



Smart Contract Security Assessment

06-23-2022

Prepared for
Pirex

Online report
[pirex-multi-token-convex-wrapper](#)

Multi Token Convex Wrapper Security Audit

Audit Overview

We were tasked with performing an audit of the Pirex codebase and in particular their multi-token Convex wrapper ecosystem permitting users to wrap their Convex tokens based on voting epochs.

Over the course of the audit, we identified some important flaws relating to the mathematical calculations around rewards & redemptions as well as certain blockchain-specific misbehaviours that could negatively impact the project and its users.

We advise the Pirex team to closely evaluate all minor-and-above findings identified in the report and promptly remediate them as well as consider all optimizational exhibits identified in the report.

Post-Audit Conclusion

The Pirex team has provided an alleviation for each exhibit in the form of an extensive PDF outlining the rationale behind some of the exhibits that required no action.

We re-assessed all exhibits based on newfound information provided to us by the Pirex team and have updated each finding's status accordingly. Overall, all findings have either been dealt with or sufficiently acknowledged.

Contracts Assessed

Files in Scope	Repository	Commit(s)
ERC1155Solmate.sol (ERC)	pirex 	0200b31558, 032604740a, 968a2bd187, 0c15e0adb7, 9468ac0317
ERC20SnapshotSolmate.sol (ERS)	pirex 	0200b31558, 032604740a, 968a2bd187, 0c15e0adb7, 9468ac0317
ERC1155PresetMinterSupply.sol (ERP)	pirex 	0200b31558, 032604740a, 968a2bd187, 0c15e0adb7, 9468ac0317
FixedPointMathLib.sol (FPM)	pirex 	0200b31558, 032604740a, 968a2bd187, 0c15e0adb7, 9468ac0317
PxCvx.sol (PCX)	pirex 	0200b31558, 032604740a, 968a2bd187, 0c15e0adb7, 9468ac0317
PirexCvx.sol (PCV)	pirex 	0200b31558, 032604740a, 968a2bd187, 0c15e0adb7, 9468ac0317
PirexFees.sol (PFS)	pirex 	0200b31558, 032604740a, 968a2bd187, 0c15e0adb7, 9468ac0317
PirexCvxConvex.sol (PCC)	pirex 	0200b31558, 032604740a, 968a2bd187, 0c15e0adb7, 9468ac0317

UnionPirexVault.sol (UPV)	pirex		0200b31558, 032604740a, 968a2bd187, 0c15e0adb7, 9468ac0317
---------------------------	-------	---	--

Audit Synopsis

Severity	Identified	Alleviated	Partially Alleviated	Acknowledged
● Major	1	1	0	0
● Medium	7	5	0	2
● Minor	9	7	0	2
● Informational	9	7	0	2

During the audit, we filtered and validated a total of **2 findings utilizing static analysis** tools as well as identified a total of **24 findings during the manual review** of the codebase. We strongly recommend that any minor severity or higher findings are dealt with promptly prior to the project's launch as they introduce potential misbehaviours of the system as well as exploits.

pie title Total Issues "Major" : 1 "Medium" : 7 "Minor" : 9 "Informational" : 9

Compilation

The project utilizes `hardhat` as its development pipeline tool, containing an array of tests and scripts coded in TypeScript.

To compile the project, the `compile` command needs to be issued via the `npx` CLI tool to `hardhat`:

BASH

```
npx hardhat compile
```

The `hardhat` tool automatically selects Solidity version `0.8.12` based on the version specified within the `hardhat.config.ts` file for the set of contracts in scope of the audit.

The project contains discrepancies with regards to the Solidity version used, however, they are solely contained in dependencies and can be safely ignored.

The `pragma` statements have been locked to `0.8.12`, the same version utilized for our static analysis as well as optimizational review of the codebase.

During compilation with the `hardhat` pipeline, no errors were identified that relate to the syntax or bytecode size of the contracts.

Static Analysis

The execution of our static analysis toolkit identified **804 potential issues** within the codebase of which **803 were ruled out to be false positives** or negligible findings.

The remaining **1 issues** were validated and grouped and formalized into the **2 exhibits** that follow:

ID	Severity	Addressed	Title
PCV-01S	Medium	Yes	Improper Invocation of EIP-20 Approve Function
PFS-01S	Informational	Acknowledged	Literal Equality of <code>bool</code> Variable

Manual Review

A thorough line-by-line review was conducted on the codebase to identify potential malfunctions and vulnerabilities in the Pirex multi-token Convex integrating codebase.

As the project at hand implements a complex Convex-integrating wrapper protocol, intricate care was put into ensuring that the **flow of funds within the system conforms to the specifications and restrictions** laid forth within the protocol's specification as well as Convex's documentation and deployed code.

We validated that **all state transitions of the system occur within sane criteria** and that all rudimentary formulas within the system execute as expected. We **pinpointed multiple important flaws** within the system which could have had **severe ramifications** to its overall operation, however, they were conveyed ahead of time to the Pirex team to be **promptly remediated**.

Additionally, the system was investigated for any other commonly present attack vectors such as re-entrancy attacks, mathematical truncations, logical flaws and **ERC / EIP** standard inconsistencies. The documentation of the project was satisfactory to a certain extent, however, we strongly recommend the documentation of the project to be expanded at certain complex points such as the in-line description of each of the project's token types as well as standardized documentation of **external**-facing functions.

A total of **24 findings** were identified over the course of the manual review of which **17 findings** concerned the behaviour and security of the system. The non-security related findings, such as optimizations, are included in the separate **Code Style** chapter.

The finding table below enumerates all these security / behavioural findings:

ID	Severity	Addressed	Title
ERP-01M	Minor	Yes	Misleading Contract Documentation
ERS-01M	Medium	Yes	Insecure Elliptic Curve Recovery Mechanism
PCV-01M	Major	Yes	Incorrect Proportionate Claim Mechanism
PCV-02M	Medium	Acknowledged	Improper Prevention of Emergency Migration

ID	Severity	Addressed	Title
PCV-03M	Medium	Nullified	Mismatch of Maximum Redemption Time
PCV-04M	Minor	Acknowledged	Disassociation of Votium Time Rewards
PCV-05M	Minor	Yes	Improper Sanitization of New Fee
PCV-06M	Minor	Acknowledged	Inexistent Migration Delay
PCV-07M	Minor	Yes	Inexistent Prevention of Duplicate Entries
PCV-08M	Minor	Nullified	Potentially Harmful Epoch Rounding
PCV-09M	Informational	Yes	Unaudited External Contract Interaction
PCC-01M	Medium	Yes	Improper Adjustment of Delegation Space & Registry
PCC-02M	Medium	Nullified	Incorrect Usage of Funds at Rest
PCC-03M	Medium	Acknowledged	Unlimited CVX Allowance Centralization
PCC-04M	Minor	Yes	Unsafe & Inefficient Type
PFS-01M	Minor	Yes	Improper & Inefficient Calculation of Fees
PCX-01M	Minor	Yes	Snapshot Initialization Race Condition

PirexCvx Static Analysis Findings

PCV-01S: Improper Invocation of EIP-20 Approve Function

Type	Severity	Location
Standard Conformity	Medium	PirexCvx.sol:L281, L284

Description:

The linked statement does not properly validate the returned `bool` of the **EIP-20** standard `approve` function. As the **standard dictates**, callers **must not** assume that `false` is never returned.

Example:

```
contracts/PirexCvx.sol

SOL

280 if (address(unionPirex) != address(0)) {
281     pxCvx.approve(address(unionPirex), 0);
282 }
283 unionPirex = UnionPirexVault(contractAddress);
```

Recommendation:

Since not all standardized tokens are **EIP-20** compliant (such as Tether / USDT), we advise the already imported `SafeTransferLib` safe wrapper library to be utilized instead to opportunistically validate the returned `bool` only if it exists.

Alleviation:

The code was updated to utilize the `safeApprove` implementation of the `solmate` library thus ensuring that it properly evaluates the yielded `bool` for errors.

PirexFees Static Analysis Findings

PFS-01S: Literal Equality of `bool` Variable

Type	Severity	Location
Gas Optimization	Informational	PirexFees.sol:L75

Description:

The linked `bool` comparison is performed between a variable and a `bool` literal.

Example:

```
contracts/PirexFees.sol
SOL
75 if (hasRole(FEE_DISTRIBUTOR_ROLE, distributor) == false)
```

Recommendation:

We advise the `bool` variable to be utilized directly either in its negated (`!`) or original form.

Alleviation:

The Pirex team has stated that they wish to retain the current code in place. We would like to note that this is **the boolean-equal static analyser finding of slither**, and relates to code style rather than gas optimization. In any case, we consider this exhibit acknowledged as per the Pirex team's request.

ERC1155PresetMinterSupply Manual Review Findings

ERP-01M: Misleading Contract Documentation

Type	Severity	Location
Standard Conformity	Minor	ERC1155PresetMinterSupply.sol:L22-L24

Description:

The linked documentation of the contract is misleading as it states that the deployer of the contract will acquire the `pauser` role, a role that does not exist in the system.

Example:

```
contracts/ERC1155PresetMinterSupply.sol

SOL

12 /**
13  * @dev {ERC1155} token, including:
14 *
15  * - ability to check the total supply for a token id
16  * - ability for holders to burn (destroy) their tokens
17  * - a minter role that allows for token minting (creation)
18 *
19 * This contract uses {AccessControl} to lock permissioned functions using
20 * different roles - head to its documentation for details.
21 *
22 * The account that deploys the contract will be granted the minter and pauser
23 * roles, as well as the default admin role, which will let it grant both minter
24 * and pauser roles to other accounts.
25 *
26 * _Deprecated in favor of https://wizard.openzeppelin.com/\[Contracts Wizard\]
27 */
28 contract ERC1155PresetMinterSupply is
29     Context,
30     AccessControlEnumerable,
31     ERC1155Supply,
32     ERC1155Burnable
33 {
```

```
34     bytes32 public constant MINTER_ROLE = keccak256("MINTER_ROLE");
35
36     /**
37      * @dev Grants `DEFAULT_ADMIN_ROLE`, `MINTER_ROLE`, and `PAUSER_ROLE` to
38      * deploys the contract.
39      */
40     constructor(string memory uri) ERC1155(uri) {
41         _setupRole(DEFAULT_ADMIN_ROLE, _msgSender());
42
43         _setupRole(MINTER_ROLE, _msgSender());
44     }
```

Recommendation:

We advise the documentation to be corrected as it is currently misleading.

Alleviation:

The code was updated to no longer reference the `PAUSER_ROLE` in the general `contract` and `constructor` in-line documentation.

ERC20SnapshotSolmate Manual Review Findings

ERS-01M: Insecure Elliptic Curve Recovery Mechanism

Type	Severity	Location
Language Specific	Medium	ERC20SnapshotSolmate.sol:L181, L182

Description:

The `ecrecover` function is a low-level cryptographic function that should be utilized after appropriate sanitizations have been enforced on its arguments, namely on the `s` and `v` values. This is due to the inherent trait of the curve to be symmetrical on the x-axis and thus permitting signatures to be replayed with the same `x` value (`r`) but a different `y` value (`s`).

Example:

```
contracts/ERC20SnapshotSolmate.sol

SOL

148 function permit(
149     address owner,
150     address spender,
151     uint256 value,
152     uint256 deadline,
153     uint8 v,
154     bytes32 r,
155     bytes32 s
156 ) public virtual {
157     require(deadline >= block.timestamp, "PERMIT_DEADLINE_EXPIRED");
158
159     // Unchecked because the only math done is incrementing
160     // the owner's nonce which cannot realistically overflow.
161     unchecked {
162         address recoveredAddress = ecrecover(
163             keccak256(
164                 abi.encodePacked(
165                     "\x19\x01",
166                     DOMAIN_SEPARATOR(),
167                     keccak256(
168                         abi.encodePacked(
169                             owner,
170                             spender,
171                             value,
172                             deadline,
173                             v,
174                             r,
175                             s
176                         )
177                     )
178                 )
179             )
180         );
181         if (recoveredAddress != owner) revert PermitOwnerMismatch();
182     }
183 }
```

```

167     keccak256(
168         abi.encode(
169             keccak256(
170                 "Permit(address owner,address spender,uint256 deadline,
171                 uint8 v,
172                 bytes32 r,
173                 bytes32 s
174             ) ,
175             owner,
176             spender,
177             value,
178             nonces[owner]++,
179             deadline
180         )
181     );
182
183     require(
184         recoveredAddress != address(0) && recoveredAddress == owner,
185         "INVALID_SIGNER"
186     );
187
188     allowance[recoveredAddress][spender] = value;
189 }
190
191 emit Approval(owner, spender, value);
192 }
```

Recommendation:

We advise them to be sanitized by ensuring that `v` is equal to either `27` or `28` (`v ∈ {27, 28}`) and to ensure that `s` is existent in the lower half order of the elliptic curve (`0 < s < secp256k1n ÷ 2 + 1`) by ensuring it is less than `0x7FFFFFFFFFFFFFFF5D576E7357A4501DDFE92F46681B20A1`. A reference implementation of those checks can be observed in the **ECDSA** library of OpenZeppelin and the rationale behind those restrictions exists within **Appendix F of the Yellow Paper**.

Alleviation:

The `recover` function from the **ECDSA** OpenZeppelin library is now properly used in the codebase preventing the inherent cryptographic attack described from being applicable to the contract.

PirexCvx Manual Review Findings

PCV-01M: Incorrect Proportionate Claim Mechanism

Type	Severity	Location
Mathematical Operations	Major	PirexCvx.sol:L891-L933

Description:

The `redeemFuturesRewards` mechanism will fail to function properly when more than one parties have token futures for a particular epoch as the total supply is reduced via the `burn` mechanism whilst the `futuresRewards` stored in `pxCvx` remain the same.

Impact:

Any user beyond the first will claim a disproportionate amount of rewards, a trait that can be exploited. Additionally, a portion of users will be unable to claim rewards at all as their disproportionate rewards would not be fulfillable by the ERC20 transfer the function executes.

Example:

```
contracts/PirexCvx.sol

SOL

891 /**
892     @notice Redeem Futures rewards for rpCVX holders for an epoch
893     @param epoch      uint256 Epoch (ERC1155 token id)
894     @param receiver   address Receives futures rewards
895 */
896 function redeemFuturesRewards(uint256 epoch, address receiver)
897     external
898     whenNotPaused
899     nonReentrant
900 {
901     if (epoch == 0) revert InvalidEpoch();
902     if (epoch > getCurrentEpoch()) revert InvalidEpoch();
903     if (receiver == address(0)) revert ZeroAddress();
904 }
```

```

905     // Prevent users from burning their futures notes before rewards are cl
906     (, bytes32[] memory rewards, , uint256[] memory futuresRewards) = pxCvx
907     .getEpoch(epoch);
908
909     if (rewards.length == 0) revert NoRewards();
910
911     emit RedeemFuturesRewards(epoch, receiver, rewards);
912
913     // Check sender rpCVX balance
914     uint256 rpCvxBalance = rpCvx.balanceOf(msg.sender, epoch);
915     if (rpCvxBalance == 0) revert InsufficientBalance();
916
917     // Store rpCVX total supply before burning
918     uint256 rpCvxTotalSupply = rpCvx.totalSupply(epoch);
919
920     // Burn rpCVX tokens
921     rpCvx.burn(msg.sender, epoch, rpCvxBalance);
922
923     uint256 rLen = rewards.length;
924
925     // Loop over rewards and transfer the amount entitled to the rpCVX token
926     for (uint256 i; i < rLen; ++i) {
927         // Proportionate to the % of rpCVX owned out of the rpCVX total supply
928         ERC20(address(uint160(bytes20(rewards[i])))).safeTransfer(
929             receiver,
930             (futuresRewards[i] * rpCvxBalance) / rpCvxTotalSupply
931         );
932     }
933 }
```

Recommendation:

We advise the `futuresRewards` entry to also be updated within `pxCvx` to subtract the claimed proportionate rewards ensuring that consequent percentage calculations are done correctly.

Alleviation:

The local `futuresRewards` is now properly subtracted in the `for` loop operation and an external call to `updateEpochFuturesRewards` is also performed ensuring that the `futuresRewards` are synchronized in the relevant contract, alleviating this exhibit in full.

PCV-02M: Improper Prevention of Emergency Migration

Type	Severity	Location
Logical Fault	Medium	PirexCvx.sol:L323, L326

Description:

The logic within the `emergencyMigrateTokens` will fail to execute properly if the `outstandingRedemptions` match the currently held `cvx` tokens exactly as in such a case the consequent `revert` statement will execute as the migration `amount` will be zero.

Impact:

The emergency migration function will not be executable thereby locking the tokens within the contract. The scenario whereby the `outstandingRedemptions` match the `amount` held by the contract is likely as users panic in emergency scenarios and would request redemptions at an alarming pace.

Example:

```
contracts/PirexCvx.sol

SOL

318 for (uint256 i; i < tLen; ++i) {
319     uint256 amount = ERC20(tokens[i]).balanceOf(address(this));
320
321     // If it's CVX, make sure we take into account the outstandingRedemptions
322     if (tokens[i] == address(CVX)) {
323         amount -= outstandingRedemptions;
324     }
325
326     if (amount == 0) revert ZeroAmount();
327
328     amounts[i] = amount;
329     ERC20(tokens[i]).safeTransfer(pirexCvxMigration, amount);
330 }
```

Recommendation:

We advise the `if (amount == 0) revert ZeroAmount();` statement to be relocated before the special `cvx` logic. Alternatively, we advise the duplicate token entry finding to be

assimilated to the codebase that relocates the entire `cvx` logic outside the `for` loop and alleviates this exhibit as well.

Alleviation:

The Pirex team evaluated this exhibit and stated that this is an intentional design decision as users with outstanding redemptions should still be able to redeem their tokens using the legacy redemption path. After highlighting the `emergency` word usage to their team, Pirex responded by stating that the `emergency` convention is not utilized to indicate a failure scenario of the system but rather an abnormal circumstance of the protocols the contract interfaces with (namely Convex). Given the intentional design of the code we consider this exhibit acknowledged.

PCV-03M: Mismatch of Maximum Redemption Time

Type	Severity	Location
Standard Conformity	Medium	PirexCvx.sol:L76

Description:

The maximum redemption time present on the Convex V2 Locker instance is 16 weeks rather than the 17 weeks detailed in the variable's comment as well as represented by the variable itself.

Impact:

All redemptions will be unfairly penalized with a fee greater than the minimum regardless of whether the maximum unlock time has been specified.

Example:

```
contracts/PirexCvx.sol
SOL
75 // Maximum wait time (seconds) for a CVX redemption (17 weeks)
76 uint32 public constant MAX_REDEMPTION_TIME = 10281600;
```

Recommendation:

We advise this trait to be re-evaluated and if a fee increase is expected at all times the trait to be documented as otherwise the minimum fee for redemptions will be always unattainable.

Alleviation:

In light of supplemental material provided by the Pirex team, we concluded that our original finding's rationale is invalid given that all locks performed by the Pirex system are "fresh locks" from the `CvxLockerV2` perspective. As a result, the maximum redemption time specified in the contract is correct and we consider the exhibit nullified.

PCV-04M: Disassociation of Votium Time Rewards

Type	Severity	Location
Logical Fault	Minor	PirexCvx.sol:L741, L752-L754, L772-L779

Description:

The Votium rewards that are submitted to the contract are then meant to be distributed to users based on the currently active epoch, however, the Votium rewards themselves are not time-restrained with regards to their claim and the contract also meets a fatal failure in case any of the claims fail to execute properly. This can cause significant issues such as race-conditions to arise whereby a user detects a high amount of Votium claims seconds before an epoch ends, submits their funds for claim-age on the next epoch, hijacks the Votium claim mechanism, and ultimately claims a disproportionate amount of rewards for as little as a few seconds of lock time for their funds.

Impact:

Any reward submissions of more than one member can be hijacked via a race-condition by submitting a claim of the first reward with a higher priority fee thereby causing the transaction including all rewards to fail as the first one will not be claimable. In time-based edge case scenarios, a user would be able to exploit this fact to momentarily stake their `pxCvx` to the contract, acquire a disproportionate amount of futures, and unlock their stake in the next few seconds at no penalty.

Example:

```
contracts/PirexCvx.sol

SOL

737 /**
738     @notice Claim multiple Votium rewards
739     @param votiumRewards VotiumRewards[] Votium rewards metadata
740 */
741 function claimVotiumRewards(VotiumReward[] calldata votiumRewards)
742     external
743     whenNotPaused
744     nonReentrant
745 {
746     uint256 tLen = votiumRewards.length;
747     if (tLen == 0) revert EmptyArray();
748 }
```

```
749     // Take snapshot before claiming rewards, if necessary
750     pxCvx.takeEpochSnapshot();
751
752     uint256 epoch = getCurrentEpoch();
753     (uint256 snapshotId, , , ) = pxCvx.getEpoch(epoch);
754     uint256 rpCvxSupply = rpCvx.totalSupply(epoch);
755
756     for (uint256 i; i < tLen; ++i) {
757         address token = votiumRewards[i].token;
758         uint256 index = votiumRewards[i].index;
759         uint256 amount = votiumRewards[i].amount;
760         bytes32[] memory merkleProof = votiumRewards[i].merkleProof;
761
762         if (token == address(0)) revert ZeroAddress();
763         if (amount == 0) revert ZeroAmount();
764
765         emit ClaimVotiumReward(token, index, amount);
766
767         ERC20 t = ERC20(token);
768
769         // Used for calculating the actual token amount received
770         uint256 prevBalance = t.balanceOf(address(this));
771
772         // Validates `token`, `index`, `amount`, and `merkleProof`
773         votiumMultiMerkleStash.claim(
774             token,
775             index,
776             address(this),
777             amount,
778             merkleProof
779         );
780
781         (
782             uint256 rewardFee,
783             uint256 snapshotRewards,
784             uint256 futuresRewards
785         ) = _calculateRewards(
786             fees[Fees.Reward],
787             pxCvx.totalSupplyAt(snapshotId),
788             rpCvxSupply,
789             t.balanceOf(address(this)) - prevBalance
790         );
791
792         // Add reward token address and snapshot/futuresRewards amounts (sa
793         pxCvx.addEpochRewardMetadata(
794             epoch,
795             token.fillLast12Bytes(),
796             snapshotRewards,
797             futuresRewards
798         );

```

```
799  
800     // Distribute fees  
801     t.safeApprove(address(pirexFees), rewardFee);  
802     pirexFees.distributeFees(address(this), token, rewardFee);  
803 }  
804 }
```

Recommendation:

Firstly, we advise the claim process to go through even if one of the `votiumRewards` has already been claimed. This will prevent claim hijacking from malicious actors. Lastly, we advise some form of time association to be enforced for Votium claims. As an example, the contract could distribute the rewards for the previous epoch rather than the current one thereby disallowing edge-case scenarios of transaction submissions close to an epoch's end date.

Alleviation:

The Pirex team has stated that this is not a concern as users should be able to stake for one round and acquire their respective rewards. We would like to note that the real concern here is that the user is not obligated to stake for a full round; they are able to stake close to the end of a particular round and claim in the next thus staking their tokens only for mere seconds. This might cause a profitable attack scenario to unfold. The Pirex team reconsidered this exhibit based on our supplemental information and desired to acknowledge it.

PCV-05M: Improper Sanitization of New Fee

Type	Severity	Location
Input Sanitization	Minor	PirexCvx.sol:L351

Description:

While the `newFee` is validated to be within the denominator bounds, it is not validated to be different than the existing one.

Impact:

Queueing the same fee will cause multiple abnormal events to be emitted, will ultimately result in a no-op, and does not protect against multiple pending transactions extending the `effectiveAfter` timestamp.

Example:

```
contracts/PirexCvx.sol

SOL

345 /**
346     @notice Queue fee
347     @param f      enum Fee
348     @param newFee uint32 New fee
349 */
350 function queueFee(Fees f, uint32 newFee) external onlyOwner {
351     if (newFee > FEE_DENOMINATOR) revert InvalidNewFee();
352
353     uint224 effectiveAfter = uint224(block.timestamp + EPOCH_DURATION);
354
355     // Queue up the fee change, which can be set after 2 weeks
356     queuedFees[f].newFee = newFee;
357     queuedFees[f].effectiveAfter = effectiveAfter;
358
359     emit QueueFee(f, newFee, effectiveAfter);
360 }
361
362 /**
363     @notice Set fee
364     @param f  enum Fee
365 */
366 function setFee(Fees f) external onlyOwner {
367     QueuedFee memory q = queuedFees[f];
```

```
368  
369     if (q.effectiveAfter > block.timestamp)  
370         revert BeforeEffectiveTimestamp();  
371  
372     fees[f] = q.newFee;  
373  
374     emit SetFee(f, q.newFee);  
375 }
```

Recommendation:

We advise the `if` conditional to additionally evaluate that the `newFee` is different than the current `fees[f]` data entry.

Alleviation:

Additional sanitization was introduced ensuring that the `newFee` is not equal to the `fees[f]` on successful execution and alleviating this exhibit.

PCV-06M: Inexistent Migration Delay

Type	Severity	Location
Centralization Concern	Minor	PirexCvx.sol:L307

Description:

The emergency migration mechanism of the contract is available for at-will invocation by the owner as they can invoke the `setPirexCvxMigration` and `emergencyMigrateTokens` in a single chain of transactions.

Example:

```
contracts/PirexCvx.sol

SOL

287 /**
288     @notice Set pirexCvxMigration address
289     @param _pirexCvxMigration address PirexCvxMigration address
290 */
291 function setPirexCvxMigration(address _pirexCvxMigration)
292     external
293     onlyOwner
294     whenPaused
295 {
296     if (_pirexCvxMigration == address(0)) revert ZeroAddress();
297
298     pirexCvxMigration = _pirexCvxMigration;
299
300     emit SetPirexCvxMigration(pirexCvxMigration);
301 }
302
303 /**
304     @notice Withdraw ERC20 tokens to the pirexCvxMigration address in case
305     @param tokens address[] Token addresses
306 */
307 function emergencyMigrateTokens(address[] calldata tokens)
308     external
309     onlyOwner
310     whenPaused
311 {
312     if (pirexCvxMigration == address(0)) revert ZeroAddress();
313
314     uint256 tLen = tokens.length;
```

```
315     if (tLen == 0) revert EmptyArray();
316
317     uint256[] memory amounts = new uint256[] (tLen);
318
319     for (uint256 i; i < tLen; ++i) {
320         uint256 amount = ERC20(tokens[i]).balanceOf(address(this));
321
322         // If it's CVX, make sure we take into account the outstandingRedemptions
323         if (tokens[i] == address(CVX)) {
324             amount -= outstandingRedemptions;
325         }
326
327         if (amount == 0) revert ZeroAmount();
328
329         amounts[i] = amount;
330         ERC20(tokens[i]).safeTransfer(pirexCvxMigration, amount);
331     }
332
333     emit MigrateTokens(pirexCvxMigration, tokens, amounts);
334 }
```

Recommendation:

We advise a time delay to be enforced instead to ensure users have adequate time to withdraw their tokens manually if they are able to. To achieve this, a dedicated user-oriented emergency withdrawal mechanism should be available at all times regardless of the pause status of the contract, similarly to SushiSwap.

Alleviation:

The Pirex team has acknowledged this exhibit and stated that the migration actions will sit behind a multisignature of yet-to-be-confirmed reputable parties in the blockchain space thus ensuring that the protocol's owners cannot single-handedly manipulate the protocol. We consider this sufficient acknowledgement of the exhibit.

PCV-07M: Inexistent Prevention of Duplicate Entries

Type	Severity	Location
Input Sanitization	Minor	PirexCvx.sol:L321-L324

Description:

The `tokens` array can contain duplicate entries and as such permit the caller to withdraw more than `amount - outstandingRedemptions` for the `cvx` token in particular, circumventing the redemption protection mechanism.

Impact:

The `cvx` tokens can be improperly extracted from the contract disallowing it to fulfill outstanding redemptions and causing significant impact to its users.

Example:

```
contracts/PirexCvx.sol

SOL

303 /**
304     @notice Withdraw ERC20 tokens to the pirexCvxMigration address in case
305     @param tokens address[] Token addresses
306 */
307 function emergencyMigrateTokens(address[] calldata tokens)
308     external
309     onlyOwner
310     whenPaused
311 {
312     if (pirexCvxMigration == address(0)) revert ZeroAddress();
313
314     uint256 tLen = tokens.length;
315     if (tLen == 0) revert EmptyArray();
316     uint256[] memory amounts = new uint256[] (tLen);
317
318     for (uint256 i; i < tLen; ++i) {
319         uint256 amount = ERC20(tokens[i]).balanceOf(address(this));
320
321         // If it's CVX, make sure we take into account the outstandingRede
322         if (tokens[i] == address(CVX)) {
323             amount -= outstandingRedemptions;
324         }
325     }
```

```
326     if (amount == 0) revert ZeroAmount();
327
328     amounts[i] = amount;
329     ERC20(tokens[i]).safeTransfer(pirexCvxMigration, amount);
330 }
331
332 emit MigrateTokens(pirexCvxMigration, tokens, amounts);
333 }
```

Recommendation:

We advise the `cvx` token to be prevented from being present in the `tokens` array via a `require` check and it to be emptied with its special logic outside the `for` loop.

Alleviation:

The Pirex team has significantly adjusted the emergency migration mechanism to be composed of a two-party system with the configuration parameters of the emergency execution being adjusted by the Pirex team's multisignature wallet while the actual execution of the migration procedure is meant to be conducted by a dedicated, non-Pirex party that is publicly declared on-chain. While on-chain sanitization of the duplicate token entries is still advised, the current implementation in place can be considered a sufficient alleviation **as long as the actors of the system act faithfully.**

PCV-08M: Potentially Harmful Epoch Rounding

Type	Severity	Location
Mathematical Operations	Minor	PirexCvx.sol:L377-L383, L680-L711, L713-L735

Description:

All epoch calculations that utilize `getCurrentEpoch` contain a rounding-down operation that can cause harmful side-effects. Such a side-effect illustrated in the linked lines is the ability to stake `pxCvx` for one round and unstake a few seconds later if the current `block.timestamp` is close to the rounding threshold. As a result, a user can acquire a significant amount of `_mintFutures` unfairly by momentarily staking their funds and can cause reward distributions to malfunction.

Impact:

As locks are evaluated on an epoch basis rather than a timestamp basis, it is possible to perform a "full epoch" lock in a few seconds if timed properly via on-chain transactions that execute before and after the end point of an epoch.

Example:

```
contracts/PirexCvx.sol

SOL

680 /**
681     @notice Stake pCVX
682     @param rounds      uint256    Rounds (i.e. Convex voting rounds)
683     @param f            enum      Futures enum
684     @param assets       uint256   pCVX amount
685     @param receiver     address   Receives spCVX
686 */
687 function stake(
688     uint256 rounds,
689     Futures f,
690     uint256 assets,
691     address receiver
692 ) external whenNotPaused nonReentrant {
693     if (rounds == 0) revert ZeroAmount();
694     if (assets == 0) revert ZeroAmount();
695     if (receiver == address(0)) revert ZeroAddress();
696
697     // Burn pCVX
```

```

698     pxCvx.burn(msg.sender, assets);
699
700     emit Stake(rounds, f, assets, receiver);
701
702     // Mint spCVX with the stake expiry timestamp as the id
703     spCvx.mint(
704         receiver,
705         getCurrentEpoch() + EPOCH_DURATION * rounds,
706         assets,
707         UNUSED_1155_DATA
708     );
709
710     _mintFutures(rounds, f, assets, receiver);
711 }
712
713 /**
714  * @notice Unstake pCVX
715  * @param id          uint256 spCVX id (an epoch timestamp)
716  * @param assets      uint256 spCVX amount
717  * @param receiver    address Receives pCVX
718 */
719 function unstake(
720     uint256 id,
721     uint256 assets,
722     address receiver
723 ) external whenNotPaused nonReentrant {
724     if (id > block.timestamp) revert BeforeStakingExpiry();
725     if (assets == 0) revert ZeroAmount();
726     if (receiver == address(0)) revert ZeroAddress();
727
728     // Mint pCVX for receiver
729     pxCvx.mint(receiver, assets);
730
731     emit Unstake(id, assets, receiver);
732
733     // Burn spCVX from sender
734     spCvx.burn(msg.sender, id, assets);
735 }
```

Recommendation:

We advise the rounding approach to be re-evaluated and a ceiling rather than flooring approach to be used for calculating the current epoch. Alternatively, we advise the impact of this mechanism to be evaluated on a case-by-case basis (such as the `stake` / `unstake` misbehaviour), the impact to be documented properly and wherever needed safeguards to be put in place.

Alleviation:

The Pirex team has stated that this is desired behaviour as the only requirement for staking for a particular round is to contain sufficient pxCVX tokens. As a result, we consider this exhibit nullified.

PCV-09M: Unaudited External Contract Interaction

Type	Severity	Location
Standard Conformity	Informational	PirexCvx.sol:L83

Description:

The linked contract utilized for the Votium-related reward mechanisms is unaudited.

Example:

```
contracts/PirexCvx.sol
SOL
83 IVotiumMultiMerkleStash public votiumMultiMerkleStash;
```

Recommendation:

We advise its integration to the system to be performed with caution as it can contain unidentified flaws as well as undocumented future features that can compromise the integration of the contract with the Votium system.

Alleviation:

The Pirex team has stated that they will proceed with including the dependency without performing an audit for it as they deem it secure enough based on on-chain utilization.

PirexCvxConvex Manual Review Findings

PCC-01M: Improper Adjustment of Delegation Space & Registry

Type	Severity	Location
Logical Fault	Medium	PirexCvxConvex.sol:L108-L111, L195-L208

Description:

The delegation space & registry can be adjusted without nullifying previously delegated voting power and as such potentially cause improper delegation of power when it is overwritten improperly.

Impact:

Delegation will be improperly retained in the original delegation space which may ultimately be malicious.

Example:

```
contracts/PirexCvxConvex.sol

SOL

195 /**
196     @notice Set delegationSpace
197     @param _delegationSpace string Convex Snapshot delegation space
198 */
199 function setDelegationSpace(string memory _delegationSpace)
200     external
201     onlyOwner
202 {
203     bytes memory d = bytes(_delegationSpace);
204     if (d.length == 0) revert EmptyString();
205     delegationSpace = bytes32(d);
206
207     emit SetDelegationSpace(_delegationSpace);
208 }
```

Recommendation:

We advise the `setDelegationSpace` & `setConvexContract` functions to invoke the `clearDelegate` function of `cvxDelegateRegistry` should a delegation exist, a trait that can be validated via the `delegation` public function of the said contract.

Alleviation:

After a secondary consideration, the Pirex team adjusted the `setDelegationSpace` code to accept an additional argument that specifies whether any previously set delegation should be cleared via the existing `clearVoteDelegate` function. As a result, we consider the exhibit fully alleviated.

PCC-02M: Incorrect Usage of Funds at Rest

Type	Severity	Location
Logical Fault	Medium	PirexCvxConvex.sol:L143-L145

Description:

The `_lock` function is meant to calculate how many funds it is supposed to lock in the CVX Locker, however, it incorrectly calculates them leading to unfulfillable redemptions.

Impact:

Users will not be able to fulfill their outstanding redemptions as the pending locks of the system "overpower" the condition of `outstandingRedemptions` being higher than the current balance of the contract.

Example:

```
contracts/PirexCvxConvex.sol

SOL

123 /**
124     @notice Unlock CVX and relock excess
125 */
126 function _lock() internal {
127     _unlock();
128
129     uint256 balance = CVX.balanceOf(address(this));
130     bool balanceGreaterThanRedemptions = balance > outstandingRedemptions;
131
132     // Lock CVX if the balance is greater than outstanding redemptions or if
133     if (balanceGreaterThanRedemptions || pendingLocks != 0) {
134         uint256 balanceRedemptionsDifference = balanceGreaterThanRedemption
135             ? balance - outstandingRedemptions
136             : 0;
137
138         // Lock amount is the greater of the two: balanceRedemptionsDifference
139         // balanceRedemptionsDifference is greater if there is unlocked CVX
140         // pendingLocks is greater if there are more new deposits than unlock
141         cvxLocker.lock(
142             address(this),
143             balanceRedemptionsDifference > pendingLocks
144                 ? balanceRedemptionsDifference
145                 : pendingLocks,
```

```
146          0  
147      );  
148  
149      pendingLocks = 0;  
150  }  
151 }
```

Recommendation:

The `_lock` function logic should be refactored to reduce `pendingLocks` by the amount of funds that are within `outstandingRedemptions`. As such, if `outstandingRedemptions` match the `pendingLocks` then only excess funds should be locked rather than `pendingLocks`.

Alleviation:

The Pirex team has evaluated our exhibit and deemed invalid based on their internal fuzz tests as well as general test suites. As a result, we consider this exhibit nullified per the client's request.

PCC-03M: Unlimited CVX Allowance Centralization

Type	Severity	Location
Centralization Concern	Medium	PirexCvxConvex.sol:L104

Description:

The `setConvexContract` function permits the owner to set an unlimited `cvx` allowance to an arbitrary address, a trait we strongly advise against.

Impact:

Should the owner of the system be compromised, they are capable of completely compromising the system in relatively few actions.

Example:

```
contracts/PirexCvxConvex.sol
SOL
87 /**
88  * @notice Set a contract address
89  * @param c enum Contract
90  * @param contractAddress address Contract
91 */
92 function setConvexContract(ConvexContract c, address contractAddress)
93     external
94     onlyOwner
95 {
96     if (contractAddress == address(0)) revert ZeroAddress();
97
98     emit SetConvexContract(c, contractAddress);
99
100    if (c == ConvexContract.CvxLocker) {
101        // Revoke approval from the old locker and add allowances to the new
102        CVX.safeApprove(address(cvxLocker), 0);
103        cvxLocker = ICvxLocker(contractAddress);
104        CVX.safeApprove(contractAddress, type(uint256).max);
105    }
106
107    if (c == ConvexContract.CvxDelegateRegistry) {
108        cvxDelegateRegistry = ICvxDelegateRegistry(contractAddress);
109        return;
110    }
111}
```

```
111      }  
112 }
```

Recommendation:

We advise some alternative migration methodologies to be utilized instead, such as by ensuring any locker changes are accompanied by a time delay to allow sufficient time for users to extract their funds from the system.

Alleviation:

The Pirex team has acknowledged this exhibit and stated that the administrative actions of the ecosystem will sit behind a multisignature of reputable team members thus ensuring that each protocol owner cannot single-handedly manipulate the protocol. We consider this sufficient acknowledgement of the exhibit.

PCC-04M: Unsafe & Inefficient Type

Type	Severity	Location
Mathematical Operations	Minor	PirexCvxConvex.sol:L174, L178

Description:

The linked statements utilize the `uint8` data type and perform an unsafe cast of the reward array yielded by Convex.

Impact:

While relatively unlikely, should the total available rewards of Convex exceed 255 entries the cast operation will overflow silently causing the system to misbehave.

Example:

```
contracts/PirexCvxConvex.sol
SOL
160 /**
161     @notice Get claimable rewards and balances
162     @return rewards ConvexReward[] Claimable rewards and balances
163 */
164 function _claimableRewards()
165     internal
166     view
167     returns (ConvexReward[] memory rewards)
168 {
169     address addr = address(this);
170
171     // Get claimable rewards
172     ICvxLocker.EarnedData[] memory c = cvxLocker.claimableRewards(addr);
173
174     uint8 cLen = uint8(c.length);
175     rewards = new ConvexReward[] (cLen);
176
177     // Get the current balances for each token to calculate the amount rece
178     for (uint8 i; i < cLen; ++i) {
179         rewards[i] = ConvexReward({
180             token: c[i].token,
181             amount: c[i].amount,
182             balance: ERC20(c[i].token).balanceOf(addr)
183         });
184 }
```

```
184      }  
185 }
```

Recommendation:

We advise the cast and corresponding loop iterator type to be adjusted to `uint256` as any sub-256-bit data type incurs extra gas on the EVM.

Alleviation:

The original `uint256` data type is now utilized as advised.

PirexFees Manual Review Findings

PFS-01M: Improper & Inefficient Calculation of Fees

Type	Severity	Location
Mathematical Operations	Minor	PirexFees.sol:L147

Description:

The fee system of `PirexFees` does not take into account the innate mathematical truncation that arithmetics boast in the EVM and as such will lead to funds improperly extracted from the `from` account whenever `distributeFees` is invoked and truncation occurs, leading to uncaptured fees.

Example:

```
contracts/PirexFees.sol

SOL

122 /**
123     @notice Distribute fees
124     @param from    address Fee source
125     @param token   address Fee token
126     @param amount  uint256 Fee token amount
127 */
128 function distributeFees(
129     address from,
130     address token,
131     uint256 amount
132 ) external onlyRole(FEE_DISTRIBUTOR_ROLE) {
133     emit DistributeFees(token, amount);
134
135     ERC20 t = ERC20(token);
136
137     // Favoring push over pull to reduce accounting complexity for differen
138     t.safeTransferFrom(
139         from,
140         treasury,
141         (amount * treasuryPercent) / PERCENT_DENOMINATOR
142     );
```

```
143
144     t.safeTransferFrom(
145         from,
146         contributors,
147         (amount * contributorsPercent) / PERCENT_DENOMINATOR
148     );
149 }
```

Recommendation:

We advise the system to instead contain a single variable for denoting the fees of either the treasury or contributors as the remaining amount is meant to be the amount designated for the other party. Additionally, the code within `distributeFees` should be adjusted to transfer the remaining amount to the secondary party by simply subtracting the calculated fees of the primary party from the total `amount` rather than performing a new multiplication and division thus accounting for any truncation that may occur.

Alleviation:

The fee calculations as well as relevant setter functions and sanitizations have been adjusted as advised ensuring an optimal truncation-accounting fee structure and the required maintenance of a single variable over two, alleviating this exhibit in full.

PxCvx Manual Review Findings

PCX-01M: Snapshot Initialization Race Condition

Type	Severity	Location
Language Specific	Minor	PxCvx.sol:L63-L64, L198-L208

Description:

The snapshot of the `currentEpoch` can be written to twice under an edge case whereby the `takeEpochSnapshot` function is invoked before the first `operator` has been set.

Impact:

The snapshot taken for the particular epoch will be incorrectly replaced thereby yielding a different snapshot for the same epoch.

Example:

```
contracts/PxCvx.sol

SOL

198 /**
199      @notice Snapshot token balances for the current epoch
200 */
201 function takeEpochSnapshot() external onlyOperatorOrNotPaused {
202     uint256 currentEpoch = getCurrentEpoch();
203
204     // If snapshot has not been set for current epoch, take snapshot
205     if (epochs[currentEpoch].snapshotId == 0) {
206         epochs[currentEpoch].snapshotId = _snapshot();
207     }
208 }
```

Recommendation:

We advise this scenario to be prohibited by ensuring the `onlyOperatorOrNotPaused` modifier also validates that the `operator` it reads from storage already is not equal to the zero-

address (`address(0)`). Alternatively, the contract should be deployed in a paused state immediately to prevent this scenario by setting the pause status in its `constructor`.

Alleviation:

The code has been updated to include an operator check in the `onlyOperatorOrNotPaused` modifier as advised thereby alleviating this exhibit.

Finding Types

A description of each finding type included in the report can be found below and is linked by each respective finding. A full list of finding types Omniscia has defined will be viewable at the central audit methodology we will publish soon.

External Call Validation

Many contracts that interact with DeFi contain a set of complex external call executions that need to happen in a particular sequence and whose execution is usually taken for granted whereby it is not always the case. External calls should always be validated, either in the form of `require` checks imposed at the contract-level or via more intricate mechanisms such as invoking an external getter-variable and ensuring that it has been properly updated.

Input Sanitization

As there are no inherent guarantees to the inputs a function accepts, a set of guards should always be in place to sanitize the values passed in to a particular function.

Indeterminate Code

These types of issues arise when a linked code segment may not behave as expected, either due to mistyped code, convoluted `if` blocks, overlapping functions / variable names and other ambiguous statements.

Language Specific

Language specific issues arise from certain peculiarities that the Solidity language boasts that discerns it from other conventional programming languages. For example, the EVM is a 256-bit machine meaning that operations on less-than-256-bit types are more costly for the EVM in terms of gas costs, meaning that loops utilizing a `uint8` variable because their limit will never exceed the 8-bit range actually cost more than redundantly using a `uint256` variable.

Code Style

An official Solidity style guide exists that is constantly under development and is adjusted on each new Solidity release, designating how the overall look and feel of a codebase should be. In these types of findings, we identify whether a project conforms to a particular naming convention and whether that convention is consistent within the codebase and legible. In case of inconsistencies, we point them out under this category. Additionally, variable shadowing falls under this category as well which is identified when a local-level variable contains the same name as a contract-level variable that is present in the inheritance chain of the local execution level's context.

Gas Optimization

Gas optimization findings relate to ways the codebase can be optimized to reduce the gas cost involved with interacting with it to various degrees. These types of findings are completely optional and are pointed out for the benefit of the project's developers.

Standard Conformity

These types of findings relate to incompatibility between a particular standard's implementation and the project's implementation, oftentimes causing significant issues in the usability of the contracts.

Mathematical Operations

In Solidity, math generally behaves differently than other programming languages due to the constraints of the EVM. A prime example of this difference is the truncation of values during a division which in turn leads to loss of precision and can cause systems to behave incorrectly when dealing with percentages and proportion calculations.

Logical Fault

This category is a bit broad and is meant to cover implementations that contain flaws in the way they are implemented, either due to unimplemented functionality, unaccounted-for edge cases or similar extraordinary scenarios.

Centralization Concern

This category covers all findings that relate to a significant degree of centralization present in the project and as such the potential of a Single-Point-of-Failure (SPoF) for the project that we urge them to re-consider and potentially omit.

Reentrant Call

This category relates to findings that arise from re-entrant external calls (such as EIP-721 minting operations) and revolve around the inapplicacy of the Checks-Effects-Interactions (CEI) pattern, a pattern that dictates checks (`require` statements etc.) should occur before effects (local storage updates) and interactions (external calls) should be performed last.

Disclaimer

The following disclaimer applies to all versions of the audit report produced (preliminary / public / private) and is in effect for all past, current, and future audit reports that are produced and hosted under Omnicia:

IMPORTANT TERMS & CONDITIONS REGARDING OUR SECURITY AUDITS/REVIEWS/REPORTS AND ALL PUBLIC/PRIVATE CONTENT/DELIVERABLES

Omnicia ("Omnicia") has conducted an independent security review to verify the integrity of and highlight any vulnerabilities, bugs or errors, intentional or unintentional, that may be present in the codebase that were provided for the scope of this Engagement.

Blockchain technology and the cryptographic assets it supports are nascent technologies. This makes them extremely volatile assets. Any assessment report obtained on such volatile and nascent assets may include unpredictable results which may lead to positive or negative outcomes.

In some cases, services provided may be reliant on a variety of third parties. This security review does not constitute endorsement, agreement or acceptance for the Project and technology that was reviewed. Users relying on this security review should not consider this as having any merit for financial advice or technological due diligence in any shape, form or nature.

The veracity and accuracy of the findings presented in this report relate solely to the proficiency, competence, aptitude and discretion of our auditors. Omnicia and its employees make no guarantees, nor assurance that the contracts are free of exploits, bugs, vulnerabilities, deprecation of technologies or any system / economical / mathematical malfunction.

This audit report shall not be printed, saved, disclosed nor transmitted to any persons or parties on any objective, goal or justification without due written assent, acquiescence or approval by Omnicia.

All the information/opinions/suggestions provided in this report does not constitute financial or investment advice, nor should it be used to signal that any person reading this report should invest their funds without sufficient individual due diligence regardless of the findings presented in this report.

Information in this report is provided 'as is'. Omniscia is under no covenant to the completeness, accuracy or solidity of the contracts reviewed. Omniscia's goal is to help reduce the attack vectors/surface and the high level of variance associated with utilizing new and consistently changing technologies.

Omniscia in no way claims any guarantee, warranty or assurance of security or functionality of the technology that was in scope for this security review.

In no event will Omniscia, its partners, employees, agents or any parties related to the design/creation of this security review be ever liable to any parties for, or lack thereof, decisions and/or actions with regards to the information provided in this security review.

Cryptocurrencies and all other technologies directly or indirectly related to cryptocurrencies are not standardized, highly prone to malfunction and extremely speculative by nature. No due diligence and/or safeguards may be insufficient and users should exercise maximum caution when participating and/or investing in this nascent industry.

The preparation of this security review has made all reasonable attempts to provide clear and actionable recommendations to the Project team (the "client") with respect to the rectification, amendment and/or revision of any highlighted issues, vulnerabilities or exploits within the contracts in scope for this engagement.

All services, the security reports, discussions, work product, attack vectors description or any other materials, products or results of this security review engagement is provided "as is" and "as available" and with all faults, uncertainty and defects without warranty or guarantee of any kind.

Omniscia will assume no liability or responsibility for delays, errors, mistakes, or any inaccuracies of content, suggestions, materials or for any loss, delay, damage of any kind which arose as a result of this engagement/security review.

Omniscia will assume no liability or responsibility for any personal injury, property damage, of any kind whatsoever that resulted in this engagement and the customer having access to or use of the products, engineers, services, security report, or any other other materials.

For avoidance of doubt, this report, its content, access, and/or usage thereof, including any associated services or materials, shall not be considered or relied upon as any form of financial, investment, tax, legal, regulatory, or any other type of advice.