Cyberscope

A TAC Security Company

Audit Report

EpowerX Token

November 2025

Network	BASE
Address	0xef5f5751cf3eca6cc3572768298b7783d33d60eb
Audited by	© cyberscope

Analysis

Critical Medium Minor / Informative Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Passed
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Passed
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
	UAR	Unexcluded Address Restrictions	Unresolved
	CR	Code Repetition	Unresolved
	CC	Commented Code	Unresolved
	HV	Hardcoded Values	Unresolved
	MFM	Marketing Fee Misallocation	Unresolved
	MEE	Missing Events Emission	Unresolved
	NWES	Nonconformity with ERC-20 Standard	Unresolved
	PLPI	Potential Liquidity Provision Inadequacy	Unresolved
	RRA	Redundant Repeated Approvals	Unresolved
	L04	Conformance to Solidity Naming Conventions	Unresolved

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

Contract Name	EPWX
Compiler Version	v0.8.20+commit.a1b79de6
Optimization	200 runs
Explorer	https://basescan.org/token/0xef5f5751cf3eca6cc3572768298b7783d33d60eb
Address	0xef5f5751cf3eca6cc3572768298b7783d33d60eb
Network	BASE
Symbol	EPWX
Decimals	9
Total Supply	500,000,000,000,000

Audit Updates

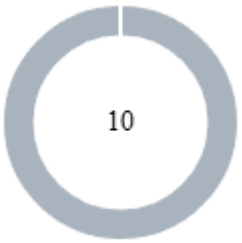
Initial Audit	24 Nov 2025

Source Files

Filename	SHA256
LiquidityFeeToken.sol	0399113b91d4d5626313f567953800a778bda42de2f865c8eb172a6dfb8fbbbb
libraries/SafeERC20.sol	861357a685bbc95e195f7a881c8e5c50315f85363adbe651d1fd520c5d125d57
libraries/OwnableUpgradeable.sol	d8bcdeeb6604ee23d9fcfdec88eac36b622e1c93faf9984de92ecb557a611102
libraries/Initializable.sol	985818cfc515a493a2d26e3e1729ea909a741779d754563529d7e6a342932b06

libraries/IERC20Permit.sol	9f4d30b273cd78086fc43063ee86b1be2ba134ff7e6cea66678d6096b314381e
libraries/IERC20.sol	0ccdd5bbccda715470386fd4f645f4647194738decbe8887049f7a32de162089
libraries/ContextUpgradeable.sol	8af7b0c2b1cfd29a10a99758e57aa79ca0cebee2facac9f36c5854da27b69920
libraries/AddressUpgradeable.sol	28d433b0603e3c7de2a4f063cc2885d43516dcf72f3ff55eab76870323a5ef3b
libraries/Address.sol	96f068984e0ba481106566b76686683b3c4479769dc2c5a175e431c9d4edff73
interfaces/IUniswapV2Router02.sol	805a1095db9b97ff1e7da3deb7bef3de4f9949478b921933d34ee28f0335cdf9
interfaces/IUniswapV2Factory.sol	2e46f7c98090f61b85447a5f87c779aac0a9983bfdecd78c2236b32585340247

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	10

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	10	0	0	0

UAR - Unexcluded Address Restrictions

Criticality	Minor / Informative
Location	LiquidityFeeToken.sol#L522
Status	Unresolved

Description

The contract incorporates operational restrictions on transactions, which can hinder seamless interaction with decentralized applications (dApps) such as launchpads, presales, lockers, or staking platforms. In scenarios where an external contract, such as a launchpad factory, needs to integrate with the contract, it should be exempt from the limitations to ensure uninterrupted service and functionality. Failure to provide such exemptions can block the successful process and operation of services reliant on this contract.

```shell
function _transfer(address from, address to, uint256
amount) private {
    ...
    // indicates if fee should be deducted from transfer
    bool takeFee = !inSwapAndLiquify;

    // don't take fee on liquidity removals
    if (from == uniswapV2Pair && to ==
address(uniswapV2Router))
        takeFee = false;

    // if any account belongs to _isExcludedFromFee
account then remove the fee
    if (
        owner() == from || owner() == to ||
        _marketingAddress == from || _marketingAddress ==
to ||
        address(this) == from || address(this) == to
    ) {
        takeFee = false;
```

```
        }
        _tokenTransfer(from, to, amount, takeFee);
    }
```

Recommendation

It is advisable to modify the contract by incorporating functionality that enables the exclusion of designated addresses from transactional restrictions. This enhancement will allow specific addresses, such as those associated with decentralized applications (dApps) and service platforms, to operate without being hindered by the standard constraints imposed on other users. Implementing this feature will ensure smoother integration and functionality with external systems, thereby expanding the contract's versatility and effectiveness in diverse operational environments.

CR - Code Repetition

Criticality	Minor / Informative
Location	LiquidityFeeToken.sol#L764
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
_transferFromExcluded(sender, recipient, amount);
  } else if (!_isExcluded[sender] &&
_isExcluded[recipient]) {
     _transferToExcluded(sender, recipient,
amount);
  } else if (!_isExcluded[sender] &&
!_isExcluded[recipient]) {
     _transferStandard(sender, recipient, amount);
  } else if (_isExcluded[sender] &&
_isExcluded[recipient]) {
```

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.

CC - Commented Code

Criticality	Minor / Informative
Location	LiquidityFeeToken.sol#L331,281,584,652,
Status	Unresolved

Description

The contract has several areas where commented code is left.Commented blocks reduce code readability and make it harder to see the actual active logic.

```
Shell
/*
    function excludeFromReward(address account) public
onlyOwner {
        require(_excluded.length <= 1000, "Cannot exclude
more accounts");
        // require(account !=
0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D, 'We can not
exclude Uniswap router.');
        require(!_isExcluded[account], "Account is already
excluded");
        if (_rOwned[account] > 0) {
            _tOwned[account] =
tokenFromReflection(_rOwned[account]);
        }
        _isExcluded[account] = true;
        _excluded.push(account);
    }

    function includeInReward(address account) external
onlyOwner {
        require(_isExcluded[account], "Account is already
included");
        for (uint256 i = 0; i < _excluded.length; i++) {
            if (_excluded[i] == account) {
                _excluded[i] = _excluded[_excluded.length -
```

```
1];
                _tOwned[account] = 0;
                _isExcluded[account] = false;
                _excluded.pop();
                break;
            }
        }
    }

*/
```

Recommendation

It is recommended to remove commended code to improve code readability.

HV - Hardcoded Values

Criticality	Minor / Informative
Location	LiquidityFeeToken.sol#L47,133,362,488
Status	Unresolved

Description

The contract contains multiple instances where numeric values are directly hardcoded into the code logic rather than being assigned to constant variables with descriptive names. Hardcoding such values can lead to several issues, including reduced code readability, increased risk of errors during updates or maintenance, and difficulty in consistently managing values throughout the contract. Hardcoded values can obscure the intent behind the numbers, making it challenging for developers to modify or for users to understand the contract effectively.

```shell
uint256 public constant MAX_FEE = 10_000 / 5;

numTokensSellToAddToLiquidity = totalSupply_ / (10 ** 3);

_amount >= (totalSupply() * 5) / (10_000),

return (_amount * _liquidityFee) / (10_000);
```

Recommendation

It is recommended to replace hardcoded numeric values with variables that have meaningful names. This practice improves code readability and maintainability by clearly indicating the purpose of each value, reducing the likelihood of errors during future modifications. Additionally, consider using constant variables which provide a reliable way to centralize and manage values, improving gas optimization throughout the contract.

MFM - Marketing Fee Misallocation

Criticality	Minor / Informative
Location	LiquidityFeeToken.sol#L465,546
Status	Unresolved

Description

When marketing fees are paid in the project's native token, the current fee-allocation logic creates a mismatch with the intended tokenomics. Although the configured marketing fee is correctly deducted and sent on each transfer, the swap-back process still treats the marketing fee as part of the accumulated fee pool—even though only liquidity fees remain in the contract. As a result, during each auto-swap cycle, the marketing wallet receives an additional share that was meant for liquidity. This causes the marketing wallet to get more than its intended percentage, while liquidity growth receives less.

Shell

```
function _takeMarketingFee(uint256 tMarketing)
private {
    address receiver = _marketingAddress;

    if (_marketingToken != address(0)) receiver =
address(this);

    if (tMarketing > 0) {
        uint256 currentRate = _getRate();
        uint256 rMarketing = tMarketing *
currentRate;
        _rOwned[receiver] = _rOwned[receiver] +
rMarketing;
        if (_isExcluded[receiver])
                _tOwned[receiver] =
_tOwned[receiver] + tMarketing;
         emit Transfer(_msgSender(), receiver,
tMarketing);
```

```
        }
    }

    uint256 _totalFees = _liquidityFee +
    _marketingFee;
```

Recommendation

It is advised to reconsider the overall fee-allocation logic to ensure that each fee category is distributed strictly according to its intended purpose.
This will help maintain full consistency with the documented tokenomics and preserve the desired balance between all fee components.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	LiquidityFeeToken.sol#L598,694
Status	Unresolved

Description

The contract performs actions and state mutations from external methods that do not result in the emission of events. Emitting events for significant actions is important as it allows external parties, such as wallets or dApps, to track and monitor the activity on the contract. Without these events, it may be difficult for external parties to accurately determine the `current` state of the contract.

```
Shell
function swapExactTokensForETH(uint256 tokenAmount)

function swapExactTokensForTokens(address tokenAddress,
uint256 tokenAmount)
```

Recommendation

It is recommended to include events in the code that are triggered each time a significant action is taking place within the contract. These events should include relevant details such as the user's address and the nature of the action taken. By doing so, the contract will be more transparent and easily auditable by external parties. It will also help prevent potential issues or disputes that may arise in the future.

NWES - Nonconformity with ERC-20 Standard

Criticality	Minor / Informative
Location	LiquidityFeeToken.sol#L525
Status	Unresolved

Description

The contract does not fully conform to the ERC20 Standard. Specifically, according to the standard, transfers of 0 values must be treated as normal transfers and fire the Transfer event. However, the contract implements a conditional check that prohibits transfers of 0 values. This discrepancy between the contract's implementation and the ERC20 standard may lead to inconsistencies and incompatibilities with other contracts.

```
Shell
function _transfer(address from, address to,
uint256 amount) private {
  ...
  require(amount > 0, "Transfer amount must be
greater than zero");
  ...
  }
```

Recommendation

The incorrect implementation of the ERC20 standard could potentially lead to problems when interacting with the contract, as other contracts or applications that expect the ERC20 interface may not behave as expected. The team is advised to review and revise the implementation of the transfer mechanism to ensure full compliance with the ERC20 standard. https://eips.ethereum.org/EIPS/eip-20.

PLPI - Potential Liquidity Provision Inadequacy

Criticality	Minor / Informative
Location	LiquidityFeeToken.sol#L598
Status	Unresolved

Description

The contract operates under the assumption that liquidity is consistently provided to the pair between the contract's token and the native currency. However, there is a possibility that liquidity is provided to a different pair. This inadequacy in liquidity provision in the main pair could expose the contract to risks. Specifically, during eligible transactions, where the contract attempts to swap tokens with the main pair, a failure may occur if liquidity has been added to a pair other than the primary one. Consequently, transactions triggering the swap functionality will result in a revert.

```shell
function swapExactTokensForETH(uint256 tokenAmount) private
returns (uint256[] memory amounts) {
        address[] memory path = new address[](2);
        path[0] = address(this);
        path[1] = uniswapV2Router.WETH();
        _approve(address(this), address(uniswapV2Router),
tokenAmount)
        amounts = uniswapV2Router.swapExactTokensForETH(
            tokenAmount,
            0,
            path,
            address(this),
            block.timestamp
        );

    }
```

Recommendation

The team is advised to implement a runtime mechanism to check if the pair has adequate liquidity provisions. This feature allows the contract to omit token swaps if the pair does not have adequate liquidity provisions, significantly minimizing the risk of potential failures.

Furthermore, the team could ensure the contract has the capability to switch its active pair in case liquidity is added to another pair.

Additionally, the contract could be designed to tolerate potential reverts from the swap functionality, especially when it is a part of the main transfer flow. This can be achieved by executing the contract's token swaps in a non-reversible manner, thereby ensuring a more resilient and predictable operation.

RRA - Redundant Repeated Approvals

Criticality	Minor / Informative
Location	LiquidityFeeToken.sol#L598,634,740
Status	Unresolved

Description

The contract is designed to `approve` token transfers during the contract's operation by calling the _approve function before specific operations. This approach results in additional gas costs since the approval process is repeated for every operation execution, leading to inefficiencies and increased transaction expenses.

```shell
function swapExactTokensForETH(uint256 tokenAmount)
    .....
       _approve(address(this), address(uniswapV2Router),
tokenAmount);

function addLiquidity(uint256 tokenAmount, uint256
ethAmount)
    .....
      _approve(address(this), address(uniswapV2Router),
tokenAmount);

function swapExactTokensForTokens(address tokenAddress,
uint256 tokenAmount)
    .....
       _approve(address(this), address(uniswapV2Router),
tokenAmount);
```

Recommendation

Since the approved address is a trusted third-party source, it is recommended to optimize the contract by approving the maximum amount of tokens once in the initial set of the variable, rather than before each operation. This change will reduce the overall gas consumption and improve the efficiency of the contract.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	LiquidityFeeToken.sol#L66,69,72,73,78,361,482,487,493
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.

```
Shell
public _taxFee;

public _liquidityFee;
...
public _marketingAddress;

(uint256 _amount) external
(uint256 _amount) private
```

```
...
```

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**
LiquidityFeeToken	Implementation	IERC20, Initializable, OwnableUpgradeable		
		Public	✓	-
	initialize	Public	✓	initializer
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	isExcludedFromReward	Public		-
	totalFees	Public		-
	deliver	Public	✓	-
	reflectionFromToken	Public		-

	tokenFromReflection	Public		-
	_transferBothExcluded	Private	✓	
	setSwapBackSettings	External	✓	onlyOwner
		External	Payable	-
	_reflectFee	Private	✓	
	_getValues	Private		
	_getTValues	Private		
	_getRValues	Private		
	_getRate	Private		
	_getCurrentSupply	Private		
	_takeLiquidity	Private	✓	
	_takeMarketingFee	Private	✓	
	calculateTaxFee	Private		
	calculateLiquidityFee	Private		
	calculateMarketingFee	Private		
	removeAllFee	Private	✓	
	restoreAllFee	Private	✓	
	_approve	Private	✓	
	_transfer	Private	✓	
	swapExactTokensForETH	Private	✓	
	swapExactTokensForTokens	Private	✓	
	manageFees	Private	✓	
	addLiquidity	Private	✓	
	_tokenTransfer	Private	✓	
	_transferStandard	Private	✓	

	_transferToExcluded	Private	✓	
	_transferFromExcluded	Private	✓	
SafeERC20	Library			
	safeTransfer	Internal	✓	
	safeTransferFrom	Internal	✓	
	safeApprove	Internal	✓	
	safeIncreaseAllowance	Internal	✓	
	safeDecreaseAllowance	Internal	✓	
	forceApprove	Internal	✓	
	safePermit	Internal	✓	
	_callOptionalReturn	Private	✓	
	_callOptionalReturnBool	Private	✓	
OwnableUpgradeable	Implementation	Initializable, ContextUpgradeable		
	__Ownable_init	Internal	✓	onlyInitializing
	__Ownable_init_unchained	Internal	✓	onlyInitializing
	owner	Public		-
	_checkOwner	Internal		
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
Initializable	Implementation			

	_disableInitializers	Internal	✓	
	_getInitializedVersion	Internal		
	_isInitializing	Internal		
IERC20Permit	Interface			
	permit	External	✓	-
	nonces	External		-
	DOMAIN_SEPARATOR	External		-
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
ContextUpgradeable	Implementation	Initializable		
	__Context_init	Internal	✓	onlyInitializing
	__Context_init_unchained	Internal	✓	onlyInitializing
	_msgSender	Internal		
	_msgData	Internal		
	_contextSuffixLength	Internal		

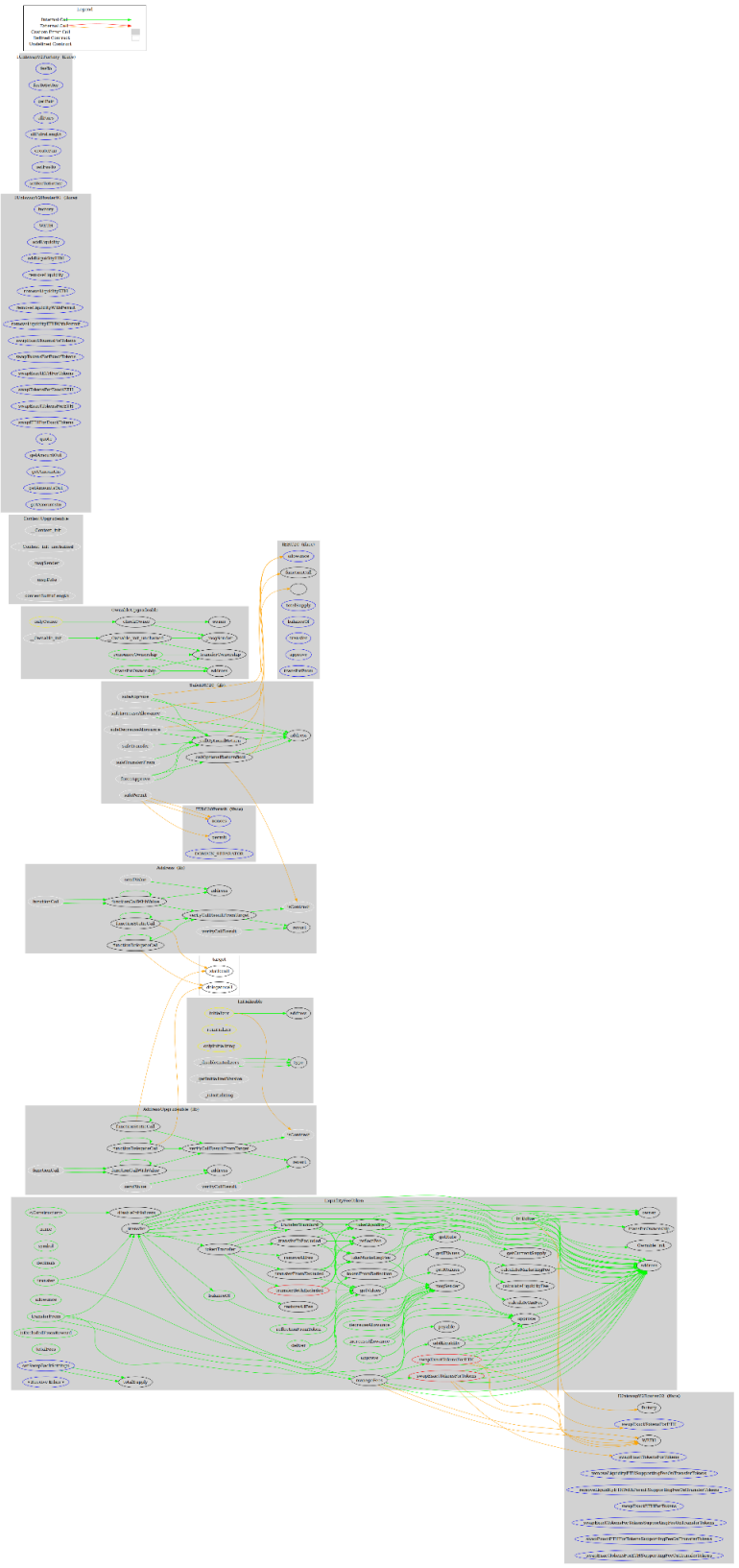
AddressUpgradeable	Library			
	isContract	Internal		
	sendValue	Internal	✓	
	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionStaticCall	Internal		
	functionStaticCall	Internal		
	functionDelegateCall	Internal	✓	
	functionDelegateCall	Internal	✓	
	verifyCallResultFromTarget	Internal		
	verifyCallResult	Internal		
	_revert	Private		
Address	Library			
	isContract	Internal		
	sendValue	Internal	✓	
	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionStaticCall	Internal		
	functionStaticCall	Internal		

	functionDelegateCall	Internal	✓	
	functionDelegateCall	Internal	✓	
	verifyCallResultFromTarget	Internal		
	verifyCallResult	Internal		
	_revert	Private		
IUniswapV2Router01	Interface			
	factory	External		-
	WETH	External		-
	addLiquidity	External	✓	-
	addLiquidityETH	External	Payable	-
	removeLiquidity	External	✓	-
	removeLiquidityETH	External	✓	-
	removeLiquidityWithPermit	External	✓	-
	removeLiquidityETHWithPermit	External	✓	-
	swapExactTokensForTokens	External	✓	-
	swapTokensForExactTokens	External	✓	-
	swapExactETHForTokens	External	Payable	-
	swapTokensForExactETH	External	✓	-
	swapExactTokensForETH	External	✓	-
	swapETHForExactTokens	External	Payable	-
	quote	External		-
	getAmountOut	External		-
	getAmountIn	External		-

	getAmountsOut	External		-
	getAmountsIn	External		-
IUniswapV2Router02	Interface	IUniswapV2Router01		
	removeLiquidityETHSupportingFeeOnTransferTokens	External	✓	-
	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	✓	-
	swapExactTokensForTokens	External	✓	-
	swapExactETHForTokens	External	Payable	-
	swapExactTokensForETH	External	✓	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
IUniswapV2Factory	Interface			
	feeTo	External		-
	feeToSetter	External		-
	getPair	External		-
	allPairs	External		-
	allPairsLength	External		-
	createPair	External	✓	-
	setFeeTo	External	✓	-
	setFeeToSetter	External	✓	-

Inheritance Graph

Flow Graph



Summary

ePowerX On Base contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. ePowerX On Base is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler error or critical issues.

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 TAC 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