OMNISCIA

May 28, 2022

SMART CONTRACT AUDIT REPORT

Pirex Multi-Token Convex Wrapper

omniscia.io

info@omniscia.io

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

Files in Scope	Repository	Commit(s)
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.

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 `bool` 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
PCV-03M	Medium	Nullified	Mismatch of Maximum Redemption Time

ID	Severity	Addressed	Title
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

Code Style

During the manual portion of the audit, we identified **7 optimizations** that can be applied to the codebase that will decrease the gas-cost associated with the execution of a particular function and generally ensure that the project complies with the latest best practices and standards in Solidity.

Additionally, this section of the audit contains any opinionated adjustments we believe the code should make to make it more legible as well as truer to its purpose.

These optimizations are enumerated below:

ID	Severity	Addressed	Title
ERS-01C	Informational	Acknowledged	Inefficient `mapping` Lookups
ERS-02C	Informational	Yes	Variable Read Optimization
PCV-01C	Informational	Yes	Illegible Numeric Value Representation
PCV-02C	Informational	Yes	Inefficient Storage Value Reads
PCV-03C	Informational	Yes	Suboptimal Numeric Representation
PCC-01C	Informational	Yes	Inexplicable `if-if` Structure
PCC-02C	Informational	Yes	Redundant Return Statements

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

```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

```sol
/**
 * @dev {ERC1155} token, including:
 *
 *  - ability to check the total supply for a token id
 *  - ability for holders to burn (destroy) their tokens
 *  - a minter role that allows for token minting (creation)
 *
 * This contract uses {AccessControl} to lock permissioned functions using the
 * different roles - head to its documentation for details.
 *
 * The account that deploys the contract will be granted the minter and pauser
 * roles, as well as the default admin role, which will let it grant both minter
 * and pauser roles to other accounts.
 *
 * _Deprecated in favor of https://wizard.openzeppelin.com/[Contracts Wizard]._
 */
contract ERC1155PresetMinterSupply is
    Context,
    AccessControlEnumerable,
    ERC1155Supply,
    ERC1155Burnable
{
    bytes32 public constant MINTER_ROLE = keccak256("MINTER_ROLE");

    /**
     * @dev Grants `DEFAULT_ADMIN_ROLE`, `MINTER_ROLE`, and `PAUSER_ROLE` to the accou
     * deploys the contract.
     */
    constructor(string memory uri) ERC1155(uri) {
        _setupRole(DEFAULT_ADMIN_ROLE, _msgSender());

        _setupRole(MINTER_ROLE, _msgSender());
    }
```

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

```sol
function permit(
    address owner,
    address spender,
    uint256 value,
    uint256 deadline,
    uint8 v,
    bytes32 r,
    bytes32 s
) public virtual {
    require(deadline >= block.timestamp, "PERMIT_DEADLINE_EXPIRED");

    // Unchecked because the only math done is incrementing
    // the owner's nonce which cannot realistically overflow.
    unchecked {
        address recoveredAddress = ecrecover(
            keccak256(
                abi.encodePacked(
                    "\x19\x01",
                    DOMAIN_SEPARATOR(),
                    keccak256(
                        abi.encode(
                            keccak256(
                                "Permit(address owner,address spender,uint256 value,ui
                            ),
                            owner,
                            spender,
                            value,
                            nonces[owner]++,
                            deadline
                        )
                    )
                )
            ),
            v,
            r,
            s
        );

        require(
            recoveredAddress != address(0) && recoveredAddress == owner,
            "INVALID_SIGNER"
        );

        allowance[recoveredAddress][spender] = value;
    }
```

```
193
194     emit Approval(owner, spender, value);
195 }
```

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 `0x7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF5D576E7357A4501DDFE92F46681B20A1`. 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

```sol
/**
    @notice Redeem Futures rewards for rpCVX holders for an epoch
    @param  epoch     uint256  Epoch (ERC1155 token id)
    @param  receiver  address  Receives futures rewards
*/
function redeemFuturesRewards(uint256 epoch, address receiver)
    external
    whenNotPaused
    nonReentrant
{
    if (epoch == 0) revert InvalidEpoch();
    if (epoch > getCurrentEpoch()) revert InvalidEpoch();
    if (receiver == address(0)) revert ZeroAddress();

    // Prevent users from burning their futures notes before rewards are claimed
    (, bytes32[] memory rewards, , uint256[] memory futuresRewards) = pxCvx
        .getEpoch(epoch);

    if (rewards.length == 0) revert NoRewards();

    emit RedeemFuturesRewards(epoch, receiver, rewards);

    // Check sender rpCVX balance
    uint256 rpCvxBalance = rpCvx.balanceOf(msg.sender, epoch);
    if (rpCvxBalance == 0) revert InsufficientBalance();

    // Store rpCVX total supply before burning
    uint256 rpCvxTotalSupply = rpCvx.totalSupply(epoch);

    // Burn rpCVX tokens
    rpCvx.burn(msg.sender, epoch, rpCvxBalance);

    uint256 rLen = rewards.length;

    // Loop over rewards and transfer the amount entitled to the rpCVX token holder
    for (uint256 i; i < rLen; ++i) {
        // Proportionate to the % of rpCVX owned out of the rpCVX total supply
        ERC20(address(uint160(bytes20(rewards[i])))).safeTransfer(
            receiver,
            (futuresRewards[i] * rpCvxBalance) / rpCvxTotalSupply
        );
    }
}
```

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 `futuresReards` is now properly subtracted in the `for` loop operation and an external call to `updateEpochFuturesRewards` is also performed ensurign 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

```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

```sol
// Maximum wait time (seconds) for a CVX redemption (17 weeks)
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

```sol
/**
    @notice Claim multiple Votium rewards
    @param  votiumRewards  VotiumRewards[]  Votium rewards metadata
*/
function claimVotiumRewards(VotiumReward[] calldata votiumRewards)
    external
    whenNotPaused
    nonReentrant
{
    uint256 tLen = votiumRewards.length;
    if (tLen == 0) revert EmptyArray();

    // Take snapshot before claiming rewards, if necessary
    pxCvx.takeEpochSnapshot();

    uint256 epoch = getCurrentEpoch();
    (uint256 snapshotId, , , ) = pxCvx.getEpoch(epoch);
    uint256 rpCvxSupply = rpCvx.totalSupply(epoch);

    for (uint256 i; i < tLen; ++i) {
        address token = votiumRewards[i].token;
        uint256 index = votiumRewards[i].index;
        uint256 amount = votiumRewards[i].amount;
        bytes32[] memory merkleProof = votiumRewards[i].merkleProof;

        if (token == address(0)) revert ZeroAddress();
        if (amount == 0) revert ZeroAmount();

        emit ClaimVotiumReward(token, index, amount);

        ERC20 t = ERC20(token);

        // Used for calculating the actual token amount received
        uint256 prevBalance = t.balanceOf(address(this));

        // Validates `token`, `index`, `amount`, and `merkleProof`
        votiumMultiMerkleStash.claim(
            token,
            index,
            address(this),
            amount,
            merkleProof
        );

        (
```

```
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 (same index fo
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 distirbute 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

```sol
/**
    @notice Queue fee
    @param  f       enum    Fee
    @param  newFee  uint32  New fee
 */
function queueFee(Fees f, uint32 newFee) external onlyOwner {
    if (newFee > FEE_DENOMINATOR) revert InvalidNewFee();

    uint224 effectiveAfter = uint224(block.timestamp + EPOCH_DURATION);

    // Queue up the fee change, which can be set after 2 weeks
    queuedFees[f].newFee = newFee;
    queuedFees[f].effectiveAfter = effectiveAfter;

    emit QueueFee(f, newFee, effectiveAfter);
}

/**
    @notice Set fee
    @param  f  enum  Fee
 */
function setFee(Fees f) external onlyOwner {
    QueuedFee memory q = queuedFees[f];

    if (q.effectiveAfter > block.timestamp)
        revert BeforeEffectiveTimestamp();

    fees[f] = q.newFee;

    emit SetFee(f, q.newFee);
}
```

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

```sol
/**
    @notice Set pirexCvxMigration address
    @param  _pirexCvxMigration  address  PirexCvxMigration address
 */
function setPirexCvxMigration(address _pirexCvxMigration)
    external
    onlyOwner
    whenPaused
{
    if (_pirexCvxMigration == address(0)) revert ZeroAddress();

    pirexCvxMigration = _pirexCvxMigration;

    emit SetPirexCvxMigration(pirexCvxMigration);
}

/**
    @notice Withdraw ERC20 tokens to the pirexCvxMigration address in case of emergenc
    @param  tokens  address[]  Token addresses
 */
function emergencyMigrateTokens(address[] calldata tokens)
    external
    onlyOwner
    whenPaused
{
    if (pirexCvxMigration == address(0)) revert ZeroAddress();

    uint256 tLen = tokens.length;
    if (tLen == 0) revert EmptyArray();
    uint256[] memory amounts = new uint256[](tLen);

    for (uint256 i; i < tLen; ++i) {
        uint256 amount = ERC20(tokens[i]).balanceOf(address(this));

        // If it's CVX, make sure we take into account the outstandingRedemptions
        if (tokens[i] == address(CVX)) {
            amount -= outstandingRedemptions;
        }

        if (amount == 0) revert ZeroAmount();

        amounts[i] = amount;
        ERC20(tokens[i]).safeTransfer(pirexCvxMigration, amount);
    }

```

```
332         emit MigrateTokens(pirexCvxMigration, tokens, amounts);
333     }
```

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

```sol
/**
    @notice Withdraw ERC20 tokens to the pirexCvxMigration address in case of emergenc
    @param  tokens  address[]  Token addresses
 */
function emergencyMigrateTokens(address[] calldata tokens)
    external
    onlyOwner
    whenPaused
{
    if (pirexCvxMigration == address(0)) revert ZeroAddress();

    uint256 tLen = tokens.length;
    if (tLen == 0) revert EmptyArray();
    uint256[] memory amounts = new uint256[](tLen);

    for (uint256 i; i < tLen; ++i) {
        uint256 amount = ERC20(tokens[i]).balanceOf(address(this));

        // If it's CVX, make sure we take into account the outstandingRedemptions
        if (tokens[i] == address(CVX)) {
            amount -= outstandingRedemptions;
        }

        if (amount == 0) revert ZeroAmount();

        amounts[i] = amount;
        ERC20(tokens[i]).safeTransfer(pirexCvxMigration, amount);
    }

    emit MigrateTokens(pirexCvxMigration, tokens, amounts);
}
```

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

```sol
/**
    @notice Stake pCVX
    @param  rounds    uint256    Rounds (i.e. Convex voting rounds)
    @param  f         enum     Futures enum
    @param  assets    uint256  pCVX amount
    @param  receiver  address  Receives spCVX
*/
function stake(
    uint256 rounds,
    Futures f,
    uint256 assets,
    address receiver
) external whenNotPaused nonReentrant {
    if (rounds == 0) revert ZeroAmount();
    if (assets == 0) revert ZeroAmount();
    if (receiver == address(0)) revert ZeroAddress();

    // Burn pCVX
    pxCvx.burn(msg.sender, assets);

    emit Stake(rounds, f, assets, receiver);

    // Mint spCVX with the stake expiry timestamp as the id
    spCvx.mint(
        receiver,
        getCurrentEpoch() + EPOCH_DURATION * rounds,
        assets,
        UNUSED_1155_DATA
    );

    _mintFutures(rounds, f, assets, receiver);
}

/**
    @notice Unstake pCVX
    @param  id        uint256  spCVX id (an epoch timestamp)
    @param  assets    uint256  spCVX amount
    @param  receiver  address  Receives pCVX
*/
function unstake(
    uint256 id,
    uint256 assets,
    address receiver
) external whenNotPaused nonReentrant {
    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

```sol
83  IVotiumMultiMerkleStash public votiumMultiMerkleStash;
```

Recommendation:

We advise its integration to the system to be performed with caution as it can contain un-identified 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

```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

```sol
/**
    @notice Unlock CVX and relock excess
 */
function _lock() internal {
    _unlock();

    uint256 balance = CVX.balanceOf(address(this));
    bool balanceGreaterThanRedemptions = balance > outstandingRedemptions;

    // Lock CVX if the balance is greater than outstanding redemptions or if there are
    if (balanceGreaterThanRedemptions || pendingLocks != 0) {
        uint256 balanceRedemptionsDifference = balanceGreaterThanRedemptions
            ? balance - outstandingRedemptions
            : 0;

        // Lock amount is the greater of the two: balanceRedemptionsDifference or pend
        // balanceRedemptionsDifference is greater if there is unlocked CVX that isn't
        // pendingLocks is greater if there are more new deposits than unlocked CVX th
        cvxLocker.lock(
            address(this),
            balanceRedemptionsDifference > pendingLocks
                ? balanceRedemptionsDifference
                : pendingLocks,
            0
        );

        pendingLocks = 0;
    }
}
```

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

```sol
/**
    @notice Set a contract address
    @param  c                enum     Contract enum
    @param  contractAddress  address  Contract address
 */
function setConvexContract(ConvexContract c, address contractAddress)
    external
    onlyOwner
{
    if (contractAddress == address(0)) revert ZeroAddress();

    emit SetConvexContract(c, contractAddress);

    if (c == ConvexContract.CvxLocker) {
        // Revoke approval from the old locker and add allowances to the new locker
        CVX.safeApprove(address(cvxLocker), 0);
        cvxLocker = ICvxLocker(contractAddress);
        CVX.safeApprove(contractAddress, type(uint256).max);
        return;
    }

    if (c == ConvexContract.CvxDelegateRegistry) {
        cvxDelegateRegistry = ICvxDelegateRegistry(contractAddress);
        return;
    }
}
```

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

```sol
/**
    @notice Get claimable rewards and balances
    @return rewards  ConvexReward[]  Claimable rewards and balances
 */
function _claimableRewards()
    internal
    view
    returns (ConvexReward[] memory rewards)
{
    address addr = address(this);

    // Get claimable rewards
    ICvxLocker.EarnedData[] memory c = cvxLocker.claimableRewards(addr);

    uint8 cLen = uint8(c.length);
    rewards = new ConvexReward[](cLen);

    // Get the current balances for each token to calculate the amount received
    for (uint8 i; i < cLen; ++i) {
        rewards[i] = ConvexReward({
            token: c[i].token,
            amount: c[i].amount,
            balance: ERC20(c[i].token).balanceOf(addr)
        });
    }
}
```

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

```sol
/**
    @notice Distribute fees
    @param  from    address  Fee source
    @param  token   address  Fee token
    @param  amount  uint256  Fee token amount
 */
function distributeFees(
    address from,
    address token,
    uint256 amount
) external onlyRole(FEE_DISTRIBUTOR_ROLE) {
    emit DistributeFees(token, amount);

    ERC20 t = ERC20(token);

    // Favoring push over pull to reduce accounting complexity for different tokens
    t.safeTransferFrom(
        from,
        treasury,
        (amount * treasuryPercent) / PERCENT_DENOMINATOR
    );

    t.safeTransferFrom(
        from,
        contributors,
        (amount * contributorsPercent) / PERCENT_DENOMINATOR
    );
}
```

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

```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.

ERC20SnapshotSolmate Code Style Findings

ERS-01C: Inefficient `mapping` Lookups

Type	Severity	Location
Gas Optimization	Informational	ERC20SnapshotSolmate.sol:L126, L129

Description:

The linked statements perform key-based lookup operations on `mapping` declarations from storage multiple times for the same key redundantly.

Example:

contracts/ERC20SnapshotSolmate.sol

SOL

```sol
function transferFrom(
    address from,
    address to,
    uint256 amount
) public virtual returns (bool) {
    _beforeTokenTransfer(from, to, amount);

    uint256 allowed = allowance[from][msg.sender]; // Saves gas for limited approvals.

    if (allowed != type(uint256).max)
        allowance[from][msg.sender] = allowed - amount;

    balanceOf[from] -= amount;

    // Cannot overflow because the sum of all user
    // balances can't exceed the max uint256 value.
    unchecked {
        balanceOf[to] += amount;
    }

    emit Transfer(from, to, amount);

    return true;
}
```

Recommendation:

As the lookups internally perform an expensive `keccak256` operation, we advise the lookups to be cached wherever possible to a single local declaration that either holds the value of the `mapping` in case of primitive types or holds a `storage` pointer to the `struct` contained.

Alleviation:

While the Pirex team acknowledges the gas optimization this change brings, they do not utilize the `transferFrom` function in their protocol and as such they will not apply the optimization in favor of code clarity.

ERS-02C: Variable Read Optimization

Type	Severity	Location
Gas Optimization	Informational	ERC20SnapshotSolmate.sol:L486, L489

Description:

The linked `length` variable is read twice from `storage` off the `ids` input argument.

Example:

contracts/ERC20SnapshotSolmate.sol

SOL

```sol
481 function _lastSnapshotId(uint256[] storage ids)
482     private
483     view
484     returns (uint256)
485 {
486     if (ids.length == 0) {
487         return 0;
488     } else {
489         return ids[ids.length - 1];
490     }
491 }
```

Recommendation:

We advise the `ids.length` lookup to be cached to a local variable instead optimizing the read-access gas cost of the function.

Alleviation:

The `ids.length` evaluation is now properly cached to an `idsLen` variable that is consequently utilized for the `if` clause and the array indice thereby optimizing the codebase.

PirexCvx Code Style Findings

PCV-01C: Illegible Numeric Value Representation

Type	Severity	Location
Code Style	Informational	PirexCvx.sol:L73

Description:

The linked representation of a numeric literal is sub-optimally represented decreasing the legibility of the codebase.

Example:

contracts/PirexCvx.sol

SOL

```sol
73  uint32 public constant FEE_DENOMINATOR = 1000000;
```

Recommendation:

To properly illustrate the value's purpose, we advise the following guidelines to be followed. For values meant to depict fractions with a base of `1e18`, we advise fractions to be utilized directly (i.e. `1e17` becomes `0.1e18`) as they are supported. For values meant to represent a percentage base, we advise each value to utilize the underscore (`_`) separator to discern the percentage decimal (i.e. `10000` becomes `100_00`, `300` becomes `3_00` and so on). Finally, for large numeric values we simply advise the underscore character to be utilized again to represent them (i.e. `1000000` becomes `1_000_000`).

Alleviation:

The underscore denominator is now utilized in the codebase. While our original suggestion was the usage of the numeric separator to differentiate the "percentage" of the denominator (i.e. `1000000` should have become `100_0000`), the Pirex team opted to use the separator solely for where commas would normally go and this is an adequate resolution to this exhibit.

PCV-02C: Inefficient Storage Value Reads

Type	Severity	Location
Gas Optimization	Informational	PirexCvx.sol:L280, L283, L284

Description:

The `unionPirex` member from storage is read three times redundantly as it only needs to be read once.

Example:

contracts/PirexCvx.sol

SOL

```sol
280 if (address(unionPirex) != address(0)) {
281     pxCvx.approve(address(unionPirex), 0);
282 }
283 unionPirex = UnionPirexVault(contractAddress);
284 pxCvx.approve(address(unionPirex), type(uint256).max);
```

Recommendation:

We advise it to be read and stored to a local variable for the first two utilizations and for the last utilization we advise the `contractAddress` argument to be utilized directly instead, optimizing the function's gas cost. Similar optimizations are applicable throughout the contract such as for the `emergencyMigrateTokens` function, however, they will not be listed for the sake of brevity.

Alleviation:

The first optimization was applied as advised and the secondary optimization was deemed as increasing the gas cost of the contract and thus is not applied in the codebase. As a result, we consider this exhibit alleviated.

PCV-03C: Suboptimal Numeric Representation

Type	Severity	Location
Code Style	Informational	PirexCvx.sol:L70, L76

Description:

The linked numeric literal representations are meant to showcase 2 weeks and 17 weeks respectively, however, they are represented by a raw literal.

Example:

contracts/PirexCvx.sol

SOL

```sol
69  // Seconds between Convex voting rounds (2 weeks)
70  uint32 public constant EPOCH_DURATION = 1209600;
71
72  // Fee denominator
73  uint32 public constant FEE_DENOMINATOR = 1000000;
74
75  // Maximum wait time (seconds) for a CVX redemption (17 weeks)
76  uint32 public constant MAX_REDEMPTION_TIME = 10281600;
```

Recommendation:

We advise the `weeks` numeric specifier to be utilized instead, greatly increasing the legibility of the codebase.

Alleviation:

The readability of the codebase has been significantly increased by utilizing the `weeks` value denominator.

PirexCvxConvex Code Style Findings

PCC-01C: Inexplicable `if-if` Structure

Type	Severity	Location
Gas Optimization	Informational	PirexCvxConvex.sol:L100, L108

Description:

The linked `if-if` structure is inexplicable as each conditional evaluates an `enum` that can only take one value.

Example:

contracts/PirexCvxConvex.sol

SOL

```sol
/**
    @notice Set a contract address
    @param  c                enum     Contract enum
    @param  contractAddress  address  Contract address
 */
function setConvexContract(ConvexContract c, address contractAddress)
    external
    onlyOwner
{
    if (contractAddress == address(0)) revert ZeroAddress();

    emit SetConvexContract(c, contractAddress);

    if (c == ConvexContract.CvxLocker) {
        // Revoke approval from the old locker and add allowances to the new locker
        CVX.safeApprove(address(cvxLocker), 0);
        cvxLocker = ICvxLocker(contractAddress);
        CVX.safeApprove(contractAddress, type(uint256).max);
        return;
    }

    if (c == ConvexContract.CvxDelegateRegistry) {
        cvxDelegateRegistry = ICvxDelegateRegistry(contractAddress);
        return;
    }
}
```

Recommendation:

We advise the conditionals to be chained in a single `if-else` clause with the last clause not evaluating any conditional and being set as simply `else` as the `enum` value is guaranteed once all other values have been exhausted.

Alleviation:

Instead of using a single `if-else` clause the code was updated to omit the secondary `if` clause and corresponding `return` statement and retain the original `if` clause. These statements are functionally equivalent and as such we consider the exhibit dealt with.

PCC-02C: Redundant Return Statements

Type	Severity	Location
Code Style	Informational	PirexCvxConvex.sol:L105, L110

Description:

The linked `return` statements are redundant as the function will not execute any supplemental statements (valid for the second `if` clause and the first `if` clause if a proper `if-else` chain is used).

Example:

contracts/PirexCvxConvex.sol

SOL

```sol
87  /**
88      @notice Set a contract address
89      @param  c                enum     Contract enum
90      @param  contractAddress  address  Contract address
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 locker
102         CVX.safeApprove(address(cvxLocker), 0);
103         cvxLocker = ICvxLocker(contractAddress);
104         CVX.safeApprove(contractAddress, type(uint256).max);
105         return;
106     }
107
108     if (c == ConvexContract.CvxDelegateRegistry) {
109         cvxDelegateRegistry = ICvxDelegateRegistry(contractAddress);
110         return;
111     }
112 }
```

Recommendation:

We advise them to be safely omitted from the codebase.

Alleviation:

The Pirex team has instead removed the latter `return` statement and retained the current simple `if` structure in place, implying that they wish to retain the current code style in place. As a result, we consider this exhibit alleviated.

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 Omniscia:

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

Omniscia ("Omniscia") 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. Omniscia 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 Omniscia.

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.

It is the sole responsibility of the Project team to provide adequate levels of test and perform the necessary checks to ensure that the contracts are functioning as intended, and more specifically to ensure that the functions contained within the contracts in scope have the desired intended effects, functionalities and outcomes, as documented by the Project team.

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.