Quantstamp Security Assessment Certificate

September 16th 2020 — Quantstamp Verified

Pie DAO

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

Executive Summary

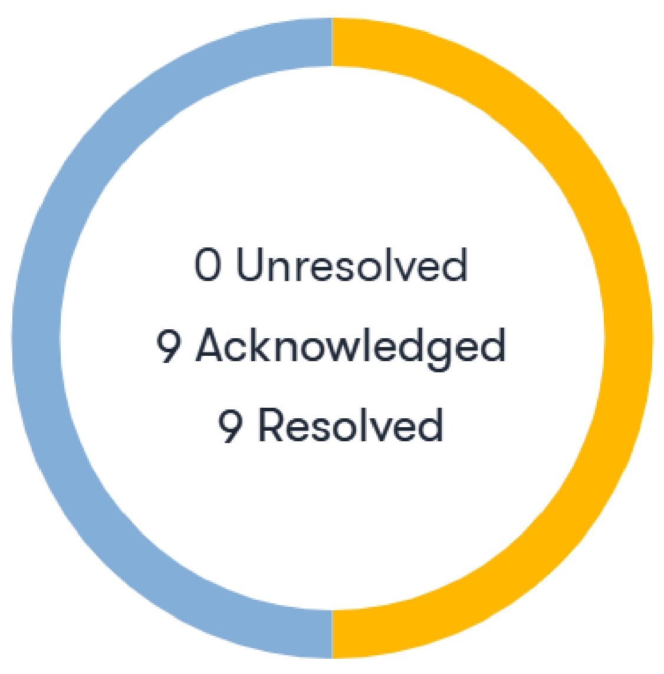
Type	Pooling protocol
Auditors	Ed Zulkoski, Senior Security Engineer Kacper Bąk, Senior Research Engineer Martin Derka, Senior Research Engineer Alex Murashkin, Senior Software Engineer
Timeline	2020-04-08 through 2020-09-16
EVM	Muir Glacier
Languages	Solidity, Javascript
Methods	Architecture Review, Unit Testing, Computer-Aided Verification, Manual Review
Specification	Pie DAO
Source Code	

Repository	Commit
pie-proxy	ece218f
pie-smart-pools	a17fc73

Goals

- Can users' funds get locked up in the contract?
- Can users withdraw their share of underlying tokens?
- May low-level routines be exploited to steal funds from the contracts?

Total Issues	18	(9 Resolved)
High Risk Issues	1	(1 Resolved)
Medium Risk Issues	2	(2 Resolved)
Low Risk Issues	5	(2 Resolved)
Informational Risk Issues	8	(4 Resolved)
Undetermined Risk Issues	2	(0 Resolved)



High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or is reasonably likely to lead to moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low-impact in view of the client's business circumstances.
Informational	The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.
? Undetermined	The impact of the issue is uncertain.

Unresolved	Acknowledged the existence of the risk, and decided to accept it without engaging in special efforts to control it.
Acknowledged	The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).
Resolved	Adjusted program implementation, requirements or constraints to eliminate the risk.
Mitigated	Implemented actions to minimize the impact or likelihood of the risk.

Summary of Findings

The code contains a lot of assembly and low-level constructs. Both obfuscate the intent and make future development, potentially, more error-prone. We have found a few issues ranging from medium to undetermined severity. We have not found any high-severity issues, however. It is important that any external runtime dependencies do not cause the contracts to fail permanently. We recommend resolving the issues we pointed out and making use of regular Solidity constructions instead of low-level code unless there is a compelling reason not to. The scope of this audit was limited to the following files:

- pie-proxy repo: `PProxyPausable.sol`, `PProxy.sol`, `PProxyStorage.sol`;
- pie-smart-pools repo: `PCappedSmartPool.sol`, `PBasicSmartPool.sol`, `ReentryProtection.sol`, `PCToken.sol`, `PCTokenStorage.sol`.

Update: some of our findings were addressed as of commits `96b4ccd` (for pie-proxy) and `677dc19` (for pie-smart-pools). We recommend addressing all our findings before deploying the contracts into production.
Update 2: Quantstamp has reviewed changed to `pie-smart-pools` up to commit b449e80. Several new issues have been appended to each section below. We recommend adding additional inline documentation to make the code more self-contained and easier to follow. As a general caution, since the security of smart pools are intrinsically tied to the security of the underlying tokens/protocols, only trusted tokens should be added to pools.
Update 3: Issues have been resolved as of commit 5d44f96.

ID	Description	Severity	Status
QSP-1	Conflicting code and specification in `calcSingleInGivenPoolOut()`	High	Fixed
QSP-2	`init()` may be called multiple times upon incorrect initialization	Medium	Resolved
QSP-3	Denial-of-Service (DoS)	Medium	Resolved
QSP-4	The function `stringToBytes32()` may convert only part of the string	Low	Resolved
QSP-5	Functions do not check if arguments are non-zero	Low	Acknowledged
QSP-6	Gas Usage / `for` Loop Concerns	Low	Acknowledged
QSP-7	Centralization of Power	Low	Acknowledged
QSP-8	Unresolved TODOs in test suite	Low	Fixed
QSP-9	`setCap()` may emit an event when the cap does not change	Informational	Resolved
QSP-10	Unlocked Pragma	Informational	Acknowledged
QSP-11	Allowance Double-Spend Exploit	Informational	Acknowledged
QSP-12	Clone-and-Own	Informational	Acknowledged
QSP-13	Race Conditions / Front-Running	Informational	Acknowledged
QSP-14	`_setOwner()` may set the `owner` to 0	Informational	Fixed
QSP-15	Gas Usage / `for` Loop Concerns	Informational	Fixed
QSP-16	Unfixed dependency versions	Informational	Fixed
QSP-17	`internalFallback()` may call address without any code and return garbage	? Undetermined	Acknowledged
QSP-18	`setCap()` may set the cap below `totalSupply`	? Undetermined	Acknowledged

Quantstamp Audit Breakdown

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

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

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

Methodology

The Quantstamp auditing process follows a routine series of steps:

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

Toolset

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

Setup

Tool Setup:

- Ganache v1.1.0
- SolidityCoverage v0.5.8
- Mythril v0.2.7
- Securify None
- Slither v0.6.6

Steps taken to run the tools:

1. Installed Ganache: `npm install -g ganache-cli`
2. Installed the solidity-coverage tool (within the project's root directory): `npm install --save-dev solidity-coverage`
3. Ran the coverage tool from the project's root directory: `./node_modules/.bin/solidity-coverage`
4. Installed the Mythril tool from Pypi: `pip3 install mythril`
5. Ran the Mythril tool on each contract: `myth -x path/to/contract`
6. Ran the Securify tool: `java -Xmx6048m -jar securify-0.1.jar -fs contract.sol`
7. Installed the Slither tool: `pip install slither-analyzer`
8. Run Slither from the project directory: `slither .s`

Findings

QSP-1 Conflicting code and specification in `calcSingleInGivenPoolOut()`

Severity: High Risk

Status: Fixed

File(s) affected: `LibPoolMath.sol`

Exploit Scenario: In the comment block associated with `calcSingleInGivenPoolOut()` on L144-153, the denominator is expected to be `(1 - (wI / tW)) * sF)`. However, the equation changes on L173: "//uint256 tAi = tAiAfterFee / (1 - (1-weightTi) * swapFee) ;", in which an extra `1 -` is included in the denominator. The code on L174-175 adheres to the comment on L173, but does not appear to adhere to the earlier comment block. It is not clear which is intended.

Recommendation: Clarify the intended semantics of `calcSingleInGivenPoolOut()` and update the function or docs accordingly.
Update: The function has been verified by the Pie DAO team as correct, and the docs have been updated.

QSP-2 `init()` may be called multiple times upon incorrect initialization

Severity: Medium Risk

Status: Resolved

File(s) affected: `PBasicSmartPool.sol`

Description: The function `init()` does not check that `_bPool` is non-zero. Consequently, if `_bPool` is zero, the function may be called multiple times and overwrite other previously initialized fields.

Recommendation: We recommend adding a `require()` statement checking that `_bPool` is non-zero. Furthermore, upon contract deployment and initialization, we recommend verifying that all fields are set to the expected values, especially because the function `approveTokens()` approves tokens returned by `bPool`.

QSP-3 Denial-of-Service (DoS)

Severity: Medium Risk

Status: Resolved

File(s) affected: `PBasicSmartPool.sol`

Description: A Denial-of-Service (DoS) attack is a situation which an attacker renders a smart contract unusable. If `_pushUnderlying()` fails entirely when any of the token transfers fails, users won't be able to withdraw any of their tokens. One possible scenario when the failure occurs is when `_pushUnderlying()` attempts to transfer 0 `_amount`.

Recommendation: We recommend redesigning the contract in such a way that allows users to pull their tokens individually, i.e., without requiring a loop. Furthermore, for `_amount` equal 0, do not attempt to do the transfer.
Update: the issue is resolved through the use of function `exitPoolTakingloss()`.

QSP-4 The function `stringToBytes32()` may convert only part of the string

Severity: Low Risk

Status: Resolved

File(s) affected: `PProxyStorage.sol`

Description: The function `stringToBytes32()` converts only the first 32 bytes of string to `bytes32`. For longer strings, the remaining part will not be converted.

Recommendation: We recommend documenting the behavior of the function. Furthermore, we recommend adding a check to ensure that the converted string can fit into 32 bytes.

QSP-5 Functions do not check if arguments are non-zero

Severity: Low Risk

Status: Acknowledged

File(s) affected: `PProxy.sol`, `PBasicSmartPool.sol`, `PProxiedFactory.sol`

Description: The functions `PProxy.setProxyOwner()`, `PProxy.setImplementation()`, `PBasicSmartPool.setController()`, `PBasicSmartPool.setPublicSwapSetter()`, `PBasicSmartPool.setTokenBinder()` do not check that arguments of type `address` have non-zero values.
In `PProxiedFactory.sol`, `init()` and `setImplementation()` do not check that arguments are non-zero.
Update: the team informed us that the privileged addresses may be set to 0x0 when all external control is removed from the contracts.

Recommendation: We recommend adding `require()` statements that check if arguments are non-zero.

QSP-6 Gas Usage / `for` Loop Concerns

Severity: Low Risk

Status: Acknowledged

File(s) affected: `PBasicSmartPool.sol`

Description: Gas usage is a main concern for smart contract developers and users, since high gas costs may prevent users from wanting to use the smart contract. Even worse, some gas usage issues may prevent the contract from providing services entirely. For example, if a `for` loop requires too much gas to exit, then it may prevent the contract from functioning correctly entirely. Iteration over `tokens` occurs in lines: 87, 158, 185, and 270.
Update: the team informed us that the underlying Balancer pool has a hard limit of 8 tokens. Consequently, iteration over the loops is was below the block gas limit. Currently joining a 4 token pool costs 618,069 gas and exiting costs 1,150,576 gas, i.e., well within the 10M gas limit. If in a future upgrade this becomes problematic, the team can upgrade the contract through DAO governance to mitigate this potential issue.

Recommendation: We recommend performing gas analysis to find out the limits for which the iteration succeeds. Although, perhaps, unlikely, too many tokens may cause block gas limit exceeded error.

QSP-7 Centralization of Power

Severity: Low Risk

Status: Acknowledged

File(s) affected: `PBasicSmartPool.sol`, `PProxiedFactory.sol`

Description: Smart contracts will often have `owner` variables to designate the person with special privileges to make modifications to the smart contract. Specifically, the contracts may feature centralization of power via the following roles: controller, swap setter, and token binder. Further, the proxy implementation may be upgraded at any time. All these parties are assumed to be trusted.

Recommendation: This centralization of power needs to be made clear to the users, especially depending on the level of privilege the contract allows to the privileged roles.
Update: the team informed us that they will document this contract characteristic.

QSP-8 Unresolved TODOs in test suite

Severity: *Low Risk*

Status: Fixed

File(s) affected: `basicPoolFunctionality.ts`

Description: There are several TODOs in the test suite which primarily relate to checking the balances of all actors in the system. For example, L227,250,288,475,510,568,627 in `basicPoolFunctionality.ts`.

Recommendation: Resolve all TODOs. Ensure that the functionality behaves as intended.

QSP-9 `setCap()` may emit an event when the cap does not change

Severity: *Informational*

Status: Resolved

File(s) affected: `PCappedSmartPool.sol`

Description: The function `setCap()` emits event `CapChanged` even if one passes cap that is equal to `lpcs().cap`. The event is emitted despite the fact that no cap change happens.

Recommendation: We recommend adding a check that the new cap is different from the old cap.
Update: the team informed us that it is not an issue.

QSP-10 Unlocked Pragma

Severity: *Informational*

Status: Acknowledged

File(s) affected: `Several Contracts`

Description: Every Solidity file specifies in the header a version number of the format `pragma solidity (^)0.4.*`. 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: the issue appears to be fixed in pie-smart-pools as of commit `677dc19`. The team informed us that they will update pie-proxy contracts as well. Furthermore, compiler is locked in the builder config.

QSP-11 Allowance Double-Spend Exploit

Severity: *Informational*

Status: Acknowledged

File(s) affected: `PCToken.sol`

Description: As it presently is constructed, the contract is vulnerable to the allowance double-spend exploit, as with other ERC20 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`.

Exploit Scenario:

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.

Recommendation: 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.

QSP-12 Clone-and-Own

Severity: *Informational*

Status: Acknowledged

File(s) affected: `PCToken.sol`

Description: The clone-and-own approach involves copying and adjusting open source code at one's own discretion. From the development perspective, it is initially beneficial as it reduces the amount of effort. However, from the security perspective, it involves some risks as the code may not follow the best practices, may contain a security vulnerability, or may include intentionally or unintentionally modified upstream libraries.

Specifically, `PCToken.sol` contains a modified clone of OpenZeppelin's `SafeMath`. Furthermore, the clone contains custom logic.

Recommendation: Rather than the clone-and-own approach, a good industry practice is to use the Truffle framework for managing library dependencies. This eliminates the clone-and-own risks yet allows for following best practices, such as, using libraries.
We recommend keeping the custom logic apart from the standard `SafeMath` code. Furthermore, we recommend documenting this custom logic.
Update: the team informed us that they intentionally use the same Math functions as Balancer to avoid having any discrepancies between the smart pool and underlying balancer pool.

QSP-13 Race Conditions / Front-Running

Severity: Informational

Status: Acknowledged

File(s) affected: `PCappedSmartPool.sol`

Description: A block is an ordered collection of transactions from all around the network. It's possible for the ordering of these transactions to impact the end result of a block. Specifically, there is a potential race condition between the functions `joinPool()` and `setCap()` due to the modifier `withinCap`.

Recommendation: Race conditions are typically benign issues and are difficult to eliminate. We recommend informing users about the race condition between the functions `joinPool()` and `setCap()`.
Update: they team will document this behavior and will create a proxy contract which can be used to add/remove liquidity to/from any of the PieDAO smart pools. The `setCap()` function is only called by the PieDAO would not be prone to front-running by malicious actors.

QSP-14 `_setOwner()` may set the `owner` to 0

Severity: Informational

Status: Fixed

File(s) affected: `Ownable.sol`

Description: The function `_setOwner()` may take `_newOwner == 0x`. It is not clear if this is intentional.
Update from the Pie DAO team: Behaviour is intended. It allows us to setup the smart pools without any governance.

Recommendation: Clarify whether it should be allowed for the owner to be set to 0. If not, add a require-statement ensuring `_newOwner != address(0)`.
**Update

QSP-15 Gas Usage / `for` Loop Concerns

Severity: Informational

Status: Fixed

File(s) affected: `PProxiedFactory.sol`

Description: Gas usage is a main concern for smart contract developers and users, since high gas costs may prevent users from wanting to use the smart contract. Even worse, some gas usage issues may prevent the contract from providing services entirely. For example, if a `for` loop requires too much gas to exit, then it may prevent the contract from functioning correctly entirely. It is best to break such loops into individual functions as possible.
In `newProxiedSmartPool()`, the `for` loop on L52 iterates over all `_tokens`, performing binds and transfers. If the number of tokens is too large, gas usage could become an issue.

Recommendation: Ensure that the number of tokens passed into the function will not cause gas-limit issues.
Update from the Pie DAO team: Max number of tokens of the underlying balancer pool is 8, which should be well within gas limits.

QSP-16 Unfixed dependency versions

Severity: Informational

Status: Fixed

File(s) affected: `package.json`

Description: Some contracts import contracts from dependencies such as `@pie-dao/mock-contracts` and `@pie-dao/proxy`. If their versions are not fixed, there remains some risk that contracts deviate from the intended functionalities or fail to compile.

Recommendation: Fix dependency versions in `package.json`.

QSP-17 `internalFallback()` may call address without any code and return garbage

Severity: Undetermined

Status: Acknowledged

File(s) affected: `PProxy.sol`

Description: The function `internalFallback()` does not check if `contractAddr` contains any code. Furthermore, the function may return garbage if data returned from the internal call is shorter than the `calldata`, since the function overwrites memory pointed by `ptr` without clearing this memory first. A similar set of deficiencies lead to a temporary shutdown of 0x exchange.

Recommendation: We recommend checking if `contractAddr` contains any code. Furthermore, instead of overwriting, we recommend writing the returned data into a fresh memory slot. Alternatively, clear the memory pointed by `ptr` before overwriting it.
Update: the team informed us that they will not set implementation to non code containing addresses.

QSP-18 `setCap()` may set the cap below `totalSupply`

Severity: Undetermined

Status: Acknowledged

File(s) affected: `PCappedSmartPool.sol`

Description: The function `setCap()` may set the cap that is below `totalSupply`.

Recommendation: Since cap cannot really be lower than `totalSupply`, we recommend disallowing such caps.
Update: the team informed us that this is a feature to effectively only allow withdrawals from the smart pools.

Adherence to Specification

Regarding `LibWeights.sol`, while we did not detect any issues above, there is no specification, and so we cannot assess whether the implementation follows the intent.

Code Documentation

Some of the important pieces of code are not well-documented. For example, `PProxyStorage.sol` shall better document `bytes32ToString()`. **Update:** resolved (function removed).

As of commit b449e80, we noted the following:

- In `LibPoolMath.sol`, on L222: `t0 / tW` should be `w0 / tW`.
- In `Math.sol`, on L20: "diffecerence" should be "difference".
- In `OwnableStorage.sol`, the comment on L10 should say "Load ownable storage".
- In `PCappedSmartPoolStorage.sol` the comment on L11 should say "PCapped" not "PBasic".
- In `ReentryProtectionStorage.sol` the comment on L10 should say "Load reentry protection storage".

Adherence to Best Practices

The code generally adheres to best practices, however,

- the code uses a lot of assembly and low-level constructions which obfuscates the intents and makes it, potentially, more error-prone.
- `PBasicSmartPool`, L204, 225, ignors return value from `transferFrom()`. **Update:** resolved.
- naming could use some improvement. For example, `_denorm` could be `denormalizedWeight`. There should be no shortening of names, e.g., `bPool` => `balancePool`. There is no tax on the number of characters in Solidity.
- `uint256(-1)` should be defined as a constant. It is used in `PBasicSmartPool.sol`, L88, L205, L226.
- in `PBasicSmartPool.sol`, L154, L178, `require()` should specify a reason as a second parameter, to make it clear why a certain condition is required.
- function `PBasicSmartPool.setController()` is marked as `noReentry`, but the other setters are not. This is inconsistent. The functions can be reentered at all. **Update:** resolved.
- for consistency, joining pool should be protected against reentrancy as well. Exiting is. If you expect to work with tokens that contain callbacks, both should be protected. However, the usage seems safe. **Update:** resolved.

As of commit b449e80, we noted the following:

- In `IPV2SmartPool.sol`, `experimental ABIEncoderV2` is unused and can be removed.
- In `IPV2SmartPool.sol`, `PV2SmartPoolStorage` is an unused import and can be removed.
- In `LibAddRemoveToken.sol`, `removeToken()` should fail if the token is not unbound; the behavior of `unbind()` cannot be inferred from the `IBPool` interface.
- In `LibFees.sol`, there is a TODO item on L28. We did not see any issues with the related code in question.
- In `LibFees.chargeOutstandingAnnualFee()`, since `v2s.lastAnnualFeeClaimed = block.timestamp;` is set on all branches, it could set the value before the if-statement (removing a duplicate line of code).
- In `PProxiedFactory.sol`, `init()` could be declared external.
- In `PV2SmartPool.sol`, `approveTokens()` could be declared external.

*

Test Results

Test Suite Results

```
  Advanced Pool Functionality
    updateWeight()
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ Updating the weigth from a non controller should fail
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ Updating down should work (387ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ Updating down while the token transfer returns false should fail (96ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ Updating down while not having enough pool tokens should fail (155ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ Updating up should work (376ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ Updating up while not having enough of the underlying should fail (104ms)
WARNING: Multiple definitions for exitPool
```

```
WARNING: Multiple definitions for joinPool
       ✓ Updating up while the token transferFrom returns false should fail (79ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
       ✓ Updating up while the underlying token is not approved should fail (81ms)
    updateWeightsGradually()
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
       ✓ Updating from a non controller should fail
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
       ✓ Updating should work (426ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
       ✓ Setting a start block in the past should set it to the current block (448ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
       ✓ Updating the weight of a token above the max should fail
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
       ✓ Updating the weight of a token below the minimum should fail
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
       ✓ Updating the weights above the total max weight should fail
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
       ✓ Updating to a start block which is bigger before the end block should fail (49ms)
    pokeWeight()
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
       ✓ Poking the weights up should work (3237ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
       ✓ Poking the weights down should work (3251ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
       ✓ Poking the weight after the end block should work (2089ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
       ✓ Poking the weight twice after the end block should fail (7612ms)
      Adding tokens
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
         ✓ Weight update should cancel when removing token (7059ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
         ✓ Weight update should cancel when adding token (8976ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
         ✓ Weight update should cancel when calling bind (8357ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
         ✓ Weight update should cancel when calling unbind (11734ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
         ✓ Weight update should cancel when calling rebind (8514ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
         ✓ Weight update should cancel when calling updateWeight (down) (8861ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
         ✓ Weight update should cancel when calling updateWeight (up) (8319ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
         ✓ commitAddToken should work (125ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
         ✓ commitAddToken from a non controller should fail (90ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
         ✓ Apply add token should work (734ms)
      removeToken
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
         ✓ removeToken should work (446ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
         ✓ removeToken should fail when controller does not have enough pool tokens (172ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
         ✓ removeToken should fail if underlying token transfer returns false (107ms)
      Setting joining and exiting enabled
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
         ✓ setJoinExitEnabled should work (85ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
         ✓ setJoinExitEnabled from a non controller address should fail (88ms)
      Circuit Breaker
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
         ✓ setCircuitBreaker should work (85ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
         ✓ setCircuitBreaker from a non controller should fail (87ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
         ✓ tripCircuitBreaker should work (1109ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
         ✓ tripCircuitBreaker from a non circuitbreaker address should fail
      Join exit disabled enforcement
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
         ✓ joinPool
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
         ✓ joinPool with front running protection
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
         ✓ exitPool
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
         ✓ exitPool with frontrunning protection
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
         ✓ exitPoolTakingLoss
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
         ✓ joinswapExternAmountIn (61ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
         ✓ joinswapPoolAmountOut
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
         ✓ exitswapPoolAmountIn
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
         ✓ exitswapExternAmountOut
      Annual Fee
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
         ✓ Setting the fee should work (106ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
         ✓ Setting the fee from a non controller should fail (83ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
         ✓ Setting the fee too high (10%) should fail (93ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
         ✓ Setting the fee recipient should work (93ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
         ✓ Setting the fee recipient from a non controller should fail (89ms)

  Basic Pool Functionality
    init
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
```

```
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ Initialising with invalid bPool address should fail (931ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ Initialising with zero supply should fail (1137ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ Token symbol should be correct
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ Token name should be correct
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ Initial supply should be correct
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ Controller should be correctly set
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ Public swap setter should be correctly set
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ Token binder should be correctly set
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ bPool should be correctly set
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ Tokens should be correctly set
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ calcTokensForAmount should work (70ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ Calling init when already initialized should fail
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ Smart pool should not hold any non balancer pool tokens after init (77ms)
    Controller functions
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ Setting a new controller should work (93ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ Setting a new controller from a non controller address should fail
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ Setting public swap setter should work (90ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ Setting public swap setter from a non controller address should fail
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ Setting the token binder should work (112ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ Setting the token binder from a non controller address should fail
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ Setting public swap should work (406ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ Setting public swap from a non publicSwapSetter address should fail
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ Setting the swap fee should work (419ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ Setting the swap fee from a non controller address should fail
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ Should revert with unsupported function error when calling finalizePool()
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ Should revert with unsupported function error when calling createPool(uint256 initialSupply)
    Joining and Exiting
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ Adding liquidity should work (1077ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ Adding liquidity when a transfer fails should fail (140ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ Adding liquidity when a token transfer returns false should fail (336ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ Removing liquidity should work (1794ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ Removing all liquidity should fail (231ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ Removing liquidity should fail when removing more than balance (1043ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ Removing liquidity when a token transfer fails should fail (210ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ Removing liquidity when a token transfer returns false should fail (139ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ Removing liquidity leaving a single token should work (1014ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ Removing all liquidity leaving a single token should fail (98ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ Removing liquidity leaving a single token should fail when removing more than balance (1098ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ Should fail to join with a single token if token is unbound (632ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ joinswapPoolAmountOut should work (2361ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ joinswapExternAmountIn should work (2732ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ Joining the pool from a single asset when public swap is disabled should fail
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ Should fail to exit with a single token if token is unbound (773ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ exitswapPoolAmountIn should work (2300ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ exitSwapExternAmountOut should work (2224ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ Exiting the pool to a single asset when public swap is disabled should fail (42ms)
    Front running protected join and exit
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ Adding liquidity with frontrunning protection should work should work (1301ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ Adding liquidity with front running protection when maxAmount of one of the tokens is too small should fail (182ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ Adding liquidity with front running protection when a transfer fails should fail (139ms)
```

```
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ Adding liquidity with front running protection when a token transfer returns false should fail (193ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ Removing liquidity with front running protection should work (1226ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ Removing liquidity with front running protection should fail when one of the token outputs is less than minAmount (143ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ Removing all liquidity with front running protection should fail (56ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ Removing liquidity with front running protection should fail when removing more than balance (1004ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ Removing liquidity with front running protection when a token transfer fails should fail (111ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ Removing liquidity with frontrunning protection when a token transfer returns false should fail (104ms)
    Token binding
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ Binding a new token should work (1801ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ Binding a token when transferFrom returns false should fail (271ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ Binding from a non token binder address should fail (303ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ Rebinding a token should work (277ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ Rebinding a token reducing the balance should work (240ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ Rebinding a token reducing the balance when the the token token transfer returns false should fail (100ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ Rebinding a token from a non token binder address should fail
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ Unbinding a token should work (294ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ Unbinding a token from a non token binder address should fail
    ready modifier
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ should revert when not ready (972ms)
    lockBPoolSwap modifier
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ If swap disabled, keep disabled (1241ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
      ✓ If swap enabled, keep enabled (7564ms)
    Utility Functions
      getDenormalizedWeight(address _token)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
        ✓ Should return denormalized weight of underlying token in bPool

  Cap
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
    ✓ Cap should initially zero
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
    ✓ Setting the cap should work (87ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
    ✓ Setting the cap from a non controller address should fail (106ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
    ✓ JoinPool with less than the cap should work (1283ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
    ✓ JoinPool with more than the cap should fail (457ms)
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
    ✓ joinswapExternAmountIn with less than the cap should work (2821ms)

  poolToken
    token metadata
      ✓ Should have 18 decimals
      ✓ Token name should be correct
      ✓ Symbol should be correct
      ✓ Initial supply should be zero
      ✓ After minting total supply should go up by minted amount (188ms)
      ✓ Burning tokens should lower the total supply (195ms)
      ✓ Burning more than an address's balance should fail (108ms)
    balanceOf
      ✓ Should return zero if no balance
      ✓ Should return correct amount if account has some tokens (100ms)
    transfer
      ✓ Should fail when the sender does not have enought balance (103ms)
      ✓ Sending the entire balance should work (182ms)
      ✓ Should emit transfer event (162ms)
      ✓ Sending 0 tokens should work (242ms)
    approve
      ✓ Should emit event (59ms)
      ✓ Should work when there was no approved amount before (72ms)
      ✓ Should work when there was a approved amount before (132ms)
      ✓ Setting approval back to zero should work (127ms)
    increaseApproval
      ✓ Should emit event (237ms)
      ✓ Should work when there was no approved amount before (95ms)
      ✓ Should work when there was an approved amount before (142ms)
      ✓ Increasing approval beyond max uint256 should fail (73ms)
    decreaseApproval
      ✓ Should emit event (65ms)
      ✓ Decreasing part of the approval should work (72ms)
      ✓ Decreasing the entire approval should work (75ms)
      ✓ Decreasing more than the approval amount should set approval to zero (67ms)
    transferFrom
      ✓ Should emit event (157ms)
      ✓ Should work when sender has enough balance and approved spender (202ms)
      ✓ Should fail when not enough allowance is set (83ms)
      ✓ Should fail when sender does not have enough balance (74ms)
      ✓ Should not change approval amount when it was set to max uint256 (152ms)

  PProxiedFactory
WARNING: Multiple definitions for exitPool
WARNING: Multiple definitions for joinPool
    ✓ Creating a new proxied pool should work (1034ms)

  ReentryProtection
    ✓ Should prevent reentry

  LibSafeApproval
    ✓ Doing double approvals which are not "safe" should fail
    ✓ Doing double approvals which are "safe" should work (112ms)


  159 passing (22m)
```

Code Coverage

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	97.78	91.67	94.74	97.92	
Ownable.sol	80	50	66.67	83.33	19
PCToken.sol	100	100	100	100	
ReentryProtection.sol	100	100	100	100	
contracts/factory/	96.55	50	66.67	96.55	
PProxiedFactory.sol	96.55	50	66.67	96.55	31
contracts/interfaces/	100	100	100	100	
IBFactory.sol	100	100	100	100	
IBPool.sol	100	100	100	100	
IERC20.sol	100	100	100	100	
IPV2SmartPool.sol	100	100	100	100	
contracts/libraries/	20.87	20.59	34.62	21.01	
LibAddRemoveToken.sol	0	0	0	0	… 82,85,87,89
LibConst.sol	100	100	100	100	
LibFees.sol	85.71	83.33	100	85.71	41,43,45
LibPoolEntryExit.sol	18.69	9.38	23.08	19.09	… 284,285,287
LibPoolMath.sol	0	100	0	0	… 390,391,393
LibPoolToken.sol	100	100	100	100	
LibSafeApprove.sol	100	100	100	100	
LibUnderlying.sol	50	25	50	50	19,21,23,27
LibWeights.sol	0	0	0	0	… 151,154,159
Math.sol	43.94	34.21	58.33	45.31	… 143,147,151
contracts/smart-**pools/**	97.48	88.89	95.45	97.64	
PV2SmartPool.sol	97.48	88.89	95.45	97.64	476,752,768
contracts/storage/	100	100	100	100	
OwnableStorage.sol	100	100	100	100	
PBasicSmartPoolStorage.sol	100	100	100	100	
PCTokenStorage.sol	100	100	100	100	
PCappedSmartPoolStorage.sol	100	100	100	100	
PV2SmartPoolStorage.sol	100	100	100	100	
ReentryProtectionStorage.sol	100	100	100	100	
contracts/test/	100	100	100	100	
TestLibSafeApprove.sol	100	100	100	100	
TestPCToken.sol	100	100	100	100	
TestReentryProtection.sol	100	100	100	100	
All files	**46.96**	**38.71**	**74.51**	**48.37**	

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

```
62d9e04145c8dd223608a9c7ae23b053a0d82fbc97baba9bff78cf8f82c239fb ./contracts/Ownable.sol

e41b6303f30944ad10b2313fda18c1ae8fca7b52d2d1e55f4d7c9d324598d9a7 ./contracts/ReentryProtection.sol

f0c0fc5ff9240db193daf6764bd0ed5eb19111f8866e95d2b1159980b7adfde3 ./contracts/PCToken.sol

f4abccfb14e0205425b3c1a7bba5756afe7776c3a613b34251a1fb4a7edf15a1 ./contracts/interfaces/IERC20.sol

a0e16d092f278ebdf80816d5019a7b3350bdbd12ff2f6d9008ff18a624014b2e ./contracts/interfaces/IBFactory.sol

dbfac0ba904d4ad25e461a8f4ac08621983875975d92f901e976c3eb5497c8b4 ./contracts/interfaces/IPV2SmartPool.sol

afb2534c828e4b2679dc12045ce6b1303fc97266466e5694b8c49cc7640a1317 ./contracts/interfaces/IBPool.sol

bd9a3f8f809f393846cf2df7aa39ff1a87547e7c7d6b7e9993576a08465cc45a ./contracts/factory/PProxiedFactory.sol

da26b84d0e17f554d5a719d637f246a47d09d2c4ff624dd443d47070a93d059c ./contracts/smart-pools/PV2SmartPool.sol

bc36376a74c8e50ccd19114721fe51058ef60bb52777654651ef332788ebf4b4 ./contracts/storage/PV2SmartPoolStorage.sol

c72c8e1d8cd4c89264756639691134407a7c6d3a83720772beb87f51f3f67def ./contracts/storage/PBasicSmartPoolStorage.sol

ceabff237c8d9b841ef6ba77419778495abe3a67bfb3f7c7cf110926eef94647 ./contracts/storage/PCappedSmartPoolStorage.sol

3c10ded2d71a6470a44413bc3bb32f0c61ea724cb9cebf007323dbeb44e044c4 ./contracts/storage/PCTokenStorage.sol

be7c4012900d059574a12df07bd521da28322aa85099ba9f5e11654f887de62e ./contracts/storage/ReentryProtectionStorage.sol

24875089ac9b82414fd2afb38f8abaaeb3e4469ae94a1b124723c2ecfae3bf5d ./contracts/storage/OwnableStorage.sol

7aa998423bcb72e820466f892acbadb4f4387099b36d3a4dc2a5750094c1200d ./contracts/libraries/LibPoolEntryExit.sol

9399b3bd669bbb3fba372829addc0d2c715626476474f71f59f107ab6118e7ed ./contracts/libraries/LibPoolToken.sol

814fa125931274a3d7a5af4013b13b7da07d38caf2cf55a795a18c3f9dab1801 ./contracts/libraries/LibAddRemoveToken.sol

94a002401ab8bf6214664c03ee6a59852c59db07de2bdcef4fcde92f087b417d ./contracts/libraries/LibConst.sol

239dbfbd37c6b49d2ce4a7b241f453d1a2203e1164862f3f12b9dd7dffd8c294 ./contracts/libraries/LibFees.sol

98987edd751c1d41239cb319e96e28b88e7a74f86f7f08c1e923f8d7ccd08e7e ./contracts/libraries/LibPoolMath.sol

830b3ec311e1d744821485c9c2701dd26519faad2ac37c96380ecc93ec6a2b20 ./contracts/libraries/Math.sol

43e2bc56e2706b1f9605cc5e1e3ccaacbcd53da10271d443dddf581a71c2437e ./contracts/libraries/LibUnderlying.sol

6ece5d1c7eced4f1fe498c4d81a2a823da24b5b9ced72779dfe7454216fdbb55 ./contracts/libraries/LibWeights.sol

d34c84ad6895b67902143204f20c74830590f1f2bd3cd5d2082b09d22ab267b6 ./contracts/libraries/LibSafeApprove.sol

bb6857ee8b87aea1ccb759ddba7b05c15b82af32df356eb38b4d1049e8993f5e ./contracts/test/TestLibSafeApprove.sol

96c14d10c6878acb10306562c192c4db8c25b9e34f59df27ab17f7dbe36da9a2 ./contracts/test/TestPCToken.sol

e8afd8a9374cea65e42cb2a9c75021d6e92266aafe844bea5aab61a81ab0c32a ./contracts/test/TestReentryProtection.sol
```

Tests

```
1959f7a0d64b8f4769d0282a047e8e46965e83e5d72dca1394eee85fad17aebb ./test/advancedPoolFunctionality.ts

73e40735dcc18a085fc2255d8c6d6948275ce395c9376acba661ada6626941c1 ./test/capPoolFunctionality.ts

4631c8dbf2321de6c4d791803cb9a29ae3a367fbd5d45b428c000335150f60f1 ./test/reentryProtection.ts

8f62bfc4490ddb6ab5f3844e2617b8cd63f9ef927028c07c23d9d676f3396249 ./test/pProxiedFactory.ts

d68f56329847198c261c7cda00d47e1567f8f67773c8f52f69f3478849b681dc ./test/basicPoolFunctionality.ts

5a6473b59d735e096661d2a401b21ca1e335234a932afd9143db619286583bef ./test/poolToken.ts

5b71623c50c112a1c79f079e568a9c4cb05a6536e5c16857e5c6b034ed0a903d ./test/safeApproval.ts
```

Changelog

- 2020-04-17 - Initial report
- 2020-04-28 - Report revised based on commits 96b4ccd (for pie-proxy) and 677dc19 (for pie-smart-pools)
- 2020-04-30 - Added responses from the team.
- 2020-08-28 - Report revised based on commit b449e80
- 2020-09-16 - 5d44f96

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.