Cyberscope

A TAC Security Company

Audit Report

plsBera

August 2025

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

Commit : 330f0ea73cde74e226411f1195d808e136887d41

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	19 Aug 2025

Source Files

Filename	SHA256
PlsBeraWhitelist.sol	ec9f6fa62e781c0a8573dfb04bf876f7a4d82f0a0f4f7b6c252e74ed8edcfbaf
PlsBeraToken.sol	4076125a26fbeb5ada8f54426313b38415cb79d30caea107c466ea964fe80d85
PlsBeraStaker.sol	14ea8a8bd1f9888ceafb8ba27e819b916f49dbb162d9e9a76a4ac942b2746016
PlsBeraLPStaker.sol	0966aa4dd3d9c2a5708844264401c220aa98911aa8320af1b090861078863689
BeraStaker.sol	bb8e0e1b1207ebb4463bfd73e4873d96f15b600e6653239bb2ed485b827f4ed1
BeraDepositor.sol	92ec3ebc9f3b81f762d2f824d97901af01dfb6ec18a092cd76bcf9ff937dd4bc
interfaces/IWBera.sol	7855af6b3df2d726e2cc60c815355454f2eddf25b6c100c870b1576a4fdf97aa
interfaces/IWBERAStakerVault.sol	3f971ab7804d405b9b187da42754d30043da556404673b0a9f56d8a5ba4a68f5
interfaces/ITokenMinterMulti.sol	9210c4ed461f97b86f7470a1201ca88cf9a863efbf9451e1a447a80bb17ee6b9
interfaces/IPlsBeraWhitelist.sol	985a2d4d4e4e7c9f6d399f81faea853b43513602fd9e5959a09db22a810fa247

interfaces/IPlsBeraStaker.sol	c8befb780cf6d2341d576907e8b526b3dbb8e3cba32e84a8f5d5b6ca64512a58
interfaces/IPlsBeraLPStaker.sol	792f83d2f598ad51af640fcf8be798d5a6c1cede5ebb15bc86b45ff9b300bc76
interfaces/IBeraStaker.sol	cd8d50516cef8665d93d0e2e697ca6eddae3c81647a46ffbe41c0753faa466a4
interfaces/IBeraDepositor.sol	d2f8db3291d98d9efd352d425958611ee8cee5ba71a17bab58d5ac2f900e995b

Overview

Overview

The PlsBERA suite centers on deposit, staking, reward distribution, token minting, and access control across Bera-based assets. The core on-chain components are `BeraDepositor`, `BeraStaker`, `PlsBeraStaker`, `PlsBeraLPStaker`, `PlsBeraToken`, and `PlsBeraWhitelist`. Together, they enable permissioned deposits of BERA or staked BERA, programmatic conversion and accounting inside a staking vault and configurable, multi-token reward distribution for staking tokens. All contracts are upgradeable via UUPS and protected by role-based access control, with optional pausing and handler/reward distributor roles.

BeraDepositor

`BeraDepositor` is the user entry-point for depositing BERA or pre-staked BERA shares into the system. It validates sender eligibility via a whitelist, funnels assets to `BeraStaker`, and mints `PlsBera` 1:1 to depositors.

- Deposits
  - `deposit(uint256)` and `depositAll()`: transfer `IWBera` from the user to `BeraStaker`.
  - `depositNative()`: accept native currency, wrap to `IWBera`, forward to `BeraStaker`.
  - `depositStakedBera(uint256)`: accept `IWBERAStakerVault` shares, convert via `previewRedeem`, and forward to `BeraStaker` marked as pre-staked.
  - Handler variants allow a privileged role to deposit on behalf of users.
- Minting
  - Mints `PlsBera` to the depositor equal to deposited wBERA amount.
- Controls
  - `HANDLER_ROLE` for meta-deposits; admin can set whitelist, pause/unpause, and upgrade.
  - Enforces a minimum deposit threshold.

BeraStaker

`BeraStaker` holds the core BERA position inside an external staker vault. It tracks the total principal staked on behalf of the system, stakes incoming wBERA into `IWBERAStakerVault`, and exposes a claim mechanism for accrued yield.

- Staking
  - `stake(uint256, bool)`: if not already pre-staked, approves and deposits wBERA into the vault; increments `totalStaked`.
- Rewards
  - `handleClaim()`: computes redeemable vault assets vs principal; if positive, transfers the excess as vault shares back to the caller (handler), enabling downstream distribution without touching principal.
- Controls
  - `HANDLER_ROLE` gates staking/claim; admin-only recovery and upgrades.

PlsBeraStaker

`PlsBeraStaker` enables staking of the `PlsBera` token itself with multi-reward support. It tracks balances and distributes arbitrary ERC20 rewards over a configurable duration using a standard reward-per-token accounting model.

- User actions
  - `stake(uint256)` and `withdraw(uint256)`: move `PlsBera` into/out of the contract.
  - `getReward()`: claim all accumulated rewards across all configured reward tokens.
- Rewards
  - `notifyRewardAmount(token, amount)`: by `REWARD_DISTRIBUTOR_ROLE`, funds and (re)starts ongoing emissions for that token; supports compounding leftover rewards mid-period.
  - Duration configurable; prevents changes during an active reward period.
- Controls
  - Pausable, upgradeable; admin/reward distributor roles; prevents recovery of active reward tokens.

PlsBeraLPStaker

`PlsBeraLPStaker` mirrors `PlsBeraStaker` for the protocol's LP token. It supports the same multi-reward distribution mechanics and lifecycle, but the staked asset is the PlsBera LP ERC20.

- User actions
  - `stake(uint256)`, `withdraw(uint256)`, `getReward()`.
- Rewards
  - Identical reward accounting and distributor flow as `PlsBeraStaker`, including duration management and leftover handling.
- Controls
  - Pausable, upgradeable with admin and reward distributor roles; safe token recovery excluding active reward tokens.

PlsBeraToken

`PlsBeraToken` is an upgradeable ERC20 with permit support and controlled mint/burn. It serves as the protocol's primary token minted to depositors via `BeraDepositor` and possibly used across staking programs.

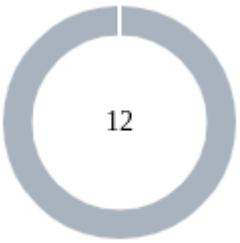
- ERC20 + EIP-2612 permit via `ERC20PermitUpgradeable`.
- Roles
  - `MINTER_ROLE` for controlled `mint(address,uint256)` and `burn(address,uint256)`.
  - Admin can authorize upgrades.
- Initialization sets name/symbol "PlsBera by Plutus" (`PlsBERA`).

PlsBeraWhitelist

`PlsBeraWhitelist` is a minimal allowlist module used by `BeraDepositor` to gate who can deposit (and also supports handler flow). It's upgradeable and role-gated.

- Admin can add/remove addresses.
- Queried by `BeraDepositor` for `isWhitelisted(address)` checks.
- UUPS upgrade authorization restricted to admin.

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	12

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	12	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
	PSTR	Potential Staked Token Recovery	Unresolved
	BT	Burns Tokens	Unresolved
	CCR	Contract Centralization Risk	Unresolved
	MT	Mints Tokens	Unresolved
	MTM	Missing Transfer Mechanism	Unresolved
	MWF	Missing Withdrawal Functionality	Unresolved
	PTAI	Potential Transfer Amount Inconsistency	Unresolved
	PLR	Potentially Locked Rewards	Unresolved
	TSI	Tokens Sufficiency Insurance	Unresolved
	UTPD	Unverified Third Party Dependencies	Unresolved
	L04	Conformance to Solidity Naming Conventions	Unresolved
	L15	Local Scope Variable Shadowing	Unresolved

PSTR - Potential Staked Token Recovery

Criticality	Minor / Informative
Location	PlsBeraStaker.sol#L243 PlsBeraLPStaker.sol#L207
Status	Unresolved

Description

The `recoverERC20` function is designed to allow the contract administrator to recover ERC20 tokens sent to the contract. However, it only prevents the recovery of reward tokens and does not exclude the primary staked token. This could allow the administrator to withdraw the staked token, potentially compromising the integrity of the staking mechanism.

```
function recoverERC20(address tokenAddress, uint256 tokenAmount) external
onlyRole(DEFAULT_ADMIN_ROLE) {
    if (rewardTokens.contains(tokenAddress)) {
        revert PBS_WithdrawRewardToken();
    }
    IERC20(tokenAddress).safeTransfer(msg.sender, tokenAmount);
    emit Recovered(tokenAddress, tokenAmount);
}
```

Recommendation

To prevent potential misuse, the function should also restrict the recovery of the primary staked token. This ensures that staked assets remain secure and the staking mechanism retains its integrity.

BT - Burns Tokens

Criticality	Minor / Informative
Location	PlsBeraToken.sol#L40
Status	Unresolved

Description

The contract's `MINTER_ROLE` has the authority to burn tokens from a specific address. The role may take advantage of it by calling the `burn` function. As a result, the targeted address will lose the corresponding tokens.

```
function burn(address _from, uint256 _amount) external override
onlyRole(MINTER_ROLE) {
_burn(_from, _amount);
}
```

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

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	PlsBeraStaker.sol#L161,204,206,214,226 PlsBeraLPStaker.sol#L162,205,207,215,227 BeraStaker.sol#L44,53,67,69
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 notifyRewardAmount(address _rewardsToken, uint256 reward)
external
nonReentrant
onlyRole(REWARD_DISTRIBUTOR_ROLE)
updateReward(address(0), _rewardsToken)
{...}
function _authorizeUpgrade(address newImplementation) internal override
onlyRole(DEFAULT_ADMIN_ROLE) { }
function setRewardsDuration(uint256 rewardsDuration_) external
onlyRole(DEFAULT_ADMIN_ROLE) {...}
function setPaused(bool _pauseContract) external onlyRole(DEFAULT_ADMIN_ROLE)
{...}
function stake(uint256 _amount, bool isStakedWBera) external override
onlyRole(HANDLER_ROLE) {...}
function handleClaim() external override nonReentrant onlyRole(HANDLER_ROLE)
returns (uint256 rewardAmount, uint256 shares_) {...}
function _authorizeUpgrade(address newImplementation) internal virtual
override onlyRole(DEFAULT_ADMIN_ROLE) {...}
function recoverErc20(IERC20 _erc20, uint256 _amount) external override
onlyRole(DEFAULT_ADMIN_ROLE) {...}
```

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.

MT - Mints Tokens

Criticality	Minor / Informative
Location	PlsBeraToken.sol#L36
Status	Unresolved

Description

The contract's `MINTER_ROLE` has the authority to mint tokens. The role may take advantage of it by calling the `mint` function. As a result, the contract tokens will be highly inflated.

```
function mint(address _to, uint256 _amount) external override
onlyRole(MINTER_ROLE) {
_mint(_to, _amount);
}
```

Recommendation

The team should carefully manage the private keys of the `MINTER_ROLE` 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 role, which will eliminate the threats but it is non-reversible.

MTM - Missing Transfer Mechanism

Criticality	Minor / Informative
Location	BeraStaker.sol#L44
Status	Unresolved

Description

The contract includes a `stake` function that can be invoked by addresses with the `HANDLER_ROLE`. When executed, this function increases the `totalStaked` variable. If the `isStakedWBera` flag is set to `false`, it also calls the `deposit` method on the `beraToken` contract.

However, the function does not implement any mechanism to transfer tokens from the user to the contract. As a result, the contract may not actually hold the staked funds. Additionally, if the `isStakedWBera` flag is set to `true`, the `totalStaked` value increases without any corresponding token transfer, which can lead to inaccurate or misleading accounting of staked assets.

```
function stake(uint256 _amount, bool isStakedWBera) external override
onlyRole(HANDLER_ROLE) {
if (!isStakedWBera) {
beraToken.approve(address(stakedWBera), _amount);
stakedWBera.deposit(_amount, address(this));
}
totalStaked += _amount;
emit Staked(msg.sender, _amount);
}
```

Recommendation

The team is advised to revisit the current implementation to ensure it aligns with the intended design. The contract should hold the actual staked assets to maintain accurate accounting and uphold trust in the staking mechanism.

MWF - Missing Withdrawal Functionality

Criticality	Minor / Informative
Location	BeraStaker.sol
Status	Unresolved

Description

The contract lacks a withdrawal functionality, preventing authorized roles from retrieving staked tokens. While the contract allows for transferring reward shares under certain conditions, there is no mechanism to withdraw the staked amount. This omission limits the contract's usability and may restrict users from reclaiming assets.

```
function handleClaim() external override nonReentrant onlyRole(HANDLER_ROLE)
returns (uint256 rewardAmount, uint256 shares_) {
uint256 totalShares = stakedWBera.balanceOf(address(this));

uint256 redeemableAmount = stakedWBera.previewRedeem(totalShares);
if (redeemableAmount > totalStaked) {
rewardAmount = redeemableAmount - totalStaked;
shares_ = stakedWBera.convertToShares(rewardAmount);
stakedWBera.safeTransfer(msg.sender, shares_);
}
}
```

Recommendation

The team is advised to implement a withdrawal function that allows authorized roles to redeem and retrieve the originally staked tokens. This ensures that funds can be retrieved, improving the contract’s usability.

PTAI - Potential Transfer Amount Inconsistency

Criticality	Minor / Informative
Location	PlsBeraStaker.sol#L122,129,150,167,210 PlsBeraLPStaker.sol#L123,130,151,168,211
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.

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

```
IERC20(_rewardsToken).safeTransferFrom(msg.sender, address(this),
reward);
```

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
```

PLR - Potentially Locked Rewards

Criticality	Minor / Informative
Location	BeraStaker.sol#L53
Status	Unresolved

Description

The `stake` function permits the handler to arbitrarily increase the `totalStaked` value without any validation or checks. This can lead to a mismatch between `totalStaked` and the actual redeemable rewards held by the contract. As a result, when the `handleClaim` function is invoked, the calculated reward amount may be insufficient to cover the recorded `totalStaked` , potentially rendering rewards unclaimable.

```
function handleClaim() external override nonReentrant onlyRole(HANDLER_ROLE)
returns (uint256 rewardAmount, uint256 shares_) {
uint256 totalShares = stakedWBera.balanceOf(address(this));

uint256 redeemableAmount = stakedWBera.previewRedeem(totalShares);
if (redeemableAmount > totalStaked) {
rewardAmount = redeemableAmount - totalStaked;
shares_ = stakedWBera.convertToShares(rewardAmount);
stakedWBera.safeTransfer(msg.sender, shares_);
}
}
```

Recommendation

To prevent inconsistencies, the team is advised to implement validation in the `stake` function to ensure that any increase in `totalStaked` corresponds to an actual transfer of staked tokens. This ensures that `totalStaked` accurately reflects the contract's real holdings, maintaining reward integrity and preventing unclaimable rewards.

TSI - Tokens Sufficiency Insurance

Criticality	Minor / Informative
Location	PlsBeraStaker.sol#L158 PlsBeraLPStaker.sol#L159 BeraStaker.sol#L44
Status	Unresolved

Description

The tokens are not held within the contract itself. Instead, the contract is designed to provide the tokens from an external administrator. While external administration can provide flexibility, it introduces a dependency on the administrator's actions, which can lead to various issues and centralization risks.

```
function notifyRewardAmount(address _rewardsToken, uint256 reward)
external
nonReentrant
onlyRole(REWARD_DISTRIBUTOR_ROLE)
updateReward(address(0), _rewardsToken)
{...}
```

```
function stake(uint256 _amount, bool isStakedWBera) external override
onlyRole(HANDLER_ROLE) {
if (!isStakedWBera) {
beraToken.approve(address(stakedWBera), _amount);
stakedWBera.deposit(_amount, address(this));
}
totalStaked += _amount;
emit Staked(msg.sender, _amount);
}
```

Recommendation

It is recommended to consider implementing a more decentralized and automated approach for handling the contract tokens. One possible solution is to hold the tokens within the contract itself. If the contract guarantees the process it can enhance its reliability, security, and participant trust, ultimately leading to a more successful and efficient process.

UTPD - Unverified Third Party Dependencies

Criticality	Minor / Informative
Location	BeraStaker.sol#L40,41 BeraDepositor.sol#L50,51
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.

```
beraToken = IERC20(_beraToken);
stakedWBera = IWBERAStakerVault(_stakedWBera);
```

```
beraToken = IWBera(_bera);
stakedBeraToken = IWBERAStakerVault(_stakedBera);
```

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.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	PlsBeraWhitelist.sol#L17,23,28 PlsBeraToken.sol#L36,40 PlsBeraStaker.sol#L46,59,92,96,109,114,158,226 PlsBeraLPStaker.sol#L46,59,93,97,110,115,159,227 interfaces/IPlsBeraStaker.sol#L21 interfaces/IPlsBeraLPStaker.sol#L21 interfaces/IBeraStaker.sol#L12 interfaces/IBeraDepositor.sol#L37,38 BeraStaker.sol#L33,44,69 BeraDepositor.sol#L37,44,61,82,90,99,136,144,148
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.

```
address _admin
address _addr
address _to
uint256 _amount
address _from
uint256 public _rewardsDuration
address _plsBeraToken
address _rewardsToken
bool _pauseContract
address _plsBeraLPToken
function REWARD_DISTRIBUTOR_ROLE() external view returns (bytes32);
function HANDLER_ROLE() external view returns (bytes32);
function MIN_DEPOSIT() external view returns (uint256);
address _beraToken

...
```

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.

L15 - Local Scope Variable Shadowing

Criticality	Minor / Informative
Location	interfaces/IPlsBeraStaker.sol#L39 interfaces/IPlsBeraLPStaker.sol#L39
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.

```
uint256 rewardsDuration
```

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.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
PlsBeraWhitelist	Implementation	Initializable, AccessControlUpgradeable, UUPSUpgradeable, IPlsBeraWhitelist		
		Public	✓	-
	initialize	Public	✓	initializer
	whitelistAdd	External	✓	onlyRole
	whitelistRemove	External	✓	onlyRole
	_authorizeUpgrade	Internal	✓	onlyRole
PlsBeraToken	Implementation	Initializable, ERC20UpgradeableERC20PermitUpgradeable, AccessControlUpgradeable, UUPSUpgradeable, ITokenMinterMulti		
		Public	✓	-
	initialize	Public	✓	initializer
	mint	External	✓	onlyRole
	burn	External	✓	onlyRole

	nonces	Public		-
	_authorizeUpgrade	Internal	✓	onlyRole
PlsBeraStaker	Implementation	Initializable, ReentrancyGuardUpgradeable, PausableUpgradeable, AccessControlUpgradeable, UUPSUpgradeable, IPlsBeraStaker		
		Public	✓	-
	initialize	Public	✓	initializer
	getRewardTokens	Public		-
	totalSupply	External		-
	balanceOf	External		-
	rewardsDuration	External		-
	lastTimeRewardApplicable	Public		-
	rewardPerToken	Public		-
	earned	Public		-
	getRewardForDuration	External		-
	stake	External	✓	nonReentrant
	withdraw	External	✓	nonReentrant
	_handleDeposit	Internal	✓	whenNotPaused updateReward
	_handleWithdraw	Internal	✓	updateReward

	getReward	Public	✓	nonReentrant updateReward
	notifyRewardAmount	External	✓	nonReentrant onlyRole updateReward
	_authorizeUpgrade	Internal	✓	onlyRole
	recoverERC20	External	✓	onlyRole
	setRewardsDuration	External	✓	onlyRole
	setPaused	External	✓	onlyRole
PlsBeraLPStaker	Implementation	Initializable, ReentrancyGuardUpgradeable, PausableUpgradeable, AccessControlUpgradeable, UUPSUpgradeable, IPlsBeraLPStaker		
		Public	✓	-
	initialize	Public	✓	initializer
	getRewardTokens	Public		-
	totalSupply	External		-
	balanceOf	External		-
	rewardsDuration	External		-
	lastTimeRewardApplicable	Public		-
	rewardPerToken	Public		-
	earned	Public		-
	getRewardForDuration	External		-

	stake	External	✓	nonReentrant
	withdraw	External	✓	nonReentrant
	_handleDeposit	Internal	✓	whenNotPaused updateReward
	_handleWithdraw	Internal	✓	updateReward
	getReward	Public	✓	nonReentrant updateReward
	notifyRewardAmount	External	✓	nonReentrant onlyRole updateReward
	_authorizeUpgrade	Internal	✓	onlyRole
	recoverERC20	External	✓	onlyRole
	setRewardsDuration	External	✓	onlyRole
	setPaused	External	✓	onlyRole
BeraStaker	Implementation	Initializable, AccessControlUpgradeable, UUPSUpgradeable, ReentrancyGuardUpgradeable, IBeraStaker		
		Public	✓	-
	initialize	Public	✓	initializer
	stake	External	✓	onlyRole
	handleClaim	External	✓	nonReentrant onlyRole
	_authorizeUpgrade	Internal	✓	onlyRole
	recoverErc20	External	✓	onlyRole

BeraDepositor	Implementation	Initializable, AccessControlUpgradeable, PausableUpgradeable, ReentrancyGuardUpgradeable, UUPSUpgradeable, IBeraDepositor		
		Public	✓	-
	initialize	Public	✓	initializer
	deposit	External	✓	whenNotPaused
	depositNative	External	Payable	whenNotPaused
	depositAll	External	✓	whenNotPaused
	handleDepositFor	External	✓	whenNotPaused onlyRole
	depositStakedBera	External	✓	whenNotPaused
	handleStakedBeraDepositFor	External	✓	whenNotPaused onlyRole
	paused	Public		-
	_deposit	Internal	✓	nonReentrant
	_isEligibleSender	Internal		
	_authorizeUpgrade	Internal	✓	onlyRole
	setWhitelist	External	✓	onlyRole
	recoverErc20	External	✓	onlyRole
	setPaused	External	✓	onlyRole

IPlsBeraWhitelist	Interface	IAccessControl		
	whitelistAdd	External	✓	-
	whitelistRemove	External	✓	-
	isWhitelisted	External		-
IPlsBeraStaker	Interface	IAccessControl		
	REWARD_DISTRIBUTOR_ROLE	External		-
	plsBeraToken	External		-
	getRewardTokens	External		-
	totalSupply	External		-
	balanceOf	External		-
	lastTimeRewardApplicable	External		-
	rewardPerToken	External		-
	earned	External		-
	getRewardForDuration	External		-
	stake	External	✓	-
	withdraw	External	✓	-
	getReward	External	✓	-
	rewardsDuration	External		-
	notifyRewardAmount	External	✓	-
	recoverERC20	External	✓	-

	setRewardsDuration	External	✓	-
	setPaused	External	✓	-
IPlsBeraLPStaker	Interface	IAccessControl		
	REWARD_DISTRIBUTOR_ROLE	External		-
	plsBeraLPToken	External		-
	getRewardTokens	External		-
	totalSupply	External		-
	balanceOf	External		-
	lastTimeRewardApplicable	External		-
	rewardPerToken	External		-
	earned	External		-
	getRewardForDuration	External		-
	rewardsDuration	External		-
	stake	External	✓	-
	withdraw	External	✓	-
	getReward	External	✓	-
	notifyRewardAmount	External	✓	-
	recoverERC20	External	✓	-
	setRewardsDuration	External	✓	-
	setPaused	External	✓	-

IBeraStaker	Interface	IAccessControl		
	HANDLER_ROLE	External		-
	beraToken	External		-
	stakedWBera	External		-
	totalStaked	External		-
	stake	External	✓	-
	handleClaim	External	✓	-
	recoverErc20	External	✓	-
IBeraDepositor	Interface	IAccessControl		
	deposit	External	✓	-
	depositNative	External	Payable	-
	depositAll	External	✓	-
	handleDepositFor	External	✓	-
	setWhitelist	External	✓	-
	recoverErc20	External	✓	-
	setPaused	External	✓	-
	depositStakedBera	External	✓	-
	handleStakedBeraDepositFor	External	✓	-
	beraToken	External		-
	stakedBeraToken	External		-
	beraStaker	External		-

	plsBera	External		-
	whitelist	External		-
	HANDLER_ROLE	External		-
	MIN_DEPOSIT	External		-
	paused	External		-

Summary

plsBera contracts implement token, staking and utility mechanisms. 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