Cyberscope

A TAC Security Company

Audit Report

Plutus Migration

August 2025

Repository : https://github.com/PlutusDao/plsMigration

Commit : 30938d725c956e11fa2894f38de57ed82a526638

Audited by © cyberscope

Table of Contents

Risk Classification

The criticality of findings in Cyberscope's smart contract audits is determined by evaluating multiple variables. The two primary variables are:

1. **Likelihood of Exploitation**: This considers how easily an attack can be executed, including the economic feasibility for an attacker.
2. **Impact of Exploitation**: This assesses the potential consequences of an attack, particularly in terms of the loss of funds or disruption to the contract's functionality.

Based on these variables, findings are categorized into the following severity levels:

1. **Critical**: Indicates a vulnerability that is both highly likely to be exploited and can result in significant fund loss or severe disruption. Immediate action is required to address these issues.
2. **Medium**: Refers to vulnerabilities that are either less likely to be exploited or would have a moderate impact if exploited. These issues should be addressed in due course to ensure overall contract security.
3. **Minor**: Involves vulnerabilities that are unlikely to be exploited and would have a minor impact. These findings should still be considered for resolution to maintain best practices in security.
4. **Informative**: Points out potential improvements or informational notes that do not pose an immediate risk. Addressing these can enhance the overall quality and robustness of the contract.

Severity	Likelihood / Impact of Exploitation
● Critical	Highly Likely / High Impact
● Medium	Less Likely / High Impact or Highly Likely/ Lower Impact
● Minor / Informative	Unlikely / Low to no Impact

Review

Audit Updates

Initial Audit	27 Aug 2025

Source Files

Filename	SHA256
XPlutusToken.sol	494f60e02f9fe2c4a27ac1e2898fdf9b2ea7f6004395e540a32b24e44e185acb
Vester.sol	a1cc0b90397dfd06a8042233d08f4dfd3d943917472b9492d95c444bb49496a1
PlutusRouterV2.sol	3eb4a45f46c7f8b985aa92d8b14d641079811516349d96073d94c04c7b0989bc
PlsMigration.sol	57e8b4d4acd1cf98e5bfe9acda6fc9c83142b702fc64fa2df7fb778729321b6f
LockedToken.sol	90c3724eda81a4eadcc72033391b2c18e100ab7db783676cba63ac61a7fc2f5f
interfaces/Interfaces.sol	5c96665c9972f38432566414b3756bef6aeab5dcd360e44f0827cebe99e3cbe6
interfaces/IXPlutusToken.sol	cc6039bdae7f9bcceb789e3bee664fd5bf46027e9dcd9e335fd719a758acaac5
interfaces/IVester.sol	6d47c16eb20af99442db0c23eb24a6471d1e7f84c3b30fe467fade2eadb1bc3d
interfaces/IPlutusToken.sol	5037b6f0839b190c1e452e28e05563d4328f26c1e3af3afd3a8cc467d6ab5928
interfaces/IPlutusRouterV2.sol	cfc03ff586ca0afc9f9c38925636720ccc1ce8be60d5a1ecd640518d1a4d548e

interfaces/IPlsMigration.sol	452395339e50c63bd72e8a27d1a8acb959b703107a7db000b62873fe5727c40b
interfaces/ILockedToken.sol	7bc68b9b9307de8278dde41ffbe33b998e20cadba9a18e904f9b370f0b706769
interfaces/ICheckPointer.sol	8dfda7dbdc837449a970a1108984be0212003a33c9b24e00ec75a4f81666a375
interfaces/ICamelotPair.sol	74bff61757a5f3a17e8575ab543bf5c8c15881682b64d1521baa0db19f55bcd6
interfaces/IBonusTracker.sol	f4cb7e802df98611ea9752ac8060677fb09ab7d14c935e96a4064cf2b91d6c2e

Disclaimer

The audit scope is to check for security vulnerabilities, validate the business logic and propose potential optimizations. The contract heavily depends on the interaction with external tracker contracts and checkpointer contracts. These contracts were not in the scope of this review hence any issues arising from these external dependencies are considered outside the audit's scope. It is highly recommended that the team interacts only with safe and audited contracts.

Overview

This migration suite centers on upgrading the PLUTUS ecosystem to a new token model with optional vesting and staking migration flows. It comprises five core contracts: `PlsMigration` , `PlutusRouterV2` , `XPlutusToken` , `Vester` , and `LockedToken` . Together, they enable users to migrate legacy PLUTUS-related assets, unwind staking positions, convert to `xPLUTUS` , redeem back to PLUTUS via configurable vesting schedules, and manage epoch-based lockups. The architecture employs upgradeable patterns (UUPS) and strict role/handler gating to secure privileged operations during migration.

PlsMigration

`PlsMigration` orchestrates the migration from legacy PLUTUS variants to the new `PLUTUS` and `xPLUTUS` with multiplier-based rates.

- **Purpose**: Enable users to convert `OLD_PLUTUS` 1:1 to `PLUTUS` or 1:1.1 to `xPLUTUS` ; convert other bPLS/vPLS variants (including positions unwound via the router) to `xPLUTUS` at bonus ratios.
- **Key flows**:
  - `migratePls(toXPlutus)` : Burns `OLD_PLUTUS` and mints/transfers `PLUTUS` or converts to `xPLUTUS` at configured ratios.
  - `migrateToXPlutus()` : Closes all positions via `PlutusRouterV2` , burns eligible tokens, and converts the resulting amounts into `xPLUTUS` .
  - Preview helpers (`getPlsMigrationPreview` , `getOtherTokensMigrationPreview` , `getMigrationPreview`).
  - Admin setup (`setupMigration`) to set operator roles, shut down legacy lockers, configure router/vester, and set multipliers/deadlines.
- **Controls**: Access-controlled admin, time-bounded by `migrationPeriod` , and safe minting with fallback to pre-funded balances via `_ensurePlutusTokens` .

PlutusRouterV2

`PlutusRouterV2` manages staking, bonus tracking, and lock/unstake flows across PLS, PLS-WETH LP, esPLS, and mpPLS trackers, with migration-guarded entry points.

- **Purpose**: Coordinate user staking/unstaking across multiple reward trackers, handle epoch-based lockers, and centralize “close all positions” for migration.
- **Key flows**:
  - Stake/lock and unlock/unstake for PLS and PLS-WETH (`stakeAndLockPls` , `unlockAndUnstakePls` , `stakeAndLockPlsWeth` , etc.).
  - Stake/unstake esPLS and claim/stake mpPLS bonuses.
  - Automatic lock extension processing before actions; delegation to voting checkpointers.
  - `closeAllPositions(user)` : Claims, exits lockers, unstakes all tracked assets, burns mpPLS, and returns totals for migration math.
- **Extensibility**: Callback system before/after actions for external integrations; owner-managed migrator, kicker, pause, and callback registry.
- **Safety**: Pausable, reentrancy guarded, and special `onlyMigrator` behavior when shutdown is active.

XPlutusToken

`XPlutusToken` is a non-transferable (except whitelisted) ERC20 representing staked/converted value with a linear, configurable vest-to-PLUTUS mechanism.

- **Purpose**: Accept PLUTUS via `convert` to mint `xPLUTUS` , then allow users to vest `xPLUTUS` over a chosen duration to redeem PLUTUS at a duration-based fixed ratio, with any excess routed to an `excessReceiver` .
- **Key flows**:
  - `convert(amount, to)` : Pulls PLUTUS and mints `xPLUTUS` .
  - `vest(amount, duration)` : Locks `xPLUTUS` into a vest entry, computing redeemable PLUTUS by a linear ratio between `minRatio` and `maxRatio` over `[minDuration, maxDuration]` .
  - `redeem(vestId)` : After maturity, burns `xPLUTUS` and sends PLUTUS to user; sends excess to `excessReceiver` .
  - `cancelVest(vestId)` : Return of `xPLUTUS` pre-maturity and vest invalidation.
- **Config**: Admin can set redeem settings, whitelist for transfer exceptions, and the `excessReceiver` . Includes enumerable tracking of vest entries per user and globally.

Vester

`Vester` is a non-transferable vesting wrapper (vPLS) over `esPLS` that programmatically converts claimable value into `xPLUTUS` over time.

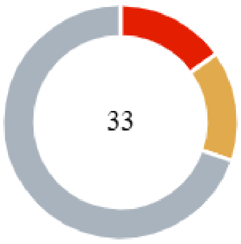
- **Purpose**: Manage long-term vesting of `esPLS` into `xPLUTUS` with optional pair tokens and max-vest constraints based on reward tracker history.
- **Key flows**:
    - Vesting accrual via `_updateVesting`; users call `claim()` to realize claimable amounts.
    - On claim, internally burns vested `esPLS`, pulls/approves claimable token, and converts to `xPLUTUS` for the receiver.
    - Supports reward-tracker-derived `maxVestableAmount`, average staked amounts, and cumulative reward adjustments.
- **Controls**: Handler-gated functions, non-transferable token semantics, owner-configurable target PLUTUS/xPLUTUS token addresses.

LockedToken

`LockedToken` is a generic epoch-based locker used by the router for PLS and PLS-WETH positions.

- **Purpose**: Time-lock tokens in rolling weekly epochs for a fixed 16-epoch duration with optional auto-extend, enabling controlled liquidity and staking strategies.
- **Key flows**:
    - `lock(funding, account, amount)`: Handler-initiated lock; tracks per-epoch unlock schedules.
    - `processExpiredLocksOnBehalf(account)`: Auto-extend matured locks or move them into current schedule if enabled.
    - `withdrawExpiredLocksOnBehalf(account, to)`: Exit matured locks and transfer underlying.
- **Admin**: Owner can set `isHandler`, toggle `shutdown` to unlock all, and uses UUPS upgradability.

Findings Breakdown



●	Critical	5
●	Medium	5
●	Minor / Informative	23

Severity		Unresolved	Acknowledged	Resolved	Other
●	Critical	0	5	0	0
●	Medium	0	5	0	0
●	Minor / Informative	18	5	0	0

Diagnostics

🔴 Critical 🟠 Medium ⚪ Minor / Informative

Severity	Code	Description	Status
🔴	ECR	Early Claim Reset	Acknowledged
🔴	IBO	Inconsistent Burn Operation	Acknowledged
🔴	IDO	Ineffective Deposit Operations	Acknowledged
🔴	IST	Ineffective Self Transfer	Acknowledged
🔴	MAC	Missing Access Control	Acknowledged
🟠	MAR	Missing Allowance Restrictions	Acknowledged
🟠	MWF	Missing Withdrawal Functionality	Acknowledged
🟠	PDD	Potential Duplicate Deposit	Acknowledged
🟠	PLT	Potential Locked Token	Acknowledged
🟠	RMD	Redundant Mapping Declaration	Acknowledged
⚪	AAO	Accumulated Amount Overflow	Unresolved
⚪	AME	Address Manipulation Exploit	Unresolved
⚪	BC	Blacklists Addresses	Acknowledged
⚪	CO	Code Optimization	Acknowledged

●	CCR	Contract Centralization Risk	Unresolved
●	ISUP	Inconsistent Stake Unstake Pattern	Unresolved
●	MT	Mints Tokens	Acknowledged
●	MMN	Misleading Method Naming	Unresolved
●	MC	Missing Check	Acknowledged
●	PBV	Pending Balance Visibility	Unresolved
●	PMD	Potential Mint Discrepancy	Unresolved
●	PTAI	Potential Transfer Amount Inconsistency	Unresolved
●	PISR	Potentially Inconsistent Staking Records	Unresolved
●	PIO	Potentially Ineffective Overrides	Unresolved
●	ST	Stops Transactions	Acknowledged
●	UTPD	Unverified Third Party Dependencies	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L11	Unnecessary Boolean equality	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L15	Local Scope Variable Shadowing	Unresolved
●	L16	Validate Variable Setters	Unresolved

●	L20	Succeeded Transfer Check	Unresolved

ECR - Early Claim Reset

Criticality	Critical
Location	Vester.sol#L103
Status	Acknowledged

Description

In the `_deposit` function, the contract first calls `_claim` and then resets both `claimedAmounts` and `cumulativeClaimAmounts` to zero, effectively erasing the user's vesting history. This causes the system to treat future deposits as if no prior claims were made, allowing users to potentially claim rewards multiple times for amounts that have been already accounted for. Such behavior undermines the intended linear vesting mechanism and can result in excessive or duplicated reward allocations.

```
claimedAmounts[_account] = 0;
cumulativeClaimAmounts[_account] = 0;
```

Recommendation

To preserve the integrity of the vesting mechanism the team is advised to prevent inconsistent state modifications. This will ensure that the user's claim history is accurately maintained, preventing duplicate reward distributions and maintaining consistent vesting behavior even under multiple operations.

Team Update

The team has acknowledged that this is not a security issue and states:

The contract is being deprecated, so no deposits will be allowed after the upgrade.

IBO - Inconsistent Burn Operation

Criticality	Critical
Location	PlutusRouterV2.sol#L324
Status	Acknowledged

Description

The `_unstake` method, when processing `plsWeth` tokens, first calls the `unstakeForAccount` function followed by the `burn` method on the pair token. However, the contract does not confirm that the necessary tokens have been transferred to the pair contract before initiating the unstake process. This oversight can lead to a failure in the unstake operation if the pair contract lacks the required token balance.

```
IRewardTracker(_rewardTracker).unstakeForAccount(_account, _token, _amount,
plsWeth);
(amount0,) = ICamelotPair(plsWeth).burn(_account);
```

Recommendation

The team should ensure that the pair tokens are transferred to the pair contract before invoking the `burn` method. This guarantees that unstakes are processed as intended and prevents potential inconsistencies in the contract's behavior.

IDO - Ineffective Deposit Operations

Criticality	Critical
Location	Vester.sol#L80,85
Status	Acknowledged

Description

The contract is designed to support token deposits as a core feature. However, both the `deposit` and `depositForAccount` functions consistently revert, regardless of the caller. This results in a permanent failure that cannot be resolved through subsequent actions, effectively rendering the deposit functionality and the overall contract's implementation ineffective.

```
function deposit(uint256 _amount) external nonReentrant {
revert("Vester: deposit not allowed");
_deposit(msg.sender, _amount);
}

function depositForAccount(address _account, uint256 _amount) external
nonReentrant {
revert("Vester: deposit not allowed");
_validateHandler();
_deposit(_account, _amount);
}
```

Recommendation

Eliminating early revert statements allows the deposit functions to execute as intended. This change ensures that the contract consistently carries out its core operations, preserving alignment with its designed functionality.

Team Update

The team has acknowledged that this is not a security issue and states:

This is the intended behavior after the upgrade, basically deprecating the core features of the contract.

IST - Ineffective Self Transfer

Criticality	Critical
Location	Vester.sol#L247
Status	Acknowledged

Description

During the `_claim` process, the contract attempts to transfer `claimableToken` to itself using the `safeTransfer` function:

```
IERC20Upgradeable(claimableToken).safeTransfer(address(this),
amount);
```

This operation is effectively a null operation as it has no effect expecting the contract to already hold the specified `amount` of `claimableToken` . If the tokens are expected to come from an external source, such as a user or another contract, this transfer will fail, causing the entire `_claim` operation to revert.

Recommendation

The team is advised to review the logic of the `_claim` function to ensure that token transfers are directed from the correct source. Transferring tokens from the contract to itself has no effect and may indicate a misuse of the intended flow. Addressing the transfer direction will prevent unnecessary reverts, ensuring the `_claim` process functions as expected.

MAC - Missing Access Control

Criticality	Critical
Location	PlutusRouterV2.sol#L58
Status	Acknowledged

Description

The contract lacks proper access control on critical functions. Although it implements the `onlyMigrator` modifier, this restriction becomes ineffective when the `isShutdown` flag is set to `true`. In such cases, key functions, such as the `unlockAndUnstakePlsFor`, can be called by any external entity without restriction. This allows unauthorized access to locked tokens owned by other users, leading to potential loss of funds.

```
modifier onlyMigrator() {
if (isShutdown) {
require(msg.sender == migrator, "Unauthorized");
}
_;
}
```

Recommendation

The team is advised to enforce strict access control at all times, including during maintenance and emergency operations. This can be achieved by consistently verifying the identity of the caller to ensure that only authorized entities can perform sensitive actions. Maintaining access control in all scenarios helps prevent unauthorized access and protects the system from potential misuse.

MAR - Missing Allowance Restrictions

Criticality	Medium
Location	LockedToken.sol#L171
Status	Acknowledged

Description

The `lock` function allows any external caller to transfer tokens from a user’s account to the contract, provided that the user has previously approved the contract to spend their tokens. Since the function does not restrict the caller, a malicious actor can exploit this by locking tokens on behalf of any user who has granted approval, without their consent or knowledge. This behavior can lead to unexpected token transfers and potential misuse of user funds.

```
function lock(address fundingAccount, address account, uint256 amount)
external {
if (account == address(0)) revert ZeroAddress();
if (amount == 0) revert ZeroAmount();
_validateHandler();

tokenToLock.safeTransferFrom(fundingAccount, address(this), amount);

_lock(account, amount, Relock.AddToPending);
}
```

Recommendation

The contract should protect users from unauthorized access to their assets. If a user has approved tokens to the contract, the contract must not access those tokens outside the scope of the intended logic.

The team is advised to implement an allowance mechanism where the caller consumes from the user's approved allowance towards the caller to execute the lock. This approach ensures that token access is limited to the user's explicit intent, preventing improper use of their assets.

Team Update

The team has acknowledged that this is not a security issue and states:

This is out of scope for our upgrade as we do not expect this to be used after we push the upgrades.

MWF - Missing Withdrawal Functionality

Criticality	Medium
Location	Vester.sol
Status	Acknowledged

Description

The contract enables users to deposit esTokens in exchange for vPLS tokens. However, it does not provide a withdrawal mechanism that would allow users to redeem their vPLS tokens and recover their original esToken deposits. As a result, users are restricted into completing the full vesting cycle, which may contradict with the expected behaviour of such a deposit.

```
function _deposit(address _account, uint256 _amount) private {
...
}
```

In addition, during the deposit phase the contract may transfer an amount of `pairToken` from the user to its balance. These tokens remain inaccesible and cannot be withdrawn from the contract.

```
IERC20Upgradeable(pairToken).safeTransferFrom(_account, address(this),
pairAmountDiff);
```

Recommendation

The team is advised to consider introducing a withdrawal mechanism that allows users to exchange their vPLS tokens in exchange for the original esTokens. This will provide users with greater flexibility and control.

PDD - Potential Duplicate Deposit

Criticality	Medium
Location	PlutusRouterV2.sol#L122
Status	Acknowledged

Description

The contract includes a `stakeAndLockPls` function that allows the migrator to invoke both the `_stake` and `lock` functionalities. Each of these operations may independently transfer tokens from the caller, which in this case is the migrator. This could lead to the unintentional duplicated spend of tokens. This discrepancy may contradict the intended business logic leading to inconsistencies in accounting for the staked and locked tokens.

```
_stake(msg.sender, msg.sender, pls, _amount, stakedPlsTracker,
bonusPlsTracker, plsCheckpointer);
lockedPls.lock(msg.sender, msg.sender, _amount);
```

Recommendation

The team is advised to review the current implementation to ensure that tokens are double spent and that the logic aligns with the intended design. This verification helps maintain the integrity of the system and prevents inconsistencies in token handling.

Team Update

The team has acknowledged that this is not a security issue and states:

No fix needed as this will be deprecated after the upgrade, not stake or lock operations will be allowed.

PLT - Potential Locked Token

Criticality	Medium
Location	PlutusRouterV2.sol#L324
Status	Acknowledged

Description

The contract burns liquidity pair tokens to retrieve the underlying assets. However, it only processes one of the returned token amounts, leaving the other unhandled and potentially locked within the contract. Additionally, the contract does not verify whether the returned token corresponds to the intended asset, which may result in further tokens being locked or cause transaction reverts.

```
(amount0,) = ICamelotPair(plsWeth).burn(_account);
```

Recommendation

The team is advised to handle all returned token amounts from the liquidity pair burn operation to ensure that no assets remain unintentionally locked. Additionally, it should verify that the returned tokens correspond to the expected assets. This validation prevents incorrect token handling, reduces the risk of transaction failures, and ensures proper asset control.

RMD - Redundant Mapping Declaration

Criticality	Medium
Location	Vester.sol#L27,28,29,31,32
Status	Acknowledged

Description

The contract defines numerous mappings and state variables that are never initialized. As a result, any operations involving these elements do not alter the contract's state. Such redundant components increase code complexity, inflate the contract size, and reduce overall readability.

```
mapping(address => uint256) public pairAmounts;
mapping(address => uint256) public cumulativeRewardDeductions;
mapping(address => uint256) public bonusRewards;

mapping(address => uint256) public transferredAverageStakedAmounts;
mapping(address => uint256) public transferredCumulativeRewards;
```

Recommendation

It is recommended to remove any unused mappings and state variables. Eliminating redundant components reduces code complexity, decreases contract size, and improves overall readability and maintainability.

AAO - Accumulated Amount Overflow

Criticality	Minor / Informative
Location	LockedToken.sol#L190
Status	Unresolved

Description

The contract is using variables to accumulate values. The contract could lead to an overflow when the total value of a variable exceeds the maximum value that can be stored in that variable's data type. This can happen when an accumulated value is updated repeatedly over time, and the value grows beyond the maximum value that can be represented by the data type.

```
balance.locked += lockAmount;
lockedSupply += lockAmount;
```

Recommendation

The team is advised to carefully investigate the usage of the variables that accumulate value. A suggestion is to add checks to the code to ensure that the value of a variable does not exceed the maximum value that can be stored in its data type.

AME - Address Manipulation Exploit

Criticality	Minor / Informative
Location	PlutusRouterV2.sol#L118
Status	Unresolved

Description

The contract's design includes functions that accept external contract addresses as parameters without performing adequate validation or authenticity checks. This lack of verification introduces a significant security risk, as input addresses could be controlled by attackers and point to malicious contracts. Such vulnerabilities could enable attackers to exploit these functions, potentially leading to unauthorized actions or the execution of malicious code under the guise of legitimate operations.

```
function toggleAutoExtend(ILockedToken _token) external
nonReentrant whenNotPaused {
ILockedToken(_token).toggleAutoExtendOnBehalf(msg.sender);
}
```

Recommendation

To mitigate this risk and enhance the contract's security posture, it is imperative to incorporate comprehensive validation mechanisms for any external contract addresses passed as parameters to functions. This could include checks against a whitelist of approved addresses, verification that the address implements a specific contract interface or other methods that confirm the legitimacy and integrity of the external contract. Implementing such validations helps prevent malicious exploits and ensures that only trusted contracts can interact with sensitive functions.

BC - Blacklists Addresses

Criticality	Minor / Informative
Location	XPlutusToken.sol#L103
Status	Acknowledged

Description

The contract owner has the authority to stop addresses from transactions. The owner may take advantage of it by calling the `updateWhitelist` function.

```
function updateWhitelist(address _account, bool _whitelisted) external
onlyRole(DEFAULT_ADMIN_ROLE) {
if (_account == address(this)) {
revert XPlutusToken_InvalidWhitelistAddress();
}
if (_account == address(0)) revert XPlutusToken_InvalidAddress();
whitelist[_account] = _whitelisted;
emit WhitelistUpdated(_account, _whitelisted);
}
```

Recommendation

The team should carefully manage the private keys of the owner’s account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

CO - Code Optimization

Criticality	Minor / Informative
Location	XPlutusToken.sol#L264,271,276,292
Status	Acknowledged

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations. In particular, the contract performs array operations in an inefficient approach that hinders code readability and future maintenance.

```
function _addVestToOwnerEnumeration(address to, uint256 vestId) private {
...
}

function _addVestToAllVestsEnumeration(uint256 vestId) private {
...
}

function _removeVestFromOwnerEnumeration(address from, uint256 vestId) private
{
...
}

function _removeVestFromAllVestsEnumeration(uint256 vestId) private {
...
}
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	LockedToken.sol#L46,171,256,264,286 PlutusRouterV2.sol#L65,111,122,143,147,151,162,183,187,191,200,210,223,232,387,389,394,398,402,410,418,426 Vester.sol#L62,64,68,72,76 PlsMigration.sol#L73,79,220,226,248 XPlutusToken.sol#L83,90,103,113,117,313
Status	Unresolved

Description

The contract's functionality and behavior are heavily dependent on external parameters or configurations. While external configuration can offer flexibility, it also poses several centralization risks that warrant attention. Centralization risks arising from the dependence on external configuration include Single Point of Control, Vulnerability to Attacks, Operational Delays, Trust Dependencies, and Decentralization Erosion.

```
function _validateHandler() internal view {
if (!isHandler[msg.sender]) {
revert UNAUTHORIZED(string.concat(symbol, ": ", "!handler"));
}
}
```

```
function shutdown() external override onlyOwner {...}
function setHandler(address _handler, bool _isActive) external onlyOwner {...}
function _authorizeUpgrade(address newImplementation) internal virtual
override onlyOwner {...}
```

Recommendation

To address this finding and mitigate centralization risks, it is recommended to evaluate the feasibility of migrating critical configurations and functionality into the contract's codebase itself. This approach would reduce external dependencies and enhance the contract's self-sufficiency. It is essential to carefully weigh the trade-offs between external configuration flexibility and the risks associated with centralization.

ISUP - Inconsistent Stake Unstake Pattern

Criticality	Minor / Informative
Location	PlutusRouterV2.sol#L210
Status	Unresolved

Description

The contract includes several functions that allow the migrator to stake tokens, with all stakes being attributed to the `msg.sender`, which is always the migrator. It also provides unstake functions, including `unstakeEsPlsFor`, which enables the migrator to unstake tokens on behalf of a user. However, since all staking actions are performed by the migrator and attributed to their address, there are no user-specific stakes. As a result, the `unstakeEsPlsFor` is inconsistent with the overall stake mechanism of the contract.

```
function unstakeEsPlsFor(address user, uint256 _amount) external override
nonReentrant whenNotPaused onlyMigrator {
_callbackAction(user, PlutusRouterV2.unstakeEsPls.selector, true);

lockedPls.processExpiredLocksOnBehalf(user);
lockedPlsWeth.processExpiredLocksOnBehalf(user);

_unstake(user, esPls, _amount, true, stakedEsPlsTracker, bonusEsPlsTracker,
esPlsCheckpointer);

_callbackAction(user, PlutusRouterV2.unstakeEsPls.selector, false);
}
```

Recommendation

As advised with the finding `PISR`, the team should ensure consistent tracking of staking operations in terms of the owner of the underlying assets. Inconsistencies in these records may result in inaccessible assets or potential loss of funds.

MT - Mints Tokens

Criticality	Minor / Informative
Location	XPlutusToken.sol#L228
Status	Acknowledged

Description

The contract mints tokens. Tokens can be minted by calling the `_convert` function. As a result, the contract tokens will be inflated.

```
function _convert(uint256 _amount, address _to) internal {
if (_amount == 0) revert XPlutusToken_AmountZero();
if (_to == address(0)) revert XPlutusToken_InvalidAddress();

plutus.safeTransferFrom(msg.sender, address(this), _amount);

_mint(_to, _amount);

emit Converted(msg.sender, _to, _amount);
}
```

Recommendation

The team should be aware that allowing tokens to be minted through the `_convert` function without any form of supply limitation introduces a significant risk of token inflation. Since this function mints tokens equivalent to the input amount, repeated use could drastically increase the total supply, potentially undermining the token’s value.

MMN - Misleading Method Naming

Criticality	Minor / Informative
Location	LockedToken.sol#L77,100
Status	Unresolved

Description

Methods can have misleading names if their names do not accurately reflect the functionality they contain or the purpose they serve. The contract uses some method names that are too generic or do not clearly convey the underneath functionality. Misleading method names can lead to confusion, making the code more difficult to read and understand.

In particular, the method named `balanceOf` is misleading, as it returns only the locked balances—excluding both the balances available to unlock and the most recent pending lock. This behavior is not clearly conveyed by the method name, which typically implies a total or available balance.

Additionally, the method `lockedBalanceOfExclPending` returns all locked balances except for the pending locks. The similarity in naming between these two methods, combined with their nuanced differences in behavior, can further confuse users and developers trying to understand or interact with the contract.

```
function balanceOf(address account) public view returns (uint256 amount) {
...
}

function lockedBalanceOfExclPending(address account) public view returns
(uint256 amount) {
...
}
```

Recommendation

It's always a good practice for the contract to contain method names that are specific and descriptive. The team is advised to keep in mind the readability of the code.

MC - Missing Check

Criticality	Minor / Informative
Location	XPlutusToken.sol#L90,121
Status	Acknowledged

Description

The contract is processing variables that have not been properly sanitized and checked that they form the proper shape. These variables may produce vulnerability issues. Specifically, `updateRedeemSettings` does not ensure that the difference between `maxRatio` and `minRatio` is large enough to avoid small fractional values in `getPlutusByVestingDuration` 's ratio calculation, which can lead to truncation and return only the `minRatio` .

```
function updateRedeemSettings(RedeemSettings memory redeemSettings_) external
onlyRole(DEFAULT_ADMIN_ROLE) {
if (redeemSettings_.minRatio > redeemSettings_.maxRatio ||
redeemSettings_.maxRatio > MAX_FIXED_RATIO) {
revert XPlutusToken_WrongRatioValues();
}
if (redeemSettings_.minDuration > redeemSettings_.maxDuration) {
revert XPlutusToken_WrongDurationValues();
}
_redeemSettings = redeemSettings_;
emit RedeemSettingsUpdated(redeemSettings_);
}

...
function getPlutusByVestingDuration(uint256 _xPlutusAmount, uint256 _duration)
public view returns (uint256) {
//...
uint256 ratio = redeemSettings_.minRatio + (((_duration -
redeemSettings_.minDuration) * (redeemSettings_.maxRatio -
redeemSettings_.minRatio)) / (redeemSettings_.maxDuration -
redeemSettings_.minDuration));
//...
}
```

Recommendation

The team is advised to properly check the variables according to the required specifications.

PBV - Pending Balance Visibility

Criticality	Minor / Informative
Location	LockedToken.sol#L99
Status	Unresolved

Description

A newly created lock remains invisible in the `balanceOf` and `activeBalanceOf` views until the next epoch begins. This delay in visibility may lead to user confusion, as it appears that the locked tokens have disappeared.

This behavior is by design, as seen in the `balanceOf` function:

```
function balanceOf(address account) public view returns (uint256 amount) {
...
if (locksLength > 0 && uint256(locks[locksLength - 1].unlockTime) -
LOCK_DURATION > getCurrentEpoch()) {
amount -= locks[locksLength - 1].amount;
}
return amount;
}
```

Recommendation

The team is advised to revisit the implementation of the `balanceOf` method to ensure that newly locked tokens are reflected immediately, regardless of the current epoch. This will provide users with accurate and up-to-date information about their token balances, improving the overall user experience.

PMD - Potential Mint Discrepancy

Criticality	Minor / Informative
Location	PlsMigration.sol#L128 PlutusRouterV2.sol#L426
Status	Unresolved

Description

The `_handleOtherTokensMigration` function invokes `closeAllPositions` from the router, which returns the `totalPls` and `mpPlsStaked_` values. It then burns the caller's balances of both `ES_PLS` and `OLD_PLUTUS` tokens. These burned amounts, along with the returned `totalPls` and `mpPlsStaked_`, are used to calculate and mint new tokens.

However, the burning process does involve the later variables returned by the router. This mismatch can lead to a discrepancy between the actual token supply of the existing tokens and the intended supply of tokens to be minted, potentially affecting the system's consistency.

```
function _handleOtherTokensMigration() internal returns (uint256
xPlutusAmount) {
// Close all bPLS positions and get staked amounts
(uint256 totalPls, uint256 mpPlsStaked_) =
PLUTUS_ROUTER.closeAllPositions(msg.sender);
...
if (esBalance > 0) {
IPlutusToken(ES_PLS).burn(msg.sender, esBalance);
}
if (oldPlsBalance > 0) {
IPlutusToken(OLD_PLUTUS).burn(msg.sender, oldPlsBalance);
}
...
}
```

Recommendation

The team should ensure that all existing tokens used for minting new tokens are handled in accordance with the intended design. This will help prevent discrepancies in token supply during the migration process.

PTAI - Potential Transfer Amount Inconsistency

Criticality	Minor / Informative
Location	LockedToken.sol#L171 XPlutusToken.sol#L222
Status	Unresolved

Description

The `transfer()` and `transferFrom()` functions are used to transfer a specified amount of tokens to an address. The fee or tax is an amount that is charged to the sender of an ERC20 token when tokens are transferred to another address. According to the specification, the transferred amount could potentially be less than the expected amount. This may produce inconsistency between the expected and the actual behavior.

```
function lock(address fundingAccount, address account, uint256 amount)
external {
if (account == address(0)) revert ZeroAddress();
if (amount == 0) revert ZeroAmount();
_validateHandler();
tokenToLock.safeTransferFrom(fundingAccount, address(this), amount);
_lock(account, amount, Relock.AddToPending);
}
```

The following example depicts the diversion between the expected and actual amount.

Tax	Amount	Expected	Actual
No Tax	100	100	100
10% Tax	100	100	90

Recommendation

The team is advised to take into consideration the actual amount that has been transferred instead of the expected.

It is important to note that an ERC20 transfer tax is not a standard feature of the ERC20 specification, and it is not universally implemented by all ERC20 contracts. Therefore, the contract could produce the actual amount by calculating the difference between the transfer call.

```
Actual Transferred Amount = Balance After Transfer - Balance Before Transfer
```

PISR - Potentially Inconsistent Staking Records

Criticality	Minor / Informative
Location	PlutusRouterV2.sol#L132,156,195
Status	Unresolved

Description

The contract includes the methods `stakeAndLockPls` , `stakeAndLockPlsWeth` , and `stakeEsPls` , which are designed to be invoked exclusively by the migrator during normal operation. Internally, these methods call the `_stake` function, passing `msg.sender` as both the `_fundingAccount` and `_account` parameters. In this case, the migrator is set as both the `_fundingAccount` and `_account` . This duplication may create ambiguity between the actual owner of the staked assets and the entity initiating the staking process, potentially leading to inconsistencies in asset ownership tracking.

```
_stake(msg.sender, msg.sender, pls, _amount, stakedPlsTracker,
bonusPlsTracker, plsCheckpointer);
_stake(msg.sender, msg.sender, plsWeth, _amount, stakedPlsWethTracker,
bonusPlsWethTracker, plsWethCheckpointer);
_stake(msg.sender, msg.sender, esPls, _amount, stakedEsPlsTracker,

bonusEsPlsTracker, esPlsCheckpointer);
```

Recommendation

The team should ensure that assigning the migrator as both the funding account and the owner of the staking operation aligns with the intended functionality. At the time of the audit, the migrator is not a known contract that interacts with these methods and can be further adapted to any arbitrary implementation.

PIO - Potentially Ineffective Overrides

Criticality	Minor / Informative
Location	Vester.sol#L268,272
Status	Unresolved

Description

The contract attempts to enforce non-transferability by overriding the `_transfer` and `_approve` functions, aiming to prevent token holders from transferring or approving transfers of their tokens. However, in some versions of the OpenZeppelin library, the `transfer()` function internally calls `_update` instead of `_transfer`. Since the contract does not override `_update`, this may allow transfers to occur despite the intended restrictions. Additionally, the contract does not specify a fixed version of the OpenZeppelin library, which increases the risk of unexpected behavior due to inconsistencies with the library's internal implementation.

```
function _transfer(address, /*from*/ address, /*to*/ uint256 /*amount*/ )
internal view override {
revert FAILED(string.concat(symbol(), ": ", "non-transferrable"));
}
```

Recommendation

The team is advised to specify an exact version of the OpenZeppelin library in the project's dependencies and override the specific internal functions. This ensures consistent behavior and prevents unexpected issues caused by library incompatibilities.

ST - Stops Transactions

Criticality	Minor / Informative
Location	XPlutusToken.sol#L113,117
Status	Acknowledged

Description

The contract owner has the authority to stop transactions for all users. The owner may take advantage of it by calling the `pause` function.

```
function pause() external onlyRole(DEFAULT_ADMIN_ROLE) {
_pause();
}

function unpause() external onlyRole(DEFAULT_ADMIN_ROLE) {
_unpause();
}
```

Recommendation

The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

UTPD - Unverified Third Party Dependencies

Criticality	Minor / Informative
Location	Vester.sol#L52,53,54,55 PlutusRouterV2.sol#L410,418
Status	Unresolved

Description

The contract uses an external contract in order to determine the transaction's flow. The external contract is untrusted. As a result, it may produce security issues and harm the transactions.

```
esToken = _esToken;
pairToken = _pairToken;
claimableToken = _claimableToken;
rewardTracker = _rewardTracker;
```

```
mpPls = _mpPls;
esPls = _esPls;

stakedPlsTracker = _plsTracker.staked;
bonusPlsTracker = _plsTracker.bonus;
lockedPls = ILockedToken(_plsTracker.locked);
plsCheckpointer = _plsTracker.checkpointer;

stakedPlsWethTracker = _plsWethTracker.staked;
bonusPlsWethTracker = _plsWethTracker.bonus;
lockedPlsWeth = ILockedToken(_plsWethTracker.locked);
plsWethCheckpointer = _plsWethTracker.checkpointer;

stakedEsPlsTracker = _esPlsTracker.staked;
bonusEsPlsTracker = _esPlsTracker.bonus;
esPlsCheckpointer = _esPlsTracker.checkpointer;

mpPlsTracker = IBonusTracker(_mpPlsTracker);
mpPlsCheckpointer = _mpPlsCheckpointer;
```

```
function addCallback(address callback) external onlyOwner {
bool added = callbacks.add(callback);

if (!added) {
revert FAILED("PlutusRouter: Callback already registered");
}
}

function removeCallback(address callback) external onlyOwner {
bool removed = callbacks.remove(callback);

if (!removed) {
revert FAILED("PlutusRouter: Callback not found");
}
}
```

Recommendation

The contract should use a trusted external source. A trusted source could be either a commonly recognized or an audited contract. The pointing addresses should not be able to change after the initialization.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	XPlutusToken.sol#L27 PlsMigration.sol#L20,21,35
Status	Unresolved

Description

State variables can be declared as constant using the constant keyword. This means that the value of the state variable cannot be changed after it has been set. Additionally, the constant variables decrease gas consumption of the corresponding transaction.

```
uint256 public vestIndex
IPlutusToken public plutusToken
IXPlutusToken public xPlutus
uint256 public migrationPeriod
```

Recommendation

Constant state variables can be useful when the contract wants to ensure that the value of a state variable cannot be changed by any function in the contract. This can be useful for storing values that are important to the contract's behavior, such as the contract's address or the maximum number of times a certain function can be called. The team is advised to add the constant keyword to state variables that never change.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	XPlutusToken.sol#L41,48 Vester.sol#L46,64,68,72,76,80,85 PlutusRouterV2.sol#L65,76,77,78,79,80,81,82,118,122,147,151,162,187,191,200,210,389,394,398,402,426 LockedToken.sol#L34,46,292
Status	Unresolved

Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```
uint256[50] private __gap
address _plutus
address _initialAuthority
address _esToken
address _claimableToken
address _rewardTracker
address _pairToken
address _plutusToken
address _xPlutusToken
bool _isActive
address _handler
bool _hasMaxVestableAmount
uint256 _amount
address _account

...
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.
Find more information on the Solidity documentation
https://docs.soliditylang.org/en/stable/style-guide.html#naming-conventions.

L11 - Unnecessary Boolean equality

Criticality	Minor / Informative
Location	LockedToken.sol#L92,267
Status	Unresolved

Description

Boolean equality is unnecessary when comparing two boolean values. This is because a boolean value is either true or false, and there is no need to compare two values that are already known to be either true or false.

it's important to be aware of the types of variables and expressions that are being used in the contract's code, as this can affect the contract's behavior and performance. The comparison to boolean constants is redundant. Boolean constants can be used directly and do not need to be compared to true or false.

```
isAutoextendDisabled[account] == false
```

Recommendation

Using the boolean value itself is clearer and more concise, and it is generally considered good practice to avoid unnecessary boolean equalities in Solidity code.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	PlutusRouterV2.sol#L436 LockedToken.sol#L144
Status	Unresolved

Description

Using an uninitialized local variable can lead to unpredictable behavior and potentially cause errors in the contract. It's important to always initialize local variables with appropriate values before using them.

```
uint256 plsUnlockedFromPlsWeth
uint256 idx
```

Recommendation

By initializing local variables before using them, the contract ensures that the functions behave as expected and avoid potential issues.

L15 - Local Scope Variable Shadowing

Criticality	Minor / Informative
Location	interfaces/ILockedToken.sol#L13
Status	Unresolved

Description

Local scope variable shadowing occurs when a local variable with the same name as a variable in an outer scope is declared within a function or code block. When this happens, the local variable "shadows" the outer variable, meaning that it takes precedence over the outer variable within the scope in which it is declared.

```
bool isAutoextendDisabled
```

Recommendation

It's important to be aware of shadowing when working with local variables, as it can lead to confusion and unintended consequences if not used correctly. It's generally a good idea to choose unique names for local variables to avoid shadowing outer variables and causing confusion.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	Vester.sol#L52,53,54,55 PlutusRouterV2.sol#L89,90,107,390,399
Status	Unresolved

Description

The contract performs operations on variables that have been configured on user-supplied input. These variables are missing of proper check for the case where a value is zero. This can lead to problems when the contract is executed, as certain actions may not be properly handled when the value is zero.

```
esToken = _esToken
pairToken = _pairToken
claimableToken = _claimableToken
rewardTracker = _rewardTracker
mpPls = _mpPls
esPls = _esPls
mpPlsCheckpointer = _mpPlsCheckpointer
migrator = _migrator
kicker = _kicker
```

Recommendation

By adding the proper check, the contract will not allow the variables to be configured with zero value. This will ensure that the contract can handle all possible input values and avoid unexpected behavior or errors. Hence, it can help to prevent the contract from being exploited or operating unexpectedly.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	PlutusRouterV2.sol#L395
Status	Unresolved

Description

According to the ERC20 specification, the transfer methods should be checked if the result is successful. Otherwise, the contract may wrongly assume that the transfer has been established.

```
IERC20Upgradeable(_erc20).transfer(owner(), _amount)
```

Recommendation

The contract should check if the result of the transfer methods is successful. The team is advised to check the SafeERC20 library from the Openzeppelin library.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
XPlutusToken	Implementation	IXPlutusToken, Initializable, ERC20Upgradeable, ERC20PausableUpgradeable, AccessControlUpgradeable, UUPSUpgradeable, ReentrancyGuardUpgradeable		
		Public	✓	-
	initialize	Public	✓	initializer
	vestCount	Public		-
	tokenOfOwnerByIndex	Public		-
	totalVests	Public		-
	vestByIndex	Public		-
	updateExcessReceiver	External	✓	onlyRole
	updateRedeemSettings	External	✓	onlyRole
	updateWhitelist	External	✓	onlyRole
	pause	External	✓	onlyRole
	unpause	External	✓	onlyRole

	getPlutusByVestingDuration	Public		-
	getAccountVests	External		-
	convert	External	✓	nonReentrant
	vest	External	✓	nonReentrant
	redeem	External	✓	nonReentrant
	cancelVest	External	✓	nonReentrant
	_convert	Internal	✓	
	_redeem	Internal	✓	
	_beforeTokenTransfer	Internal	✓	
	_addVestToOwnerEnumeration	Private	✓	
	_addVestToAllVestsEnumeration	Private	✓	
	_removeVestFromOwnerEnumeration	Private	✓	
	_removeVestFromAllVestsEnumeration	Private	✓	
	vests	External		-
	redeemSettings	External		-
	_authorizeUpgrade	Internal	✓	onlyRole
Vester	Implementation	IVester, ERC20Upgradeable, Ownable2StepUpgradeable, UUPSUpgradeable, ReentrancyGuardUpgradeable		
		Public	✓	-

	initialize	Public	✓	initializer
	_authorizeUpgrade	Internal	✓	onlyOwner
	setPlutusToken	External	✓	onlyOwner
	setXPlutusToken	External	✓	onlyOwner
	setHandler	External	✓	onlyOwner
	setHasMaxVestableAmount	External	✓	onlyOwner
	deposit	External	✓	nonReentrant
	depositForAccount	External	✓	nonReentrant
	hasRewardTracker	Public		-
	hasPairToken	Public		-
	_deposit	Private	✓	
	getPairAmount	Public		-
	getCombinedAverageStakedAmount	Public		-
	getMaxVestableAmount	Public		-
	_updateVesting	Private	✓	
	claim	External	✓	nonReentrant
	claimForAccount	External	✓	nonReentrant
	getVestedAmount	Public		-
	_getNextClaimableAmount	Private		
	claimable	Public		-
	_claim	Private	✓	
	_validateHandler	Private		
	_transfer	Internal		

	_approve	Internal		
PlutusRouterV2	Implementation	IPlutusRouterV2, IErrors, Initializable, Ownable2StepUpgradeable, UUPSUpgradeable, PausableUpgradeable, ReentrancyGuardUpgradeable		
	setShutdown	External	✓	onlyOwner
		Public	✓	-
	initialize	Public	✓	initializer
	delegateToSelf	External	✓	nonReentrant whenNotPaused onlyMigrator
	toggleAutoExtend	External	✓	nonReentrant whenNotPaused
	stakeAndLockPls	External	✓	nonReentrant whenNotPaused onlyMigrator
	unlockAndUnstakePls	External	✓	nonReentrant whenNotPaused onlyMigrator
	unlockAndUnstakePlsFor	External	✓	nonReentrant whenNotPaused onlyMigrator
	stakeAndLockPlsWeth	External	✓	nonReentrant whenNotPaused onlyMigrator
	boot	External	✓	nonReentrant whenNotPaused onlyMigrator
	unlockAndUnstakePlsWeth	External	✓	nonReentrant whenNotPaused onlyMigrator

	unlockAndUnstakePlsWethFor	External	✓	nonReentrant whenNotPaused onlyMigrator
	stakeEsPls	External	✓	nonReentrant whenNotPaused onlyMigrator
	unstakeEsPls	External	✓	nonReentrant whenNotPaused onlyMigrator
	unstakeEsPlsFor	External	✓	nonReentrant whenNotPaused onlyMigrator
	claimAndStakeMpPls	External	✓	nonReentrant whenNotPaused onlyMigrator
	claimEsPls	External	✓	nonReentrant whenNotPaused onlyMigrator
	_claimEsPls	Internal	✓	
	_callbackAction	Private	✓	
	_unlockAndUnstakePlsWeth	Private	✓	
	_unlockAndUnstakePls	Private	✓	
	_autoExtendExpiredLocks	Private	✓	
	_claimAllAndStakeMpPls	Private	✓	
	_claimAndStakeMpPlsFor	Private	✓	
	_unstake	Private	✓	
	_reduceMps	Private	✓	
	_stake	Private	✓	
	getCallback	Public		-
	getAllCallbacks	Public		-
	_authorizeUpgrade	Internal	✓	onlyOwner

	setMigrator	External	✓	onlyOwner
	recoverErc20	External	✓	onlyOwner
	setKicker	External	✓	onlyOwner
	setPaused	External	✓	onlyOwner
	addCallback	External	✓	onlyOwner
	removeCallback	External	✓	onlyOwner
	closeAllPositions	External	✓	nonReentrant whenNotPaused onlyMigrator
PlsMigration	Implementation	IPlsMigration, Initializable, UUPSUpgradeable, AccessControlUpgradeable		
		Public	✓	-
	initialize	Public	✓	initializer
	setmigrationPeriod	External	✓	onlyRole
	updateTokenMultiplier	External	✓	onlyRole
	migratePls	External	✓	beforeDeadline
	migrateToXPlutus	External	✓	beforeDeadline
	_handleOtherTokensMigration	Internal	✓	
	_ensurePlutusTokens	Internal	✓	
	getPlsMigrationPreview	External		-
	getOtherTokensMigrationPreview	External		-
	getMigrationPreview	External		-

	recoverERC20	External	✓	onlyRole
	setupMigration	External	✓	onlyRole
	_authorizeUpgrade	Internal	✓	onlyRole
LockedToken	Implementation	ILockedToken, IErrors, Ownable2StepUpgradeable, UUPSUpgradeable, ReentrancyGuardUpgradeable		
		Public	✓	-
	initialize	Public	✓	initializer
	toggleAutoExtendOnBehalf	External	✓	nonReentrant
	shutdown	External	✓	onlyOwner
	lockedBalanceOf	External		-
	lockedBalanceOfExclPending	Public		-
	activeBalanceOf	External		-
	balanceOf	Public		-
	pendingLockOf	External		-
	lockedBalances	External		-
	getCurrentEpoch	Public		-
	lock	External	✓	-
	_lock	Internal	✓	
	_processExpiredLocks	Internal	✓	
	withdrawExpiredLocksOnBehalf	External	✓	nonReentrant

	processExpiredLocksOnBehalf	External	✓	nonReentrant
	_toUint224	Internal		
	_toUint32	Internal		
	_validateHandler	Internal		
	setHandler	External	✓	onlyOwner
	_authorizeUpgrade	Internal	✓	onlyOwner
IErrors	Interface			
ITracker	Interface			
	stakedAmounts	External		-
	totalSupply	External		-
IBaseToken	Interface	IErrors		
	inPrivateTransferMode	External		-
	isHandler	External		-
IBaseMintableToken	Interface	IBaseToken		
	mint	External	✓	-
	burn	External	✓	-
	setMinter	External	✓	-
	isMinter	External		-

	setBurner	External	✓	-
	isBurner	External		-
IBaseDistributorShared	Interface			
	tokensPerSecond	External		-
	rewardToken	External		-
IBaseDistributor	Interface	IBaseDistributorShared, IErrors		
	pendingRewards	External		-
	distribute	External	✓	-
	getRate	External		-
IRewardDistributor	Interface	IBaseDistributor		
	setTokensPerSecond	External	✓	-
IBonusDistributor	Interface	IBaseDistributor		
	setBonusMultiplier	External	✓	-
IRewardTracker	Interface	IBaseDistributorShared		
	stakedSynthAmounts	External		-
	distributor	External		-

	depositBalances	External		-
	stakedAmounts	External		-
	updateRewards	External	✓	-
	stake	External	✓	-
	stakeForAccount	External	✓	-
	unstake	External	✓	-
	unstakeForAccount	External	✓	-
	claim	External	✓	-
	claimForAccount	External	✓	-
	claimable	External		-
	averageStakedAmounts	External		-
	cumulativeRewards	External		-
IVester	Interface			
	claimForAccount	External	✓	-
	transferredAverageStakedAmounts	External		-
	transferredCumulativeRewards	External		-
	cumulativeRewardDeductions	External		-
	bonusRewards	External		-
	transferStakeValues	External	✓	-
	setTransferredAverageStakedAmounts	External	✓	-
	setTransferredCumulativeRewards	External	✓	-
	setCumulativeRewardDeductions	External	✓	-

	setBonusRewards	External	✓	-
	getMaxVestableAmount	External		-
	getCombinedAverageStakedAmount	External		-
IPlutusRouterCallback	Interface			
	handleActionBefore	External	✓	-
	handleActionAfter	External	✓	-
IXPlutusToken	Interface			
	convert	External	✓	-
	vest	External	✓	-
	redeem	External	✓	-
	cancelVest	External	✓	-
	getAccountVests	External		-
	getPlutusByVestingDuration	External		-
	vests	External		-
	vestCount	External		-
	tokenOfOwnerByIndex	External		-
	totalVests	External		-
	vestByIndex	External		-
	redeemSettings	External		-

IVester	Interface	IErrors		
	claim	External	✓	-
	claimForAccount	External	✓	-
	getVestedAmount	External		-
	claimable	External		-
	setPlutusToken	External	✓	-
	setXPlutusToken	External	✓	-
IPlutusToken	Interface	IERC20		
	mint	External	✓	-
	burn	External	✓	-
	name	External		-
	symbol	External		-
	decimals	External		-
	owner	External		-
	transferOwnership	External	✓	-
	renounceOwnership	External	✓	-
	setOperator	External	✓	-
IPlutusRouterV2	Interface			
	claimAndStakeMpPls	External	✓	-
	stakeEsPls	External	✓	-

	stakeAndLockPlsWeth	External	✓	-
	stakeAndLockPls	External	✓	-
	unstakeEsPls	External	✓	-
	unlockAndUnstakePls	External	✓	-
	unlockAndUnstakePlsWeth	External	✓	-
	unlockAndUnstakePlsWethFor	External	✓	-
	unlockAndUnstakePlsFor	External	✓	-
	unstakeEsPlsFor	External	✓	-
	closeAllPositions	External	✓	-
	setMigrator	External	✓	-
	setShutdown	External	✓	-
IPlsMigration	Interface			
ILockedToken	Interface			
	lock	External	✓	-
	withdrawExpiredLocksOnBehalf	External	✓	-
	setHandler	External	✓	-
	processExpiredLocksOnBehalf	External	✓	-
	isAutoextendDisabled	External		-
	lockedBalanceOfExclPending	External		-
	shutdown	External	✓	-
	toggleAutoExtendOnBehalf	External	✓	-

	lockedBalanceOf	External		-
	balanceOf	External		-
	activeBalanceOf	External		-
	lockedBalances	External		-
ICheckPointer	Interface			
	isDelegationEnabled	External		-
	increment	External	✓	-
	decrement	External	✓	-
	totalSupply	External		-
	getTotalSupplyWithMultiplier	External		-
	getPastTotalSupplyWithMultiplier	External		-
	getVotesWithMultiplier	External		-
	getPastVotesWithMultiplier	External		-
	getMultiplier	External		-
	getPastMultiplier	External		-
	delegateOnBehalf	External	✓	-
ICamelotPair	Interface			
	name	External		-
	symbol	External		-
	decimals	External		-
	totalSupply	External		-

	balanceOf	External		-
	allowance	External		-
	approve	External	✓	-
	transfer	External	✓	-
	transferFrom	External	✓	-
	DOMAIN_SEPARATOR	External		-
	PERMIT_TYPEHASH	External		-
	nonces	External		-
	permit	External	✓	-
	MINIMUM_LIQUIDITY	External		-
	factory	External		-
	token0	External		-
	token1	External		-
	getReserves	External		-
	getAmountOut	External		-
	kLast	External		-
	setFeePercent	External	✓	-
	mint	External	✓	-
	burn	External	✓	-
	swap	External	✓	-
	swap	External	✓	-
	skim	External	✓	-
	sync	External	✓	-

	initialize	External	✓	-
IBonusTracker	Interface			
	stakeForAccount	External	✓	-
	unstakeForAccount	External	✓	-
	depositSources	External		-
	stakedAmounts	External		-
	burntAmounts	External		-
	setHandler	External	✓	-

Summary

Plutus contracts implement a migration mechanism for the Plutus ecosystem. This audit investigates security issues, business logic concerns and potential improvements.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope's prior written consent. This report is not nor should be considered an “endorsement” or “disapproval” of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

A **TAC Security** Company

The Cyberscope team

cyberscope.io