Cyberscope

Audit Report

PPLDAO

November 2025

Network :	BSC Testnet
Address :	0x8A3C8eF8c4A7f5BDCD4ebF50b28Ab8A509ceCc70
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

Explorer	https://bscscan.com/address/0x8a3c8ef8c4a7f5bdcd4ebf50b28ab8a509cecc70

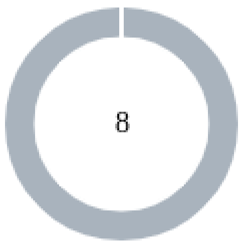
Audit Updates

Initial Audit	29 Oct 2025

Source Files

Filename	SHA256
PPLtesttoken.sol	b4885862cffd691200178f9cb8087d142cace33b8da070806d2bf239914eccf7
OpenZeppelin.sol	53662ec06d5988f189b3c7a95f29a6e441aec0e32702162c3bfc20a9c1eeffe9

Findings Breakdown



🟥	Critical	0
🟧	Medium	0
●	Minor / Informative	8

	Severity	Unresolved	Acknowledged	Resolved	Other
🟥	Critical	0	0	0	0
🟧	Medium	0	0	0	0
●	Minor / Informative	8	0	0	0

Diagnostics

🔴 Critical 🟠 Medium ⚪ Minor / Informative

Severity	Code	Description	Status
⚪	MT	Mints Tokens	Unresolved
⚪	MC	Missing Check	Unresolved
⚪	STR	Start Time Reinitialization	Unresolved
⚪	IMR	Inconsistent Minting Restrictions	Unresolved
⚪	CR	Code Repetition	Unresolved
⚪	CCR	Contract Centralization Risk	Unresolved
⚪	EVR	Early Vesting Release	Unresolved
⚪	L04	Conformance to Solidity Naming Conventions	Unresolved

MT - Mints Tokens

Criticality	Minor / Informative
Location	PPLtesttoken.sol#L229
Status	Unresolved

Description

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

Shell
```
function mintTo(address to, uint256 amount, string calldata
purpose) external onlyOwner {
require(to != address(0), "zero addr");
require(amount > 0, "zero amount");

// Ensure total minted (including vesting-reserved but not
yet minted) never exceeds cap
require(totalSupply() + totalAllocated + amount <=
TOTAL_SUPPLY, "cap exceeded");

_mint(to, amount);
emit OwnerMint(to, amount, purpose);
}
```

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.

MC - Missing Check

Criticality	Minor / Informative
Location	PPLtesttoken.sol#L101
Status	Unresolved

Description

The contract is processing variables that have not been properly sanitized and checked that they form the proper shape. In particular, the `_globalStartTime` parameter is not checked to confirm that it represents a future timestamp. This lack of validation could lead to unintended behavior or potential vulnerabilities if an invalid or past timestamp is provided.

Shell
```
function setGlobalStartTime(uint256 _globalStartTime)
external onlyOwner {
require(_globalStartTime != 0, "invalid start");
globalStartTime = _globalStartTime;
emit GlobalStartTimeSet(_globalStartTime);
}
```

Recommendation

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

STR - Start Time Reinitialization

Criticality	Minor / Informative
Location	PPLtesttoken.sol#L101
Status	Unresolved

Description

The `setGlobalStartTime` function allows the contract owner to update the `globalStartTime` variable multiple times without restriction. This can lead to inconsistent behavior if other components rely on a fixed timestamp. Changing the start time after initialization may allow beneficiaries to access tokens earlier or later than intended, compromising the integrity of the vesting logic.

Shell
```
function setGlobalStartTime(uint256 _globalStartTime)
external onlyOwner {
require(_globalStartTime != 0, "invalid start");
globalStartTime = _globalStartTime;
emit GlobalStartTimeSet(_globalStartTime);

}
```

Recommendation

The team is advised to restrict the `setGlobalStartTime` function so it can only be called once. This ensures the vesting schedule remains consistent and prevents manipulation of the token release timeline.

IMR - Inconsistent Minting Restrictions

Criticality	Minor / Informative
Location	PPLtesttoken.sol#L215,229
Status	Unresolved

Description

The contract uses the `totalAllocated` variable to reserve tokens for future vesting, helping to ensure that the total token supply does not exceed the `TOTAL_SUPPLY` cap. However, when tokens are eventually minted, the `totalAllocated` value is not decreased to reflect the allocation being fulfilled. This results in double-counting, where the same tokens are included in both `totalAllocated` and `totalSupply`.

```Shell
function _increaseAllocated(uint256 amount) internal {
// total future minting (already minted + allocated
yet-to-mint)may not exceed cap
require(totalSupply() + totalAllocated + amount <=
TOTAL_SUPPLY, "cap exceeded");
totalAllocated += amount;
}

function mintTo(address to, uint256 amount, string calldata
purpose) external onlyOwner {
require(to != address(0), "zero addr");
require(amount > 0, "zero amount");

// Ensure total minted (including vesting-reserved but not
yet minted) never exceeds cap
require(totalSupply() + totalAllocated + amount <=
TOTAL_SUPPLY, "cap exceeded");
_mint(to, amount);
emit OwnerMint(to, amount, purpose);}
```

Recommendation

The team is advised to properly decrease the `totalAllocated` value when tokens are minted from previously reserved allocations. This adjustment ensures that the variable accurately reflects only the remaining unfulfilled allocations, preventing double-counting and maintaining consistency with the `TOTAL_SUPPLY` cap.

CR - Code Repetition

Criticality	Minor / Informative
Location	PPLtesttoken.sol#L303,313,319,326,335,345,351,358
Status	Unresolved

Description

The contract contains repetitive code segments. There are potential issues that can arise when using code segments in Solidity. Some of them can lead to issues like gas efficiency, complexity, readability, security, and maintainability of the source code. It is generally a good idea to try to minimize code repetition where possible.

```shell
function _claimableMonthly(VestingSchedule storage s)
internal returns (uint256) {...}
function _calcMonthly(VestingSchedule storage s) internal
view returns (uint256 amount, uint256 intervals, uint256
nowTs) {...}
function _claimableYearlyCliff10(VestingSchedule storage s)
internal returns (uint256) {...}

function _calcYearly(VestingSchedule storage s) internal
view returns (uint256 amount, uint256 intervals, uint256
nowTs) {...}
```

Recommendation

The team is advised to avoid repeating the same code in multiple places, which can make the contract easier to read and maintain. The authors could try to reuse code wherever possible, as this can help reduce the complexity and size of the contract. For instance, the contract could reuse the common code segments in an internal function in order to avoid repeating the same code in multiple places.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	PPLtesttoken.sol#L101,108,134,155,229,261,368,380
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.

Shell
```
function setGlobalStartTime(uint256 _globalStartTime)
external onlyOwner {...}
function setBeneficiaries(
    address _marketing,
    address _foundation,
    address _team,
    address _angels
) external onlyOwner {...}
function setupFixedSchedules() external onlyOwner {...}
function premintMarketing200B() external onlyOwner {...}
function mintTo(address to, uint256 amount, string calldata
purpose) external onlyOwner {...}
function airdropClaimable(address beneficiary) external
onlyOwner {...}
function updateVestingBeneficiary(address oldBeneficiary,
address newBeneficiary) external onlyOwner {...}

function changeContractOwner(address newOwner) external

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.

EVR - Early Vesting Release

Criticality	Minor / Informative
Location	PPLtesttoken.sol#L322,329,354,361
Status	Unresolved

Description

The contract allows tokens to be released immediately upon entering a new vesting period. This behavior results in tokens being distributed as soon as the period begins, which may conflict with expected vesting mechanisms that typically delay rewards until the end of a period. Such immediate distribution can lead to inconsistencies and may not align with standard vesting expectations.

Shell
```
intervals = (nowTs - s.nextReleaseTime) /
s.releaseIntervalSeconds + 1;
```

Recommendation

The team is advised to review the current implementation to ensure it aligns with the intended vesting design. Ensuring consistency with expected behavior will help maintain user trust and prevent confusion.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	PPLtesttoken.sol#L101,109,110,111,112
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.

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.

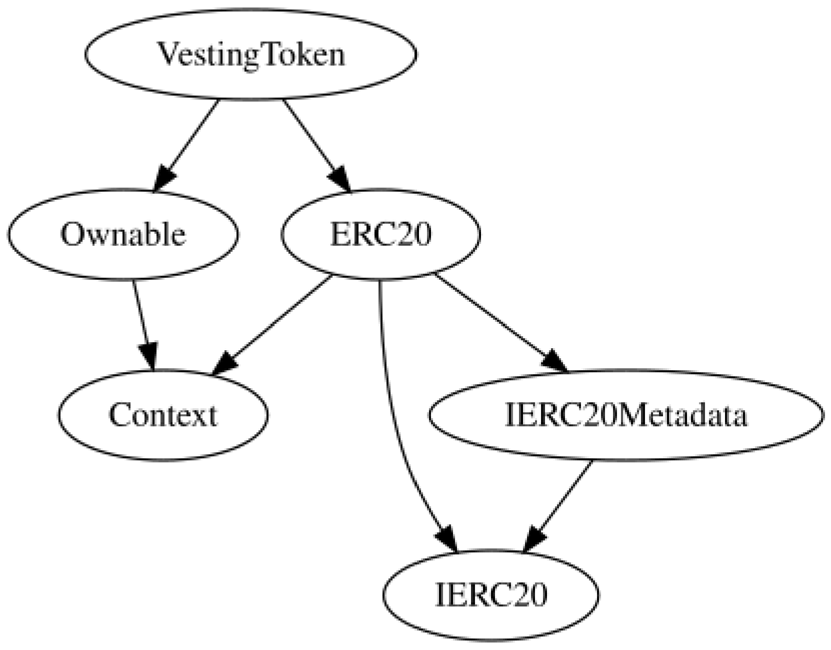
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
VestingToken	Implementation	ERC20, Ownable		
		Public	✓	ERC20 Ownable
	setGlobalStartTime	External	✓	onlyOwner
	setBeneficiaries	External	✓	onlyOwner
	setupFixedSchedules	External	✓	onlyOwner
	premintMarketing200B	External	✓	onlyOwner
	_addMonthlyMarketing	Internal	✓	
	_addYearlyCliff10	Internal	✓	
	_increaseAllocated	Internal	✓	
	mintTo	External	✓	onlyOwner
	claim	External	✓	-
	airdropClaimable	External	✓	onlyOwner
	getClaimableTokens	External		-
	_claimableMonthly	Internal	✓	
	_previewMonthly	Internal		
	_calcMonthly	Internal		
	_calcMonthlyView	Internal		
	_claimableYearlyCliff10	Internal	✓	
	_previewYearlyCliff10	Internal		

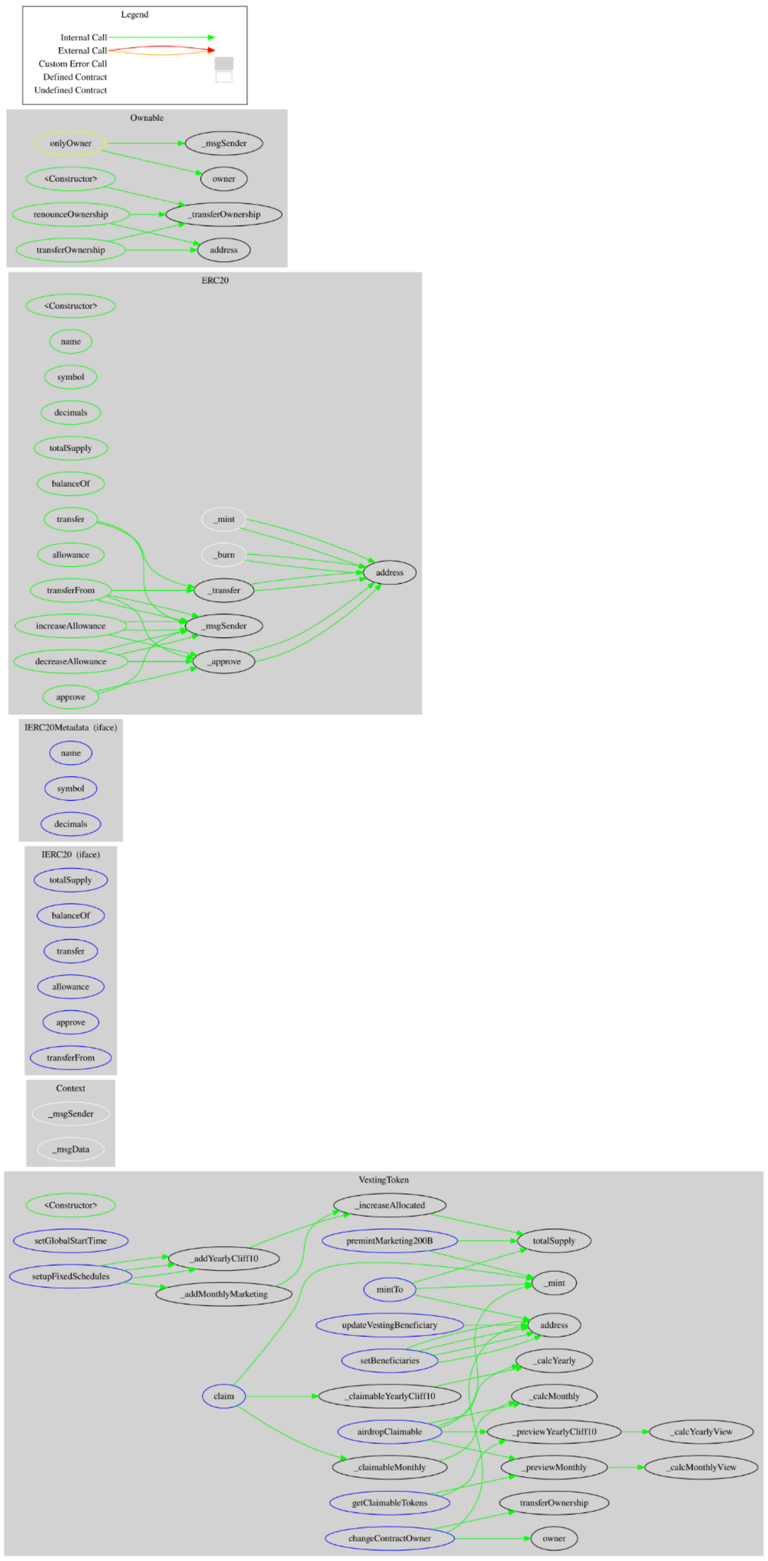
	_calcYearly	Internal		
	_calcYearlyView	Internal		
	updateVestingBeneficiary	External	✓	onlyOwner
	changeContractOwner	External	✓	onlyOwner
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-
ERC20	Implementation	Context, IERC20, IERC20Metadata		

		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Custom Error Call
Defined Contract
Undefined Contract
Ownable
onlyOwner
_msgSender
<Constructor>
owner
renounceOwnership
_transferOwnership
transferOwnership
address
ERC20
<Constructor>
name
symbol
decimals
totalSupply
balanceOf
transfer
_mint
allowance
_burn
address
transferFrom
_transfer
increaseAllowance
_msgSender
decreaseAllowance
_approve
approve
IERC20Metadata (iface)
name
symbol
decimals
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
Context
_msgSender
_msgData
VestingToken
<Constructor>
_increaseAllocated
setGlobalStartTime
premintMarketing200B
totalSupply
_addYearlyCliff10
setupFixedSchedules
_addMonthlyMarketing
mintTo
_mint
updateVestingBeneficiary
address
setBeneficiaries
_calcYearly
claim
_claimableYearlyCliff10
_calcMonthly
airdropClaimable
_previewYearlyCliff10
_calcYearlyView
_claimableMonthly
_previewMonthly
_calcMonthlyView
getClaimableTokens
transferOwnership
changeContractOwner
owner

Summary

PPLDAO contract implements a vesting mechanism. 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.

The Cyberscope team

https://www.cyberscope.io