iteration 2)
- 2021-02-23 - Re-audit (Audit iteration 2)

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.