Quantstamp Security Assessment Certificate

November 10th 2022 — Quantstamp Verified

Ensuro

This audit report was prepared by Quantstamp, the leader in blockchain security.

Executive Summary

Type	DeFi Insurance
Auditors	Danny Aksenov, Security Auditor Ibrahim Abouzied, Auditing Engineer Guillermo Escobero, Security Auditor Jennifer Wu, Auditing Engineer
Timeline	2022-09-26 through 2022-10-20
EVM	Paris
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	Product Docs White Paper
Documentation Quality	High
Test Quality	High

Source Code

Repository	Commit
ensuro/ensuro	23f4b9c initial audit
ensuro/aave-asset-manager	7b0cdd0 initial audit
ensuro/price-risk-module	b5f2668 initial audit
ensuro/flight-delay-risk-module	c05d9bb initial audit
ensuro/ensuro	b57c1f9 fixes
ensuro/flight-delay-risk-module	fae44b8 fixes
ensuro/price-risk-module	dd1d66f fixes
ensuro/aave-asset-manager	595188b fixes

Total Issues	**23** (16 Resolved)
High Risk Issues	**1** (1 Resolved)
Medium Risk Issues	**2** (1 Resolved)
Low Risk Issues	**12** (10 Resolved)

Informational Risk Issues	4 (2 Resolved)
Undetermined Risk Issues	4 (2 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).
Fixed	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

Ensuro is a blockchain protocol that provides capital coverage for insurance risks. Its main objective being to guarantee that the protocol will be able to cover the losses for the risks taken. Overall the code for the protocol is well-written, and provides exceptional documentation on how they structure their policies and payouts. The testing is also of a good quality, however it could benefit from additional branch coverage for several contracts. Outside of some upgradeability misconfigurations, the protocol does not appear to have any high severity issues. One thing to note is that some of the protocol's policy creation and resolution will be managed by off chain software, removing with it some on chain validation. We believe it is critical to have this software audited and well documented before deploying to production. Additionally, any new risk module should be audited as well, given their ability to receive payouts from the premium account holders once accepted in the policy pool. Unaudited risk modules can result in loss of funds for premium account holders.

ID	Description	Severity	Status
QSP-1	Upgradeable Misconfigurations	High	Fixed
QSP-2	Aave Price Oracle Using Deprecated Chainlink Method to Get Current Price	Medium	Fixed
QSP-3	Utilization Rate Could Exceed the Maximum on Withdrawals	Medium	Acknowledged
QSP-4	Upgradeable Contract Storage Gaps	Low	Fixed
QSP-5	Revocable Admin Roles	Low	Acknowledged
QSP-6	Missing Input Validation	Low	Mitigated
QSP-7	Loss of Precision Borrows More Funds than Necessary	Low	Fixed
QSP-8	Unlocked Pragma	Low	Fixed
QSP-9	Policies Can Be Created for Flights Already in Transit	Low	Fixed
QSP-10	Base Contracts Should Update Their Initialization Modifiers	Low	Fixed
QSP-11	Inaccurate Accounting if Deflationary Tokens Are Accepted	Low	Fixed
QSP-12	Checks-Effects-Interactions Pattern Violation	Low	Fixed
QSP-13	Risk of Killing Upgrades	Low	Mitigated
QSP-14	`PriceRiskModule` Susceptible to Oracle Manipulation and Flash Loan Attacks	Low	Acknowledged
QSP-15	Resolve Same Policy Multiple Times if `ERC777` Tokens Are Accepted	Low	Fixed
QSP-16	Missing Events to Signal State Changes	Informational	Fixed
QSP-17	Max Approval to Junior and Senior Tranches	Informational	Fixed
QSP-18	Profitability of the Protocol Depends on Accuracy of Risk Model	Informational	Acknowledged
QSP-19	Upgradability	Informational	Acknowledged
QSP-20	Loans Can Be Repayed when the Contract Is Paused	Undetermined	Fixed
QSP-21	May Not Have Adequate Funds to Deinvest	Undetermined	Acknowledged
QSP-22	Risk of Reentrancy when Creating a New Policy	Undetermined	Mitigated
QSP-23	`TrustfulRiskModule` Interacts with Off-Chain Software	Undetermined	Acknowledged

Quantstamp Audit Breakdown

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

DISCLAIMER:
If the final commit hash provided by the client contains features that are not within the scope of the audit or an associated fix review, those features are excluded from consideration in this report.

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 Upgradeable Misconfigurations

Severity: *High Risk*

Status: Fixed

File(s) affected: `AccessManager.sol`, `EToken.sol`, `LPManualWhitelist.sol`, `PolicyPool.sol`, `PolicyPoolComponent.sol`, `PremiumsAccount.sol`, `RiskModule.sol`, `TrustfulRiskModule.sol`, `PriceRiskModule.sol`, `FlightDelayRiskModule.sol`, `Reserve.sol`

Description: The implementation contracts are not guaranteed to have their initializer disabled. This makes it possible for a malicious actor to call the `initialize()` function on the implementation contracts. With some of the contracts containing delegate calls, it may be possible for the user to self-destruct an implementation contract, leaving the proxy contracts without an implementation contract to call on.
Additionally, the `initializer` modifier can only be called once even when using inheritance, so parent/base contracts should use the `onlyInitializing` modifier for their initialization functions. The following functions should have their modifier changed to `onlyInitializing`:

- `AccessManager.__AccessManager_init_unchained()`
- `EToken.__EToken_init_unchained()`
- `PolicyPool.__PolicyPool_init_unchained()`
- `PolicyPoolComponent.__PolicyPoolComponent_init()`
- `PremiumsAccount.__PremiumsAccount_init()`
- `PremiumsAccount.__PremiumsAccount_init_unchained()`
- `RiskModule.__RiskModule_init()`
- `RiskModule.__RiskModule_init_unchained()`

Recommendation: Add `_disableInitializers()` to all constructors of upgradeable contracts and make the recommended modifier changes to `onlyInitializing`.

Update: The team has addressed the recommended configurations regarding upgradability in the following two commits: commit1 , commit2.

QSP-2 Aave Price Oracle Using Deprecated Chainlink Method to Get Current Price

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `PriceRiskModule.sol`

Description: The AaveOracle contract uses the deprecated `ChainlinkAggregator.latestAnswer()` function to retrieve the current price of an asset. If `ChainlinkAggregator.latestAnswer()` fails to return a non-zero price, the AAVE oracle uses a fallback oracle. Although zero price is checked, stale price can still occur depending on the oracle used; loss of capital for users or the protocol is possible if stale price data is used to resolve Price Risk Module payouts. Note that Chainlink recommends using `ChainlinkAggregator.latestRoundData()` function instead of deprecated `latestAnswer()`.

Recommendation: Consider using `Chainlink.latestRoundData()` to obtain the current price of an asset. Remove hardcoded decimals and obtain the decimals from the oracle. Add a check to compare the current block timestamp and timestamp at which the price data is obtained at. Consider reverting the transaction if the price data obtained is determined to be stale.

Update: The team has updated the code to use Chainlink to get price data and added additional validations on the fetched price in the following commit.

QSP-3 Utilization Rate Could Exceed the Maximum on Withdrawals

Severity: *Medium Risk*

Status: Acknowledged

File(s) affected: `EToken.sol`

Description: The `withdraw()` function decreases the total supply without affecting the value of the `scr`, thus increasing the utilization rate. No validation is done on the utilization rate to confirm that it stays below the maximum, resulting in an appropriate course of action when the rate is exceeded.

Recommendation: Validate that the max utilization rate is not exceeded during withdrawals.

Update: The team has acknowledged this issue with the following statement: "This a feature, not a bug. The `maxUtilizationRate` parameter is there only to avoid selling more policies (locking more capital) when the UR is high, but we don't want to restrict LPs from withdrawing their funds. The only limit for withdrawal is UR < 100% (Check https://docs.ensuro.co/product-docs/smart-contracts/liquidity-pools#operation-summary). For restricting withdrawals besides SCR locker, we have the parameter `liquidityRequirement` that's there for specific situations (More on https://docs.ensuro.co/product-docs/smart-contracts/liquidity-pools#withdrawal).".

QSP-4 Upgradeable Contract Storage Gaps

Severity: *Low Risk*

Status: Fixed

File(s) affected: `PolicyPoolComponent.sol`, `RiskModule.sol`, `Reserve.sol`

Description: For future update considerations, add storage gaps in the upgradeable base contracts to avoid storage collisions when introducing new variables. This pattern is used in OpenZeppelin's upgradeable contracts.

Recommendation: Add a storage gap at the end of the upgradeable base contracts.

Update: The team has implemented the recommended storage gaps in the following commits: commit1, commit2, commit3.

QSP-5 Revocable Admin Roles

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `AccessManager.sol`

Description: OpenZeppelin's `AccessControlUpgradeable` contract contains the `revokeRole()` and `renounceRole()` functions which allows privileged addresses to give up privileged roles. If the `DEFAULT_ADMIN_ROLE` privileged role was renounced/revoked, all admin associated functionality associated would be inaccessible.

Recommendation: Consider if admin role revocation is a necessary feature. If it is not, override admin revocation functionality to disable it by modifying `renounceRole()` and `revokeRole()` functions to ensure that the `DEFAULT_ADMIN_ROLE` cannot be revoked. If it is, add end-user documentation stating the risk.

Update: The team has decided to leave the revocation methods as is and added a cautionary comment in the following commit.

QSP-6 Missing Input Validation

Severity: *Low Risk*

Status: Mitigated

File(s) affected: `FlighDelayRiskModule.sol`, `PolicyPool.sol`, `PriceRiskModule.sol`, `LPManualWhitelist.sol`, `LiquidityThresholdAssetManager.sol`, `RiskModule.sol`, `Reserve.sol`, `EToken.sol`, `Policy.sol`

Description: It is crucial to validate inputs, even if the inputs come from trusted addresses, to avoid human error. A lack of robust input validation can only increase the likelihood and impact in the event of mistakes.
The following is a non-exhaustive list of functions that do not validate their input addresses to be non-zero:

- `EToken._setAssetManager()`
- `EToken.repayLoan()`
- `PriceRiskModule.constructor()`
- `PolicyPool.setTreasury()`
- `PolicyPoolComponent.constructor()`
- `PremiumAccounts._setAssetManager()`
- `PremiumAccounts._borrowFromEtk()`

- `PremiumAccounts.withdrawWonPremiums()`
- `LPManualWhitelist.constructor()`
- `FlightDelayRiskModule.__FlightDelayRiskModule_init_unchained()`
- `FlightDelayRiskModule.setOracleParams()`
- `Reserve._transferTo()`
- `RiskModule.setWallet()`
- `LiquidityThresholdAssetManager.constructor()`

Additionally some functions do not validate non address related inputs:

- `EToken._validateParameters()`: Validate that `_params.minUtilizationRate` is less than `_params.maxUtilizationRate`.
- `EToken.setParam()`: Call `_validateParameters()`.
- `EToken.replayLoan()`: The amount should be greater than zero.
- `EToken._mint()`: The amount should be greater than zero.
- `PolicyPool.initialize()`: Validate `name_` and `symbol_` are non-empty strings.
- `RiskModule.params()`: Validate that `_params` has been set.
- `PriceRiskModule.newPolicy()`: Validate that the caller has the `PRICER_ROLE`.
- `FlightDelayRiskModule.newPolicy()`: Validate that the flight number is a non empty string.
- `PolicyPool.addComponent()`: Validate that `kind` correctly matches the component being added.
- `Policy.hash()`: Validate that the hash being returned is non-zero.

Recommendation: Implement the recommended checks.

Update: The team has fixed certain validations in the following commits: commit1, commit2, commit3. While acknowledging others with explanations found here.

QSP-7 Loss of Precision Borrows More Funds than Necessary

Severity: *Low Risk*

Status: Fixed

File(s) affected: `PremiumAccounts.sol`

Description: `setDeficitRatio()` borrows funds from eTokens if the surplus is less than the new `maxDeficit`. However, the new `maxDeficit` is calculated using the `newRatio` before it is later truncated when assigned in `_params.deficitRatio = (newRatio / 1e14).toUint16();`. `_maxDeficit(deficitRatio())` will be less than or equal to `_maxDeficit(newRatio)`, meaning more funds than necessary may be borrowed.

Recommendation: Truncate the `newRatio` before using it to call `int256 maxDeficit = _maxDeficit(newRatio);`.

Update: The team has truncated the ratio and verified before borrowing as seen in this commit.

QSP-8 Unlocked Pragma

Severity: *Low Risk*

Status: Fixed

Related Issue(s): SWC-103,

Description: Every Solidity file specifies in the header a version number of the format `pragma solidity (^)0.8.*`. 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, we recommend removing the caret to lock the file onto a specific Solidity version.

Update: The team has locked the pragma to `0.8.16` in the following commit. The pragma was left unlocked in their npm package (@ensuro/core), to ease integration for developers.

QSP-9 Policies Can Be Created for Flights Already in Transit

Severity: *Low Risk*

Status: Fixed

File(s) affected: `FlightDelayRiskModule.sol`

Description: `FlightDelayRiskModule.newPolicy()` does the following timestamp validation:

```
require(expectedArrival > block.timestamp, "expectedArrival can't be in the past");
require(departure != 0 && expectedArrival > departure, "expectedArrival <= departure!");
```

This leaves it possible to create policies for flights that have already departed if `expectedArrival > block.timestamp > departure`. A policy could be created for a flight that is known to be late if its departure time is known to be too late to make its expected arrival time.

Recommendation: Validate that `block.timestamp < departure`.

Update: The team has added a validation that prevents them from creating a flight risk policy 4 hours or less before a flight departure. See the following commit.

QSP-10 Base Contracts Should Update Their Initialization Modifiers

Severity: *Low Risk*

Status: Fixed

File(s) affected: `PolicyPoolComponent.sol`, `RiskModule.sol`

Description: The `initializer` modifier can only be called once even when using inheritance, so parent/base contracts should use the `onlyInitializing` modifier.

Recommendation: Replace all instances of the `initializer` modifier with `onlyInitializing`.

Update: This issue was addressed with the fix for QSP-1.

QSP-11 Inaccurate Accounting if Deflationary Tokens Are Accepted

Severity: *Low Risk*

Status: Fixed

File(s) affected: `PolicyPool.sol`

Description: The `PolicyPool.deposit()` function mints users eTokens based on the `amount` deposited. If deflationary tokens are accepted in the future, internal accounting will be inaccurate for tokens that subtract fees during transfer. The value stored in the protocol may be different from the value tracked using the `amount` in the `PolicyPool.deposit()` function. Inaccurate accounting of deflationary tokens can lead to insolvency during withdrawal.

Recommendation: Since USDC is only supported and the team controls which tokens are accepted, this finding is marked as low. However, if the team wishes to handle tokens with fees on transfer correctly, only account for the difference between the pre-transfer balance and the post-transfer balance of the pool. The difference will capture the amount of tokens that have been transferred, regardless of the token transfer mechanism.

Update: The team has implemented the recommended change, computing the amount of tokens deposited via the difference between the balance before and after transfer, in the following commit.

QSP-12 Checks-Effects-Interactions Pattern Violation

Severity: *Low Risk*

Status: Fixed

File(s) affected: `EToken.sol`, `PremiumAccounts.sol`

Related Issue(s): SWC-107

Description: The Checks-Effects-Interactions coding pattern is meant to mitigate any chance of other contracts manipulating the state of the blockchain in unexpected and possibly malicious ways before control is returned to the original contract. As the name implied, only after checking whether appropriate conditions are met and acting internally on those conditions should any external calls to, or interactions with, other contracts be done. The following functions violate these conditions:

- `EToken.transferFrom()`, `_approve` is called after `_transfer`.
- `EToken.repayLoan()`, `currency().safeTransferFrom()` is called prior to making state changes to `loan`.
- `PremiumsAccount.recieveGrant()`, `currency().safeTransferFrom()` is called prior to `_storePurePremiumWon(amount)`.

Recommendation: We recommend refactoring the code so that it conforms to the Checks-Effects-Interactions pattern.

Update: The team has implemented the recommended code changes in the following commit.

QSP-13 Risk of Killing Upgrades

Severity: *Low Risk*

Status: Mitigated

Description: The UUPS pattern was chosen for the upgradeable contracts. One of the drawbacks of using such a pattern is that if a future implementation does not implement the `upgradeTo` function, then upgrades for the contracts have effectively been killed.

Recommendation: Understand the drawbacks of using a UUPS pattern and document the potential risks for users.

Update: The team has acknowledged this issue with the following statement:
"We understand the risk around updates. To mitigate that risk we use the OpenZeppelin Upgrades library that does several checks. Also we create specific upgrade/fork tests before any upgrade to a live contract. Check this upgrade we did for Ensuro 1.0. With the storage gaps introduced in this version and the related test, we have even more protection. ".

QSP-14 `PriceRiskModule` Susceptible to Oracle Manipulation and Flash Loan Attacks

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `PriceRiskModule.sol`

Description: The loss probability is computed based on the difference between the current price and the trigger price of the insured asset. The current price of the insured asset is obtained from the oracle; depending on the liquidity market condition of the insured asset, the oracle values can be manipulated.
Anyone can obtain a price policy insurance using `newPolicy()` and resolve the policy with `triggerPolicy()`. Depending on the exposure limit `exposureLimit_` set in the PriceRiskModule, an economic exploit can be profitable.
Furthermore, since there is no minimum policy duration, flash loan attacks are possible if the oracle is vulnerable to flash loan manipulation.

Exploit Scenario:

1. Ensuro creates a PriceRiskModule for MNGO token.
2. A trader obtains price insurance for MGNO. The current price of MNGO is $ 0.02 and the trigger price is set to $ 0.91.
3. The trader set the payout to the maximum available payout as determined in `maxPayoutPerPolicy_`.
4. The trader opens multiple policies to reach the maximum exposure limit of the risk module using `newPolicy()`.
5. Since the price movement of the policy is large, the loss probability calculated is low, therefore the premium paid by the trader is low.
6. The trader manipulates the markets to drive up the price of MNGO to $ 0.91.

7. The trader immediately resolves all policies to obtain the maximum payouts using `triggerPolicy()`.

Recommendation: This is a limitation of the protocol design. Consider using an oracle that is less susceptible to price manipulation and be cautious of the asset offered in the `PriceRiskModule`, as well as potentially adding a minimum policy duration to make the protocol less susceptible to flash loan attacks.

Update: The team has acknowledged this issue and introduced the following code change , providing the following statement:
"We added an additional parameter for the PriceRiskModule defining the “minimum time that must elapse before a policy can be triggered”. This gives some protection against oracle update delays or acquisition of policies immediately after an event that can take prices down. For this product we will use only assets with high volume / market cap, that aren’t easily manipulable. Since we are using Chainlink oracles, we aren’t exposed to flash-loan attacks.".

QSP-15 Resolve Same Policy Multiple Times if `ERC777` Tokens Are Accepted

Severity: *Low Risk*

Status: Fixed

File(s) affected: `PolicyPool.sol`, `PremiumsAccount.sol`

Description: If ERC777 tokens are accepted, it would be possible for a recipient to re-enter the `resolvePolicy()` function. Depending on the risk module setup, if a policyholder can resolve a policy through the risk module, it is possible to resolve an existing policy multiple times with re-entrancy. The policy data is only deleted once the fund transfer has been completed in `_resolvePolicy()`.
The ERC777 token standard was created to extend the capabilities available in ERC20 tokens and one of the features allows the ERC777 token contract to notify the sender and recipients when ERC777 tokens are sent or received. Therefore it is possible to call the `resolvePolicy()` function again during the ERC777 hook when initiating the ERC777 token transfer.

Recommendation: Add nonReentrant modifier to `_resolvePolicy()` function.

Update: The team has chosen to mitigate this issue in the following commit, utilizing the checks-effects-interactions pattern.

QSP-16 Missing Events to Signal State Changes

Severity: *Informational*

Status: Fixed

File(s) affected: `FlightDelayRiskModule.sol`

Description: Events are important for signaling state changes in a contract. This helps debug the contract in the event of any attacks or critical bugs and helps indicate to users any configuration changes in the contract.
A non-exhaustive list of functions where events can be useful includes:

- `FlightDelayRiskModule.setOracleParams()`

Recommendation: Add events to signal the contract's state changes.

Update: The team has added and emitted the recommended event in the following commit.

QSP-17 Max Approval to Junior and Senior Tranches

Severity: *Informational*

Status: Fixed

File(s) affected: `PremiumsAccount.sol`

Description: The `PremiumsAccount` contract approves the maximum amount for the `currency` to the junior and senior contracts. Providing the maximum token access to other contracts can be problematic in the event of compromised contracts.

Recommendation: Since this is a design choice, this issue is listed as an informational. Developers should consider approving only when needed and with the required amount to limit exposure risk.

Update: The team has removed infinite approvals, opting to approve only the amount required for loan repayment in the following commit.

QSP-18 Profitability of the Protocol Depends on Accuracy of Risk Model

Severity: *Informational*

Status: Acknowledged

Description: A policyholder pays a premium to obtain a policy from Ensuro; a part of the premium represents the expected loss of the risks taken. If the payout exceeds the premium paid by the policyholder, Ensuro needs to pay beyond the premium provided by the policyholder. Therefore, the profitability of the protocol relies on the accuracy of the risk models, computed off-chain, used to assess the risks taken.
Furthermore, the risk model can be completely accurate, but a black swan event impacting multiple policies can lead to fund insolvency when there are insufficient funds or funds to borrow to resolve impacted policies. Fund insolvency can be remedied using `PremiumsAccount.receiveGrant()` which injects funds into the premium account.

Recommendation: None. This is a limitation of the protocol design.

Update: The team has acknowledged this issue and added a section in their documentation describing their risk management process.

QSP-19 Upgradability

Severity: *Informational*

Status: Acknowledged

Description: Many contracts within the project are upgradeable. While this is not a vulnerability, users should be aware that the behavior of the contracts could drastically change if the contracts are upgraded. Furthermore, new vulnerabilities not present during the audit could be introduced in upgraded versions of the contract, or if contract upgrade deployments are not done correctly.

Recommendation: The fact that the contract can be upgraded and reasons for future upgrades should be communicated to users beforehand.

Update: The team has acknowledged this issue and provided documentation on their approach towards upgradability.

QSP-20 Loans Can Be Repayed when the Contract Is Paused

Severity: *Undetermined*

Status: Fixed

File(s) affected: `EToken.sol`

Description: While most token operations require the `whenNotPaused` modifier, `repayLoan()` can be called when the contract is paused.

Recommendation: Clarify if this behavior is intentional. If not, add the missing `whenNotPaused` modifier.

Update: The team has added the necessary modifier in the following commit, forbidding repayments when the token is paused.

QSP-21 May Not Have Adequate Funds to Deinvest

Severity: *Undetermined*

Status: Acknowledged

File(s) affected: `PremiumAccounts.sol`

Description: The function `_repayLoan()` will deinvest money if the contract's balance is not sufficient to repay the loan. The function has been reproduced below:

```
function _repayLoan(uint256 purePremiumWon, IEToken etk) internal returns (uint256) {
    if (purePremiumWon < NEGLIGIBLE_AMOUNT) return purePremiumWon;
    uint256 borrowedFromEtk = etk.getLoan(address(this));
    if (borrowedFromEtk == 0) return purePremiumWon;
    uint256 repayAmount = Math.min(purePremiumWon, borrowedFromEtk);

    // If not enought liquidity, it deinvests from the asset manager
    if (currency().balanceOf(address(this)) < repayAmount) {
      _refillWallet(repayAmount);
    }
    etk.repayLoan(repayAmount, address(this));
    return purePremiumWon - repayAmount;
}
```

The contract calls `_refillWallet(repayAmount)` when it only requires `_refillWallet(repayAmount - currency().balanceOf(address(this)))`. Requesting a larger amount may result in a revert if funds are insufficient.

Recommendation: Update `_repayLoan()` to call `_refillWallet(repayAmount - currency().balanceOf(address(this)))`.

Update: The team has acknowledged this issue, adding code comments, and provided the following statement:
"The code was correct as it was, because the `IAssetManager` expects the full amount that needs to be paid, not just the missing amount. The reason for this behaviour is the asset manager will try to deinvest more than the required amount, to avoid continuous deinvestments. We added a comment to clarify this in the code." .

QSP-22 Risk of Reentrancy when Creating a New Policy

Severity: *Undetermined*

Status: Mitigated

File(s) affected: `PolicyPool.sol`

Description: In `PolicyPool.sol`, the `newPolicy()` function calls `safeMint()` which invokes an external call to verify if the recipient address is an ERC721 receiver during `_checkOnERC721Received()`. Since it is an external call to an unknown contract, where the recipient contract may define any arbitrary logic to be executed, `newPolicy()` function should be protected. Similarly, an untrusted ERC20 token can re-enter during policy creation or resolution.

Recommendation: Add `nonReentrant` modifier to the `newPolicy()` function.

Update: The team has mitigated this issue by choosing to implement the checks-effects-interactions pattern in the following commit, along with the code changes implemented for `QSP-15`.

QSP-23 `TrustfulRiskModule` Interacts with Off-Chain Software

Severity: *Undetermined*

Status: Acknowledged

File(s) affected: `TrustfulRiskModule.sol`

Description: Based off the descriptions found in the `TrustfulRiskModule` and documentation provided to us, our understanding is that this risk module is handled by off-chain software, with the ability to create and resolve policies with minimal on-chain validation. Additional code that handles such critical operations, should be audited and well documented.

Recommendation: Audit and provide extensive documentation on the off-chain software interacting with the `TrustfulRiskModule`.

Update: The team has acknowledged this issue with the following statement: "We understand this risk. It will depend of each particular risk, but in some cases the permissions to trigger the policies are delegated to our partners. They have a (written) contractual obligation of resolving the policies based on the terms of the coverage. We do random audits on samples of the policies triggered to verify they complain. Besides this, we track the performance of the risk modules and we unlock increasing exposure limits as we gain confidence on the partner. ".

Adherence to Specification

In the product docs provided to us, we noticed what we believe to be a typo in the following instance discussing the collateralization ratio:

```
As mentioned, 54.1% is the collateralization ratio for 99.5% confidence. If we take 70% confidence, the junior collateralization ratio will be 50.8%. And as we know, chances of getting a head are 50%.
So, for a $ 1 payout, the solvency will be $ 0.541 broken down this way:
pure premium = $ 0.50
Junior Scr = $ 0.08
Senior Scr = $ 0.33
```

Following the explanation on how collateralization payouts are calculated, we believe `Junior Src` should be $0.008 and `Senior Scr` should be $0.033.
Update: The team has corrected this in their documentation.

Code Documentation

- `LiquidityThresholdAssetManager.sol#L84`, code comment seems to be unfinished.
- `PolicyPool.sol#L169`, `role` is misspelled as `rolea`.
- `PremiumAccounts.sol#L144`, `loans` is misspelled as `lons`.
- `PremiumAccounts.sol#445`, `enough` is misspelled as `enought`.
- `PolicyPool.sol#451`, TODO notes should be removed in production-ready code.
- Comments in `PolicyPoolComponent.sol` are not consistent. Based on the comment, `LEVEL2` should be able to `addComponent`, but only `LEVEL1` is allowed. Consider modifying the comment.
- Adding comments to every non-trivial function is recommended.

Adherence to Best Practices

The team has implemented best practice fixes in the following commit.

- [**Fixed**] For `LPManualWhitelist.whitelistAddress()`, only emit the `LPWhitelisted` event if the whitelisted value has changed.
- [**Fixed**] In `RiskModule#L117`, extract 24 * 365 into a constant.
- [**Not Implemented**] When creating a new policy in `PolicyPool.newPolicy()` the distribution of premium can be internally tracked in the `PolicyPool` with a new mapping balance. Instead of transferring the payment to participants, the participants can pull the payments instead.
- [**Not Implemented**] The following functions can be renamed to more accurately convey its functionality:
  - `AccessManager.hasComponentRole()` can be renamed to `AccessManager.hasGlobalRoleOrComponentRole()`.
  - `AccessManager.checkComponentRole()` can be renamed to `AccessManager.checkGlobalRoleOrComponentRole()`.
- [**Fixed**] Consider replacing usage of magic numbers in the following functions, but not limited to:
  - `RiskModule.__RiskModule_init_unchained()`
  - `RiskModule._validateParameters()`
  - `RiskModule._4toWad()`
  - `RiskModule._wadTo4()`
- [**Fixed**] `PremiumsAccount` inherits from `Reserve` which inherits from `PolicyPoolComponent`. However, when initializing `PremiumsAccount`, it calls into the initialization function for `PolicyPoolComponent`, skipping over `Reserve`, making it confusing in regards to what contract is being initialized. Consider implementing an initialization function for `Reserve` that calls into `PolicyPoolComponent`, that will then be called from `PremiumsAccount`. This will result in better readability.

Test Results

Test Suite Results

We ran both the brownie and hardhat tests. The majority of tests were passing, however 10 tests were skipped in the Brownie tests.

```
AccessManager
    ✓ Only allows roleAdmin to grant component roles (159ms)
    ✓ Checks global roles only when asked to (91ms)

 Test add, remove and change status of PolicyPool components
    ✓ Change status and remove eToken (218ms)
    ✓ Change status and remove RiskModule (240ms)
    ✓ Change status and remove PremiumsAccount (322ms)

 Constructor validations
    ✓ Checks PolicyPool constructor validations (140ms)
    ✓ Checks EToken constructor validations
    ✓ Checks PremiumsAccount constructor validations
    ✓ Checks TrustfulRiskModule constructor validations (298ms)
    ✓ Checks SignedQuoteRiskModule constructor validations (120ms)
    ✓ Checks LPManualWhitelist constructor validations
    ✓ Checks ERC4626AssetManager constructor validations

 Etoken
    ✓ Refuses transfers to null address (215ms)
    ✓ Checks user balance
    ✓ Returns the available funds
    ✓ Only allows PolicyPool to add new borrowers
    ✓ Only allows PolicyPool to remove borrowers
    ✓ Allows setting whitelist to null
    ✓ Can't create etoken without name or symbol

Test Implementation contracts can't be initialized
    ✓ Does not allow initialize AccessManager implementation
    ✓ Does not allow initialize PolicyPool implementation (63ms)
    ✓ Does not allow initialize EToken implementation (157ms)
    ✓ Does not allow initialize PremiumsAccount implementation
    ✓ Does not allow initialize TrustfulRiskModule implementation (212ms)
    ✓ Does not allow initialize SignedQuoteRiskModule implementation
    ✓ Does not allow initialize LPManualWhitelist implementation

Test Initialize contracts
    ✓ Does not allow reinitializing PolicyPool
    ✓ Does not allow reinitializing Etoken
    ✓ Does not allow reinitializing AccessManager
    ✓ Does not allow reinitializing PremiumsAccount
    ✓ Does not allow reinitializing RiskModule
    ✓ Does not allow reinitializing Whitelist

Test pause, unpause and upgrade contracts
    ✓ Pause and Unpause PolicyPool (185ms)
    ✓ Pause/Unpause and resolve policy with full payout (129ms)
    ✓ Pause/Unpause and expire policy (141ms)
    ✓ Pause and Unpause EToken (140ms)
    ✓ Pause and Unpause RiskModule resolve policy (122ms)
    ✓ Pause and Unpause RiskModule resolve policy full payout (95ms)
    ✓ Pause and Unpause PremiumsAccount policyExpired (117ms)
    ✓ Pause and Unpause PremiumsAccount policyResolvedWithPayout (129ms)
    ✓ Pause and Unpause EToken trying to create and expire policies (100ms)

PolicyPool contract
    ✓ Only allows LEVEL1_ROLE to change the treasury (171ms)
```

```
    ✓ Only allows LEVEL1_ROLE to add components (99ms)
    ✓ Does not allow adding an existing component (56ms)
    ✓ Does not allow adding different kind of component (188ms)
    ✓ Does not allow adding a component that belongs to a different pool (140ms)
    ✓ Adds the PA as borrower on the jr etoken (127ms)
    ✓ Removes the PA as borrower from the jr etoken on PremiumsAccount removal (138ms)
    ✓ Adds the PA as borrower on the sr etoken (187ms)
    ✓ Removes the PA as borrower from the sr etoken on PremiumsAccount removal (199ms)
    ✓ Does not allow suspending unknown components (70ms)
    ✓ Only allows riskmodule to create policies
    ✓ Only allows to resolve a policy once (805ms)
    ✓ Only allows riskmodule to resolve unexpired policies
    ✓ Does not allow a bigger payout than the one setup in the policy
    ✓ Initialize PolicyPool without name and symbol fails (129ms)

  Policy initialize
    ✓ Does not allow premium greater than payout
    ✓ Correctly collateralizes with jr etoken

  RiskModule contract
    ✓ Set params jrCollRatio validations
    ✓ Allows msg.sender as payer
    ✓ Doesn't allow another payer by default
    ✓ Allows another payer given the right allowances (38ms)
    ✓ Does not allow an exposure limit of zero
    ✓ Does not allow wallet with zero address
    ✓ Does not allow a maxpayout of zero

  SignedQuoteRiskModule contract tests
    ✓ Creates a policy if the right signature is provided (676ms)
    ✓ Rejects a policy if signed by unauthorized user
    ✓ Rejects a policy if receives an invalid signature
    ✓ Rejects a policy if quote expired or validUntil changed
    ✓ Creates a policy where payer != msg.sender using newPolicyPaidByHolder (110ms)
    ✓ If creation is not open, only authorized users can create policies (493ms)

  Storage Gaps
    ✓ EToken has a proper storage gap (93ms)
    ✓ AccessManager has a proper storage gap (144ms)
    ✓ LPManualWhitelist has a proper storage gap (70ms)
    ✓ PolicyPool has a proper storage gap (69ms)
    ✓ PolicyPoolComponent has a proper storage gap (71ms)
    ✓ PremiumsAccount has a proper storage gap (75ms)
    ✓ Reserve has a proper storage gap (75ms)
    ✓ RiskModule has a proper storage gap (83ms)
    ✓ SignedQuoteRiskModule has a proper storage gap (81ms)
    ✓ TrustfulRiskModule has a proper storage gap (76ms)

  Supports interface implementation
    ✓ Checks AccessManager supported interfaces (69ms)
    ✓ Checks PolicyPool supported interfaces (143ms)
    ✓ Checks EToken supported interfaces
    ✓ Checks PremiumsAccount supported interfaces (245ms)
    ✓ Checks Reserves reject invalid asset manager
    ✓ Checks TrustfulRiskModule supported interfaces
    ✓ Checks SignedQuoteRiskModule supported interfaces
    ✓ Checks LPManualWhitelist supported interfaces
    ✓ Checks ERC4626AssetManager supported interfaces

  Test Upgrade contracts
    ✓ Should be able to upgrade EToken (343ms)
    ✓ Can upgrade EToken with componentRole (50ms)
    ✓ Should not be able to upgrade EToken with different pool (141ms)
    ✓ Should be able to upgrade PremiumsAccount contract (48ms)
    ✓ Can upgrade PremiumsAccount with componentRole (55ms)
    ✓ Should not be able to upgrade PremiumsAccount with different pool or jrEtk (263ms)
    ✓ Should be able to upgrade RiskModule contract (480ms)
    ✓ Can upgrade RiskModule with componentRole (68ms)
    ✓ Should not be able to upgrade RiskModule with different pool or PremiumsAccount (307ms)
    ✓ Should be able to upgrade Whitelist (258ms)
    ✓ Can upgrade Whitelist with componentRole (60ms)
    ✓ Should be able to upgrade AccessManager contract (206ms)

Test PriceRiskModule contract
    ✓ Should return the asset oracle (618ms)
    ✓ Should return the reference oracle (129ms)
    ✓ Should return the oracle tolerance (121ms)
    ✓ Should never allow reinitialization (131ms)
    ✓ Should only allow ORACLE_ADMIN to set the oracleTolerance (157ms)
    ✓ Should only allow ORACLE_ADMIN to set the minDuration (159ms)
    ✓ Should only allow PRICER to set CDFs (184ms)
    ✓ Should not allow setting prices for policies with no duration (222ms)
    ✓ Should not allow new policies if prices are not defined (155ms)
    ✓ Should not allow new policies if prices are not fresh (147ms)
    ✓ Should not allow address(0) for the asset oracle
    ✓ Should not allow address(0) for the policy owner (109ms)
    ✓ Should reject if trigger price has already been reached (138ms)
    ✓ Should allow address(0) for the reference oracle (87ms)
    ✓ Should calculate exchange rate for single asset with no reference (111ms)
    ✓ Should calculate policy premium and loss for single asset with no reference (201ms)
    ✓ Should calculate exchange rate between different assets (Wad vs 6 decimals) (226ms)
    ✓ Should calculate exchange rate between different assets (Ray vs 9 decimals) (220ms)
    ✓ Should not allow policy creation/triggering below min duration (339ms)
    ✓ Should calculate policy premium and loss probability (1% slots) (226ms)
    ✓ Should calculate policy premium and loss probability (13% slots) (218ms)
    ✓ Should calculate policy premium and loss probability (5% slots, shorted asset) (215ms)
    ✓ Should calculate policy premium and loss probability (1% slots, Wad vs 6 decimals) (227ms)
    ✓ Should calculate policy premium and loss probability (1% slots, Ray vs 9 decimals) (234ms)
    ✓ Should calculate policy premium and loss probability (1% slots, Ray vs 6 decimals) (219ms)
    ✓ Should trigger the policy only if threshold met (326ms)
    ✓ Should trigger the policy only if threshold met - Shorted asset (308ms)
    ✓ Should not allow operations when paused (177ms)
    ✓ Should work with real chainlink oracles (forking at https://polygonscan.com/block/34906609) (2681ms)

  Storage Gaps
    ✓ PriceRiskModule has a proper storage gap (58ms)

 Test AAVE asset manager - running at https://polygonscan.com/block/33313517
    ✓ Creates an asset manager and invests in AAVE-v2 (380ms)
    ✓ The same AM contract can be shared between reserves - just shares code - AAVE-v2 (333ms)
    ✓ Creates an asset manager and invests in AAVE-v3 (378ms)
    ✓ The same AM contract can be shared between reserves - just shares code - AAVE-v3 (250ms)

 FlightDelayRiskModule contract
    ✓ Allows only oracle admin to set oracle params (793ms)
    ✓ Emits an event when setting oracle params (129ms)
    ✓ Oracle address validation (152ms)
    ✓ Allows only PRICER_ROLE to add new policies (553ms)
    ✓ Performs the oracle request and receive cycle (110ms)
    ✓ Transfers the oracle fee for new policies (80ms)
    ✓ Allows only PRICER_ROLE to resolve policies (129ms)
    ✓ Resolves policy with no payout when flight arrives on time (110ms)
    ✓ Resolves policy with full payout when flight arrives too late (123ms)
    ✓ Resolves policy with full payout when flight is cancelled (119ms)
    ✓ Resolves policy with full payout when oracle is missing arrival past tolerance (147ms)
    ✓ Resolves with full payout on manual resolve for cancelled flight (132ms)
    ✓ Resolves with zero payout on manual resolve for on-time flight (116ms)
    ✓ Rejects policies with expected arrival in the past
    ✓ Rejects policies with expected arrival before departure
    ✓ Reverts when resolving unknown policy
    ✓ Cant create policy in the past

  Storage Gaps
    ✓ FlightDelayRiskModule has a proper storage gap (92ms)

tests/test_etoken.py::test_only_policy_pool_validation[prototype] PASSED                                                                                [  0%]
tests/test_etoken.py::test_only_policy_pool_validation[ethereum] PASSED                                                                                 [  1%]
tests/test_etoken.py::test_deposit_withdraw[prototype] PASSED                                                                                           [  1%]
tests/test_etoken.py::test_deposit_withdraw[ethereum] PASSED                                                                                            [  2%]
tests/test_etoken.py::test_lock_unlock_scr[prototype] PASSED                                                                                            [  3%]
tests/test_etoken.py::test_lock_unlock_scr[ethereum] PASSED                                                                                             [  3%]
tests/test_etoken.py::test_etoken_erc20[prototype] PASSED                                                                                               [  4%]
tests/test_etoken.py::test_etoken_erc20[ethereum] PASSED                                                                                                [  5%]
tests/test_etoken.py::test_multiple_policies[prototype] PASSED                                                                                          [  5%]
tests/test_etoken.py::test_multiple_policies[ethereum] PASSED                                                                                           [  6%]
tests/test_etoken.py::test_multiple_lps[prototype] PASSED                                                                                               [  7%]
tests/test_etoken.py::test_multiple_lps[ethereum] PASSED                                                                                                [  7%]
tests/test_etoken.py::test_lock_scr_validation[prototype] PASSED                                                                                        [  8%]
tests/test_etoken.py::test_lock_scr_validation[ethereum] PASSED                                                                                         [  8%]
tests/test_etoken.py::test_internal_loan[prototype] PASSED                                                                                              [  9%]
```

```
tests/test_etoken.py::test_internal_loan[ethereum] PASSED [ 10%]
tests/test_etoken.py::test_etk_asset_manager[prototype] PASSED [ 10%]
tests/test_etoken.py::test_etk_asset_manager[ethereum] PASSED [ 11%]
tests/test_etoken.py::test_etk_asset_manager_liquidity_under_minimum[prototype] PASSED [ 12%]
tests/test_etoken.py::test_etk_asset_manager_liquidity_under_minimum[ethereum] PASSED [ 12%]
tests/test_etoken.py::test_name_and_others[prototype] PASSED [ 13%]
tests/test_etoken.py::test_name_and_others[ethereum] PASSED [ 14%]
tests/test_etoken.py::test_max_utilization_rate[prototype] PASSED [ 14%]
tests/test_etoken.py::test_max_utilization_rate[ethereum] PASSED [ 15%]
tests/test_etoken.py::test_unlock_scr[prototype] PASSED [ 16%]
tests/test_etoken.py::test_unlock_scr[ethereum] PASSED [ 16%]
tests/test_etoken.py::test_unlock_scr_with_adjustment[prototype] PASSED [ 17%]
tests/test_etoken.py::test_unlock_scr_with_adjustment[ethereum] PASSED [ 17%]
tests/test_etoken.py::test_unlock_scr_with_neg_adjustment[prototype] PASSED [ 18%]
tests/test_etoken.py::test_unlock_scr_with_neg_adjustment[ethereum] PASSED [ 19%]
tests/test_etoken.py::test_getset_etk_parameters_tweaks[prototype] PASSED [ 19%]
tests/test_etoken.py::test_getset_etk_parameters_tweaks[ethereum] PASSED [ 20%]
tests/test_etoken.py::test_mint_to_zero_address[prototype] SKIPPED (mint not fully implemented in Python) [ 21%]
tests/test_etoken.py::test_mint_to_zero_address[ethereum] PASSED [ 21%]
tests/test_fixedratevault.py::test_constructor[prototype] PASSED [ 22%]
tests/test_fixedratevault.py::test_constructor[ethereum] PASSED [ 23%]
tests/test_fixedratevault.py::test_deposit_withdraw_one_lp[prototype] PASSED [ 23%]
tests/test_fixedratevault.py::test_deposit_withdraw_one_lp[ethereum] PASSED [ 24%]
tests/test_fixedratevault.py::test_deposit_withdraw_two_lp[prototype] PASSED [ 25%]
tests/test_fixedratevault.py::test_deposit_withdraw_two_lp[ethereum] PASSED [ 25%]
tests/test_fixedratevault.py::test_value_grows_with_interest_rate[prototype] PASSED [ 26%]
tests/test_fixedratevault.py::test_value_grows_with_interest_rate[ethereum] PASSED [ 26%]
tests/test_policypool.py::test_transfers[prototype] PASSED [ 27%]
tests/test_policypool.py::test_transfers[ethereum] PASSED [ 28%]
tests/test_policypool.py::test_transfers_usdc[prototype] PASSED [ 28%]
tests/test_policypool.py::test_transfers_usdc[ethereum] PASSED [ 29%]
tests/test_policypool.py::test_not_accept_rm[prototype] SKIPPED [ 30%]
tests/test_policypool.py::test_not_accept_rm[ethereum] SKIPPED [ 30%]
tests/test_policypool.py::test_walkthrough[prototype] PASSED [ 31%]
tests/test_policypool.py::test_walkthrough[ethereum] PASSED [ 32%]
tests/test_policypool.py::test_nfts[prototype] PASSED [ 32%]
tests/test_policypool.py::test_nfts[ethereum] PASSED [ 33%]
tests/test_policypool.py::test_policy_holder_contract[prototype] SKIPPED (This test doesn't make much sense on prototype) [ 33%]
tests/test_policypool.py::test_policy_holder_contract[ethereum] PASSED [ 34%]
tests/test_policypool.py::test_partial_payout[prototype] PASSED [ 35%]
tests/test_policypool.py::test_partial_payout[ethereum] PASSED [ 35%]
tests/test_policypool.py::test_internal_loan_partial_payout[prototype] PASSED [ 36%]
tests/test_policypool.py::test_internal_loan_partial_payout[ethereum] PASSED [ 37%]
tests/test_policypool.py::test_increase_won_pure_premiums[prototype] PASSED [ 37%]
tests/test_policypool.py::test_increase_won_pure_premiums[ethereum] PASSED [ 38%]
tests/test_policypool.py::test_payout_bigger_than_pure_premium[prototype] PASSED [ 39%]
tests/test_policypool.py::test_payout_bigger_than_pure_premium[ethereum] PASSED [ 39%]
tests/test_policypool.py::test_asset_manager[prototype] PASSED [ 40%]
tests/test_policypool.py::test_asset_manager[ethereum] PASSED [ 41%]
tests/test_policypool.py::test_assets_under_liquidity_middle[prototype] SKIPPED [ 41%]
tests/test_policypool.py::test_assets_under_liquidity_middle[ethereum] SKIPPED [ 42%]
tests/test_policypool.py::test_distribute_negative_earnings[prototype] PASSED [ 42%]
tests/test_policypool.py::test_distribute_negative_earnings[ethereum] PASSED [ 43%]
tests/test_policypool.py::test_distribute_negative_earnings_full_capital_from_etokens[prototype] PASSED [ 44%]
tests/test_policypool.py::test_distribute_negative_earnings_full_capital_from_etokens[ethereum] PASSED [ 44%]
tests/test_policypool.py::test_distribute_negative_earnings_from_pool_and_etokens[prototype] SKIPPED [ 45%]
tests/test_policypool.py::test_distribute_negative_earnings_from_pool_and_etokens[ethereum] SKIPPED [ 46%]
tests/test_policypool.py::test_lp_whitelist[prototype] PASSED [ 46%]
tests/test_policypool.py::test_lp_whitelist[ethereum] PASSED [ 47%]
tests/test_policypool.py::test_expire_policy[prototype] PASSED [ 48%]
tests/test_policypool.py::test_expire_policy[ethereum] PASSED [ 48%]
tests/test_policypool.py::test_expire_policy_payout[prototype] PASSED [ 49%]
tests/test_policypool.py::test_expire_policy_payout[ethereum] PASSED [ 50%]
tests/test_policypool.py::test_withdraw_won_premiums[prototype] PASSED [ 50%]
tests/test_policypool.py::test_withdraw_won_premiums[ethereum] PASSED [ 51%]
tests/test_policypool.py::test_risk_provider_cant_drain_liquidity_provider[prototype] PASSED [ 51%]
tests/test_policypool.py::test_risk_provider_cant_drain_liquidity_provider[ethereum] PASSED [ 52%]
tests/test_policypool.py::test_same_asset_manager_for_etk_and_pa[prototype] PASSED [ 53%]
tests/test_policypool.py::test_same_asset_manager_for_etk_and_pa[ethereum] PASSED [ 53%]
tests/test_policypool.py::test_same_asset_manager_for_etk_and_pa_with_policy[prototype] PASSED [ 54%]
tests/test_policypool.py::test_same_asset_manager_for_etk_and_pa_with_policy[ethereum] PASSED [ 55%]
tests/test_policypool.py::test_same_asset_manager_for_etk_and_pa_resolve_policy[prototype] PASSED [ 55%]
tests/test_policypool.py::test_same_asset_manager_for_etk_and_pa_resolve_policy[ethereum] PASSED [ 56%]
tests/test_policypool.py::test_repay_loan[prototype] PASSED [ 57%]
tests/test_policypool.py::test_repay_loan[ethereum] PASSED [ 57%]
tests/test_premiumsaccount.py::test_premiums_account_creation[prototype] PASSED [ 58%]
tests/test_premiumsaccount.py::test_premiums_account_creation[ethereum] PASSED [ 58%]
tests/test_premiumsaccount.py::test_receive_grant[prototype] PASSED [ 59%]
tests/test_premiumsaccount.py::test_receive_grant[ethereum] PASSED [ 60%]
tests/test_premiumsaccount.py::test_withdraw_won_premiums[prototype] PASSED [ 60%]
tests/test_premiumsaccount.py::test_withdraw_won_premiums[ethereum] PASSED [ 61%]
tests/test_premiumsaccount.py::test_withdraw_won_premiums_with_borrowed_active_pp[prototype] PASSED [ 62%]
tests/test_premiumsaccount.py::test_withdraw_won_premiums_with_borrowed_active_pp[ethereum] PASSED [ 62%]
tests/test_premiumsaccount.py::test_policy_created_without_etokens[prototype] PASSED [ 63%]
tests/test_premiumsaccount.py::test_policy_created_without_etokens[ethereum] PASSED [ 64%]
tests/test_premiumsaccount.py::test_create_and_expire_policy_with_sr_etk[prototype] PASSED [ 64%]
tests/test_premiumsaccount.py::test_create_and_expire_policy_with_sr_etk[ethereum] PASSED [ 65%]
tests/test_premiumsaccount.py::test_policy_resolved_with_payout[prototype] PASSED [ 66%]
tests/test_premiumsaccount.py::test_policy_resolved_with_payout[ethereum] PASSED [ 66%]
tests/test_premiumsaccount.py::test_policy_created_with_jr_etoken[prototype] PASSED [ 67%]
tests/test_premiumsaccount.py::test_policy_created_with_jr_etoken[ethereum] PASSED [ 67%]
tests/test_premiumsaccount.py::test_policy_created_with_sr_etoken[prototype] PASSED [ 68%]
tests/test_premiumsaccount.py::test_policy_created_with_sr_etoken[ethereum] PASSED [ 69%]
tests/test_premiumsaccount.py::test_policy_created_with_jr_and_sr_etoken[prototype] PASSED [ 69%]
tests/test_premiumsaccount.py::test_policy_created_with_jr_and_sr_etoken[ethereum] PASSED [ 70%]
tests/test_premiumsaccount.py::test_pay_from_premium[prototype] PASSED [ 71%]
tests/test_premiumsaccount.py::test_pay_from_premium[ethereum] PASSED [ 71%]
tests/test_premiumsaccount.py::test_set_deficit_ratio[prototype] PASSED [ 72%]
tests/test_premiumsaccount.py::test_set_deficit_ratio[ethereum] PASSED [ 73%]
tests/test_premiumsaccount.py::test_set_deficit_ratio_without_adjustment[prototype] PASSED [ 73%]
tests/test_premiumsaccount.py::test_set_deficit_ratio_without_adjustment[ethereum] PASSED [ 74%]
tests/test_premiumsaccount.py::test_ratio_adjustment[prototype] PASSED [ 75%]
tests/test_premiumsaccount.py::test_ratio_adjustment[ethereum] PASSED [ 75%]
tests/test_premiumsaccount.py::test_set_deficit_ratio_and_create_policy[prototype] PASSED [ 76%]
tests/test_premiumsaccount.py::test_set_deficit_ratio_and_create_policy[ethereum] PASSED [ 76%]
tests/test_premiumsaccount.py::test_set_deficit_ratio_refuses_loss_of_precision[prototype] PASSED [ 77%]
tests/test_premiumsaccount.py::test_set_deficit_ratio_refuses_loss_of_precision[ethereum] PASSED [ 78%]
tests/test_premiumsaccount.py::test_pa_asset_manager[prototype] PASSED [ 78%]
tests/test_premiumsaccount.py::test_pa_asset_manager[ethereum] PASSED [ 79%]
tests/test_riskmodule.py::test_getset_rm_parameters[prototype] PASSED [ 80%]
tests/test_riskmodule.py::test_getset_rm_parameters[ethereum] PASSED [ 80%]
tests/test_riskmodule.py::test_getset_rm_parameters_tweaks[prototype] PASSED [ 81%]
tests/test_riskmodule.py::test_getset_rm_parameters_tweaks[ethereum] PASSED [ 82%]
tests/test_riskmodule.py::test_avoid_repeated_tweaks[prototype] SKIPPED (Tweaks not fully implemented in Python) [ 82%]
tests/test_riskmodule.py::test_avoid_repeated_tweaks[ethereum] PASSED [ 83%]
tests/test_riskmodule.py::test_set_rm_parameter_overflow[prototype] SKIPPED (Python doesn't have int limits ✓) [ 83%]
tests/test_riskmodule.py::test_set_rm_parameter_overflow[ethereum] PASSED [ 84%]
tests/test_riskmodule.py::test_new_policy[prototype] PASSED [ 85%]
tests/test_riskmodule.py::test_new_policy[ethereum] PASSED [ 85%]
tests/test_riskmodule.py::test_moc[prototype] PASSED [ 86%]
tests/test_riskmodule.py::test_moc[ethereum] PASSED [ 87%]
tests/test_riskmodule.py::test_minimum_premium[prototype] PASSED [ 87%]
tests/test_riskmodule.py::test_minimum_premium[ethereum] PASSED [ 88%]
tests/test_riskmodule.py::test_get_minimum_premium_with_high_jr_coll_ratio[prototype] PASSED [ 89%]
tests/test_riskmodule.py::test_get_minimum_premium_with_high_jr_coll_ratio[ethereum] PASSED [ 89%]
tests/test_riskmodule.py::test_get_minimum_premium_with_low_sr_coll_ratio[prototype] PASSED [ 90%]
tests/test_riskmodule.py::test_get_minimum_premium_with_low_sr_coll_ratio[ethereum] PASSED [ 91%]
tests/test_riskmodule.py::test_default_premium[prototype] PASSED [ 91%]
tests/test_riskmodule.py::test_default_premium[ethereum] PASSED [ 92%]
tests/test_riskmodule.py::test_premium_too_high[prototype] PASSED [ 92%]
tests/test_riskmodule.py::test_premium_too_high[ethereum] PASSED [ 93%]
tests/test_riskmodule.py::test_expiration_in_the_past_should_revert[prototype] PASSED [ 94%]
tests/test_riskmodule.py::test_expiration_in_the_past_should_revert[ethereum] PASSED [ 94%]
tests/test_riskmodule.py::test_max_duration[prototype] PASSED [ 95%]
tests/test_riskmodule.py::test_max_duration[ethereum] PASSED [ 96%]
tests/test_riskmodule.py::test_customer_with_zero_address[prototype] PASSED [ 96%]
tests/test_riskmodule.py::test_customer_with_zero_address[ethereum] PASSED [ 97%]
tests/test_riskmodule.py::test_exceeded_max_payout[prototype] PASSED [ 98%]
tests/test_riskmodule.py::test_exceeded_max_payout[ethereum] PASSED [ 98%]
tests/test_riskmodule.py::test_exceeded_max_exposure[prototype] PASSED [ 99%]
tests/test_riskmodule.py::test_exceeded_max_exposure[ethereum] PASSED
```

Code Coverage

Quantstamp typically recommends getting branch coverage to 90% or above to reduce the chances of any unexpected behavior in production.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	98.85	89.22	99.07	100	
AccessManager.sol	100	95.83	100	100	
ERC4626AssetManager.sol	100	100	100	100	
EToken.sol	98.64	87.5	100	100	
LPManualWhitelist.sol	100	100	100	100	
LiquidityThresholdAssetManager.sol	100	81.82	100	100	
Policy.sol	100	100	100	100	
PolicyPool.sol	99.13	93.65	96.77	100	
PolicyPoolComponent.sol	91.43	85.71	94.74	100	
PremiumsAccount.sol	100	87.78	100	100	
Reserve.sol	100	87.5	100	100	
RiskModule.sol	100	87.9	100	100	
TimeScaled.sol	96.15	83.33	100	100	
TrustfulRiskModule.sol	100	92.86	100	100	
contracts/interfaces/	100	100	100	100	
IAccessManager.sol	100	100	100	100	
IAssetManager.sol	100	100	100	100	
IEToken.sol	100	100	100	100	
ILPWhitelist.sol	100	100	100	100	
IPolicyHolder.sol	100	100	100	100	
IPolicyPool.sol	100	100	100	100	
IPolicyPoolComponent.sol	100	100	100	100	
IPremiumsAccount.sol	100	100	100	100	
IRiskModule.sol	100	100	100	100	
All files	**98.85**	**89.22**	**99.07**	**100**	

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	100	88.24	100	100	
FlightDelayRiskModule.sol	100	88.24	100	100	
All files	**100**	**88.24**	**100**	**100**	

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	100	100	100	100	
AAVEv2AssetManager.sol	100	100	100	100	
AAVEv3AssetManager.sol	100	100	100	100	
contracts/dependencies/aave-v2/	100	100	100	100	
DataTypes.sol	100	100	100	100	
ILendingPool.sol	100	100	100	100	
ILendingPoolAddressesProvider.sol	100	100	100	100	
contracts/dependencies/aave-v3/	100	100	100	100	
DataTypes.sol	100	100	100	100	
IPool.sol	100	100	100	100	
IPoolAddressesProvider.sol	100	100	100	100	
All files	**100**	**100**	**100**	**100**	

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	98.25	95.45	100	100	
PriceRiskModule.sol	98.25	95.45	100	100	
contracts/interfaces/	100	100	100	100	
IPriceRiskModule.sol	100	100	100	100	
All files	**98.25**	**95.45**	**100**	**100**	

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

```
88652f98e12701c8d5bdb9791eb3340fe188b579fe69c9f0d29b6d15c8a31a88 ./PriceRiskModule.sol
b20bf045ffe71c476b5ea745b1c2ec21bce84c7b4aac17d50626042306b5f770 ./FlightDelayRiskModule.sol
5f6d1cefdc93cb974e2796e933b56a364d1e29b0a5835c9441e4dda3dadad0ad ./AAVEv2AssetManager.sol
8d6b04c83adf0a8eecae97dfcd9aa45dda4864b53a53f9b9e4a6b34521cb3ccb ./AAVEv3AssetManager.sol
18107e5a7bd035343fe744c2c1ae9002401b7ec06fb9b82bc2b18a3642a34c16 ./contracts/PremiumsAccount.sol
2818cdbf3d0cbfc91a2d557651aa50d0b85f12ffafcd5ce851c8274a9aff7266 ./contracts/Policy.sol
5c0b803082a7290ec9d6dcf40f5d20f8111f7ee15fffb69ec486fa40c5ee4280 ./contracts/AccessManager.sol
666b64d294d1407064e89614728dadaaa3c90ef62ab4f2c873759a68928f1bd6 ./contracts/PolicyPoolComponent.sol
5fcf5721b9567826434c61d57fbbd1aec2fcb23368d22f6eaebb0b61d401e499 ./contracts/TrustfulRiskModule.sol
33436ba6cdc182cd4951bd8bb54c2e29c742103fd74bcdc1f7cc2276baf96afb ./contracts/LPManualWhitelist.sol
b3039c356d56359abbc24c99066d45edd01062b6aada65405ff24ecaff801cb3 ./contracts/LiquidityThresholdAssetManager.sol
bde5f98c7f861d3847b147f96ddd1d17b7571bc69b33bfc101ee93c34b741e68 ./contracts/EToken.sol
4de423a10ead85a03d40feb8d8d9dcffc900084dc846fc4f15642ca2177240b4 ./contracts/ERC4626AssetManager.sol
c868f6db83edc1bea1b72348e7b2ff83ef9150533ce301d346ac69d1f087db03 ./contracts/RiskModule.sol
92ee89a0814f74784ced6e56b885c139f9a5a49de9264f86cddb2220361eabd2 ./contracts/TimeScaled.sol
6953e74484c98a9143c7531ae601b6f1b468ef59d1cf69504f430ed66605bdc6 ./contracts/PolicyPool.sol
32e299fcae99da9503f9014927929f6a9b030b1eead7193bb2f7ab0046c4e630 ./contracts/Reserve.sol
20b3fc842ff83559fd19f3fffea6f246f6e854b83adc281e29229dae9196489c ./contracts/interfaces/IPolicyHolder.sol
3e16c34747b0b15d3923501661ea42fcecc6ecd67c69cd7d56a8c07589386de0 ./contracts/interfaces/IPolicyPool.sol
fdeb3b2c7cff587778fa8fdbebf90f664ca6bc4a5fe1d9e844aa44e7e2f55f26 ./contracts/interfaces/IPremiumsAccount.sol
c170699fdb1109469b00ba03fff9b93ad4b49fb61aa78aae288a7c0e21c6109b ./contracts/interfaces/IEToken.sol
af6605ee72643fd4eb94f5eddbb462f8ca36dd95f1657ee03bed9dca2db4825e ./contracts/interfaces/IAssetManager.sol
543f00f4852f123f03f0e6e5bf6b2ebbd13090f527ee416a92b531ab28af5a27 ./contracts/interfaces/IRiskModule.sol
d010996e804b72900f698af7798c30e33d22e44346e652a9661936c50303a141 ./contracts/interfaces/ILPWhitelist.sol
5dbfd416547b7f25b5adb0b1eb4781034625a458528c305729b5c7093991827e ./contracts/interfaces/IAccessManager.sol
27e14b7cbcdc08982aa000024da59d475aed07cb3e897cc16038b5e4c51be461 ./contracts/interfaces/IPolicyPoolComponent.sol
```

Tests

```
88652f98e12701c8d5bdb9791eb3340fe188b579fe69c9f0d29b6d15c8a31a88 ./PriceRiskModule.sol
65230f5f3bd7bdb906dcc984462e9136a8caccee2c6dd9bf2fb59faeba20514f ./test-flight-delay-risk-module.js
bc9d6d558a06a2f62ca143a1bb07811f6d2042a6b150061d7304a48a4e5220ea ./test-storage-gaps.js
f3dff044bc40db063e38cd947b49ff1cbf94a2861fe8b05eca565867364fd34b ./test-aave-asset-manager.js
effcb0643f0f0b111dd1b44bf668bc2eabd185c7c6c13f890f8e77c62d9f27ee ./test-access-manager.js
d5c0a1ce6781eff7a4f25ca826cb447c07f7c682e0fcd39d3afe51763fd6793b ./test-add-remove-change-components.js
846998d6bc180eb2ac52384a2dc5f030b6b6a39963dfb6686602095fe46dd8c0 ./test-constructors.js
24605cf6665959f20b07a160d0a4129c346529634c5d5a77f5dea668d7bbac2a ./test-etoken.js
b719c0f67ddccd7e2567fd615371833e08bffece39199113eb5117fdf9597ae7 ./test-implementation-disabled-init.js
faacba5021fe3b6161c44a0e5f4dadb29550d5a7f81f7e6d75bbc6525f899cfc ./test-initializers.js
5e09e78b28256fe3e5220f4468306956c6140def738847db83262bc39dd95f94 ./test-pause-unpause-permissions.js
db2e977b4e5ef2d247aff94fc10a967e9dddeb4570c716f63795baa4a17b5088 ./test-policy-pool.js
c9626a792cc845d1b2bb3b71af2d8b43dfbca628918443308a09088614ef2afc ./test-policy.js
a29ff17255e1285b59525df44fd9730119808fc2bd183436b35637c1c37fb490 ./test-risk-module.js
fe7c0ea0669facf7f4c86e00bf4430719ad3f0bd67ee8d0181bc590e3e1379b2 ./test-signed-quote-risk-module.js
```

```
7e8d5b416609a1a110bf552c0784c069aeee5eb93b53f35443510ecba5007aed ./test-storage-gaps.js
b7e0277a01f6aa907ec3d74ada7356214c920f6f4fb898a8c76fc7263d8f797d ./test-supports-interface.js
3a846ea02c5da6619cedc259b6cf73c827961545e895a3e948589371a2c3cd3c ./test-upgrade-permissions.js
359abfef4770d2dff975acd608f9187d52c54a33b9e324042f50104f7b517c67 ./test-utils.js
a511cf513a9cadc2f9c909a308c7491eff22f1603f6e01331f655c0f632d43ea ./tests/conftest.py
559052c9e14dd3397e2dc0c3808707d88b18ddb3e103bb33efb8cf853759bc98 ./tests/test_riskmodule.py
1a1814a66a325273885a59da475e3439fc821c6ffdc71669d6344191456be5d5 ./tests/test_fixedratevault.py
9fa5e6f58123e94afe7fe21e2ea62a8019c12d986503056affabe28815ef6a2a ./tests/test_etoken.py
d3291a03822f0ad6fcaa28e67b248e0c5b60c17902d1b163e3a9626629f575b1 ./tests/test_premiumsaccount.py
7eeb90704713542dc78ba5123a44be6ab3fffa3550c6dffcd1f586c4f0c03742 ./tests/test_policypool.py
```

Changelog

- 2022-10-20 - Initial Draft
- 2022-10-24 - Initial Report
- 2022-11-08 - Final Report

About Quantstamp

Quantstamp is a global leader in blockchain security backed by Pantera, Softbank, and Commonwealth among other preeminent investors. Founded in 2017, Quantstamp's mission is to securely onboard the next billion users to Web3 through its white glove security and risk assessment services.

The team consists of web3 thought leaders hailing from top organizations including Microsoft, AWS, BMW, Meta, and the Ethereum Foundation. Many of the auditors hold PhDs or advanced computer science degrees, with decades of combined experience in formal verification, static analysis, blockchain audits, penetration testing, and original leading-edge research.

To date, Quantstamp has performed more than 250 audits and secured over $200 billion in digital asset risk from hackers. In addition to providing an array of security services, Quantstamp facilitates the adoption of blockchain technology through strategic investments within the ecosystem and acting as a trusted advisor to help projects scale.

Quantstamp's collaborations and partnerships showcase our commitment to world-class research, development and security. We're honored to work with some of the top names in the industry and proud to secure the future of web3.

Notable Collaborations & Customers:

- Blockchains: Ethereum 2.0, Near, Flow, Avalanche, Solana, Cardano, Binance Smart Chain, Hedera Hashgraph, Tezos
- DeFi: Curve, Compound, Aave, Maker, Lido, Polygon, Arbitrum, SushiSwap
- NFT: OpenSea, Parallel, Dapper Labs, Decentraland, Sandbox, Axie Infinity, Illuvium, NBA Top Shot, Zora
- Academic institutions: National University of Singapore, MIT