Cyberscope

Audit Report

Quidax Token

February 2025

Network	BSC
Address	0x08BC237aA00F275b758542c9F5c125b568bc390A
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

Contract Name	BridgeMintableTokenV2
Compiler Version	v0.6.12+commit.27d51765
Optimization	200 runs
Proxy Explorer	https://bscscan.com/address/0x08BC237aA00F275b758542c9F5c125b568bc390A
Implementation Explorer	https://bscscan.com/address/0x877d233c59741f36154bb7fbaea853e317fbdcd6
Proxy Address	0x08BC237aA00F275b758542c9F5c125b568bc390A
Implementation Address	0x877d233c59741f36154bb7fbaea853e317fbdcd6
Network	BSC
Symbol	QDX
Decimals	18
Total Supply	750124102546702810775565 19

Audit Updates

Initial Audit	18 Feb 2025

Source Files

Filename	SHA256
BridgeMintableTokenV2.sol	27e681296297f5f8fe0c5080b87e26c8f5334651e9e58165f554a9259741c689

Overview

BridgeMintableTokenV2

The `BridgeMintableTokenV2` is an upgradeable, pausable, and mintable ERC-20 token contract designed for cross-chain bridging. It allows authorized entities, such as the `chainportBridge`, to `mint` tokens when assets are bridged from another blockchain. The contract integrates blacklist enforcement through a `BlacklistRegistry`, preventing transactions involving blacklisted addresses. Additionally, governance is managed by the `chainportCongress`, which has the ability to pause, unblock, and update bridge settings. The contract also includes security mechanisms such as blocking malicious addresses, burning blacklisted funds, and recovering mistakenly sent tokens.

Initialization

The contract is initialized with the token's name, symbol, decimals, governance, and the `BlacklistRegistry`. It also sets the deployer as the initial bridge address. This function is called only once during deployment.

Minting Tokens

The `mint` function allows authorized entities to create new tokens and send them to a specified address. This function can only be executed when the contract is not paused, and it ensures that neither the sender nor the recipient is blacklisted before proceeding. Once the minting process is successful, the contract emits a Mint event to record the transaction.

Burning Tokens

The `burn` function enables token holders to permanently remove a specific amount of their tokens from circulation. This operation reduces the total supply and updates the sender’s balance. Before executing, the function checks whether the caller is blacklisted, ensuring compliance with security restrictions.

The `burnFrom` function allows an approved spender to burn tokens on behalf of another account. The function first verifies that the account has granted a sufficient allowance to the caller. It then deducts the burned amount from the allowance and updates the account's token balance accordingly.

The `destroyBlackFunds` function allows `chainportCongress` to permanently remove tokens from an address. Once executed, the specified amount of tokens is burned, and an event is emitted to log the action.

Pausing and Resuming Token Transfers

The `pause` function temporarily freezes all token transfers within the contract. Only authorized entities can trigger this function. While paused, users cannot transfer, mint, or burn tokens, providing an emergency mechanism to prevent suspicious activity.

The `unpause` function lifts the freeze on token transfers, allowing normal operations to resume. This function is exclusively callable by `chainportCongress`, ensuring that only governance-level approval can reinstate token transfers after a pause.

Blocking and Unblocking an Address

The `blockAddress` function restricts a specific address from transferring tokens. It is intended as a security measure against malicious actors or compromised accounts. Only authorized entities can block an address, and the contract emits an `AddressBlocked` event upon execution.

The `unblockAddress` function restores the ability of a previously blocked address to transfer tokens. Only `chainportCongress` can perform this action, ensuring careful review before unblocking. Upon execution, the contract emits an `AddressUnblocked` event.

Updating the Bridge Contract

The `setSideBridgeContract` function updates the designated bridge contract. This is useful in cases where the bridge infrastructure changes or requires an upgrade. The function can only be executed by `chainportCongress` and ensures the new address is valid before assignment.

Updating the Governance

The `setChainportCongress` function assigns a new governance authority. This function is critical in ensuring decentralized control and decision-making over the contract. Only the current `chainportCongress` can invoke this function to prevent unauthorized governance changes.

Recovering Stuck Tokens

The `withdrawTokenIfStuck` function provides a mechanism for retrieving accidentally sent ERC-20 tokens that may be stuck in the contract. The function transfers the stuck tokens to a specified beneficiary and can only be executed by `chainportCongress`. This ensures safe recovery while preventing unauthorized access.

Roles

ChainportCongress

The `chainportCongress` address is the highest authority in the contract, responsible for critical governance actions and security measures.

- `pause()`
- `unpause()`
- `setSideBridgeContract(address _chainportBridge)`
- `setChainportCongress(address _chainportCongress)`
- `unblockAddress(address addressToUnblock)`
- `destroyBlackFunds(address maliciousAddress, uint256 amount)`
- `withdrawTokenIfStuck(address token, address beneficiary)`

ChainportBridge

The `chainportBridge` address is responsible for minting tokens when assets are bridged from another blockchain.

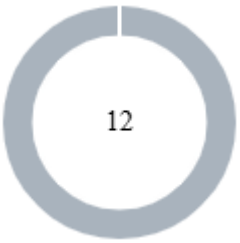
- `mint(address _to, uint256 _amount)`
- `pause()`
- `blockAddress(address addressToBlock)`

Token holders

Any address holding `BridgeMintableTokenV2` tokens can interact with standard ERC-20 functions, including transfers, approvals, and burning. Token holders are also subject to blacklist restrictions.

- `transfer(address recipient, uint256 amount)`
- `approve(address spender, uint256 amount)`
- `transferFrom(address sender, address recipient, uint256 amount)`
- `burn(uint256 amount)`
- `burnFrom(address account, uint256 amount)`

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
	CCR	Contract Centralization Risk	Unresolved
	BC	Blacklists Addresses	Unresolved
	BT	Burns Tokens	Unresolved
	MT	Mints Tokens	Unresolved
	ST	Stops Transactions	Unresolved
	UTPD	Unverified Third Party Dependencies	Unresolved
	L09	Dead Code Elimination	Unresolved
	L16	Validate Variable Setters	Unresolved
	L17	Usage of Solidity Assembly	Unresolved
	L18	Multiple Pragma Directives	Unresolved
	L19	Stable Compiler Version	Unresolved
	L20	Succeeded Transfer Check	Unresolved

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	BridgeMintableTokenV2.sol#L1155,1215,1222,1231,1241,1251,1262,1274,1285
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 mint(address _to, uint256 _amount)
public
whenNotPaused
onlyClassifiedAuthority
onlyNonBlacklistedAddress(_to, address(0x0))
{}
function pause() external onlyClassifiedAuthority {}
function unpause() external onlyChainportCongress {}
function blockAddress(address addressToBlock) external
onlyClassifiedAuthority {}
function unblockAddress(address addressToUnblock) external
onlyChainportCongress {}
function setSideBridgeContract(address _chainportBridge)
external onlyChainportCongress {}
function setChainportCongress(address _chainportCongress)
external onlyChainportCongress {}
function destroyBlackFunds(address maliciousAddress, uint256
amount) external onlyChainportCongress {}
function withdrawTokenIfStuck(address token, address
beneficiary) external onlyChainportCongress {}
```

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.

BC - Blacklists Addresses

Criticality	Minor / Informative
Location	BridgeMintableTokenV2.sol#L1095,1231
Status	Unresolved

Description

Classified Authorities can stop addresses from transactions. They may take advantage of it by calling the `blockAddress` function.

```
function blockAddress(address addressToBlock) external
onlyClassifiedAuthority {
    isBlocked[addressToBlock] = true;
    emit AddressBlocked(addressToBlock);
}

function _beforeTokenTransfer(address from, address to, uint256
amount) internal override {
    //...
    require(!isBlocked[from], "ChainportToken: Sender address
blocked.");
    //...
}
```

Additionally `blacklistRegistry`, which is an external contract, is able to block addresses from transactions. This is also mentioned in `UTPD` section.

```
modifier onlyNonBlacklistedAddress(address user1, address
user2) {
    _onlyNonBlacklistedAddress(user1, user2);
    _;
}

function _onlyNonBlacklistedAddress(address user1, address
user2) internal view {
    //...
    (, bool isBlacklisted) =
blacklistRegistry.isAnyBlacklisted(addresses);
    require(
        !isBlacklisted,
        "ONLY_NON_BLACKLISTED"
    );
}

function _transfer(address sender, address recipient, uint256
amount) internal override onlyNonBlacklistedAddress(sender,
recipient) {
    super._transfer(sender, recipient, amount);
}
```

Recommendation

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

The team could:

- 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.

Additionally the team should verify that the external contract `blacklistRegistry` by following the recommendations of the `UTPD`.

BT - Burns Tokens

Criticality	Minor / Informative
Location	BridgeMintableTokenV2.sol#L1274
Status	Unresolved

Description

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

```
function destroyBlackFunds(address maliciousAddress, uint256
amount) external onlyChainportCongress {
    _burn(maliciousAddress, amount);
    emit BlackFundsDestroyed(maliciousAddress, amount);
}
```

Recommendation

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

The team could:

- 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.

MT - Mints Tokens

Criticality	Minor / Informative
Location	BridgeMintableTokenV2.sol#L1155
Status	Unresolved

Description

Classified Authorities are able to mint tokens. They 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)
public
whenNotPaused
onlyClassifiedAuthority
onlyNonBlacklistedAddress(_to, address(0x0))
{
    _mint(_to, _amount);
    emit Mint(_to, _amount);
}
```

Recommendation

The team should carefully manage the private keys of the Classified Authorities accounts. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions.

The team could:

- 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.

ST - Stops Transactions

Criticality	Minor / Informative
Location	BridgeMintableTokenV2.sol#L1215
Status	Unresolved

Description

Classified Authorities are able to stop the sales for all users excluding the `chainportCongress` address. They may take advantage of it by calling the `pause` function. As a result, the contract may operate as a honeypot.

```
function paused() public view virtual returns (bool) {
    return _paused;
}

function pause() external onlyClassifiedAuthority {
    _pause();
}

function _beforeTokenTransfer(address from, address to, uint256
amount) internal override {
    super._beforeTokenTransfer(from, to, amount);

    // Exclude chainportCongress'
    if(from != chainportCongress) {
        require(!paused(), "ChainportToken: Token paused.");
        require(!isBlocked[from], "ChainportToken: Sender
address blocked.");
    }
}
```

Recommendation

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

The team could:

- 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.

UTPD - Unverified Third Party Dependencies

Criticality	Minor / Informative
Location	BridgeMintableTokenV2.sol#L1124
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.

```
function initialize(
    **params**,
    IBlacklistRegistry blacklistRegistry_
)
public
initializer
{
    //...
    blacklistRegistry = blacklistRegistry_;
}
```

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.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	BridgeMintableTokenV2.sol#L540
Status	Unresolved

Description

In Solidity, dead code is code that is written in the contract, but is never executed or reached during normal contract execution. Dead code can occur for a variety of reasons, such as:

- Conditional statements that are always false.
- Functions that are never called.
- Unreachable code (e.g., code that follows a return statement).

Dead code can make a contract more difficult to understand and maintain, and can also increase the size of the contract and the cost of deploying and interacting with it.

```
function __Context_init() internal initializer {
        __Context_init_unchained();
    }
```

Recommendation

To avoid creating dead code, it's important to carefully consider the logic and flow of the contract and to remove any code that is not needed or that is never executed. This can help improve the clarity and efficiency of the contract.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	BridgeMintableTokenV2.sol#L1122,1124
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.

```
chainportCongress = chainportCongress_
blacklistRegistry = blacklistRegistry_;
```

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.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	BridgeMintableTokenV2.sol#L36,159
Status	Unresolved

Description

Using assembly can be useful for optimizing code, but it can also be error-prone. It's important to carefully test and debug assembly code to ensure that it is correct and does not contain any errors.

Some common types of errors that can occur when using assembly in Solidity include Syntax, Type, Out-of-bounds, Stack, and Revert.

```
assembly { size := extcodesize(account) }

assembly {
    let returndata_size := mload(returndata)
    revert(add(32, returndata), returndata_size)
}
```

Recommendation

It is recommended to use assembly sparingly and only when necessary, as it can be difficult to read and understand compared to Solidity code.

L18 - Multiple Pragma Directives

Criticality	Minor / Informative
Location	BridgeMintableTokenV2.sol#L6,175,230,447,527,561,875,973,1052,1062
Status	Unresolved

Description

If the contract includes multiple conflicting pragma directives, it may produce unexpected errors. To avoid this, it's important to include the correct pragma directive at the top of the contract and to ensure that it is the only pragma directive included in the contract.

```
pragma solidity >=0.6.2 <0.8.0;
pragma solidity >=0.4.24 <0.8.0;
pragma solidity >=0.6.0 <0.8.0;
pragma solidity ^0.6.12;
pragma solidity 0.6.12;
```

Recommendation

It is important to include only one pragma directive at the top of the contract and to ensure that it accurately reflects the version of Solidity that the contract is written in.

By including all required compiler options and flags in a single pragma directive, the potential conflicts could be avoided and ensure that the contract can be compiled correctly.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	BridgeMintableTokenV2.sol#L1052
Status	Unresolved

Description

The `^` symbol indicates that any version of Solidity that is compatible with the specified version (i.e., any version that is a higher minor or patch version) can be used to compile the contract. The version lock is a mechanism that allows the author to specify a minimum version of the Solidity compiler that must be used to compile the contract code. This is useful because it ensures that the contract will be compiled using a version of the compiler that is known to be compatible with the code.

```
pragma solidity ^0.6.12;
```

Recommendation

The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest version that provides all the required functionality for the codebase. As a result, the project will be compiled in a well-tested LTS (Long Term Support) environment.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	BridgeMintableTokenV2.sol#L1287
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.

```
IERC20(token).transfer(beneficiary, amount)
```

Recommendation

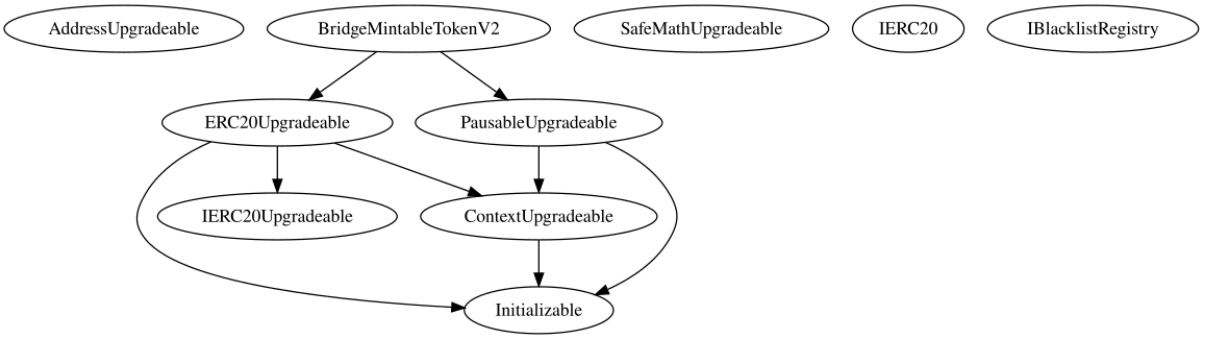
The contract should check if the result of the transfer methods is successful. The team is advised to check the SafeERC20 library from the Openzeppelin library.

Functions Analysis

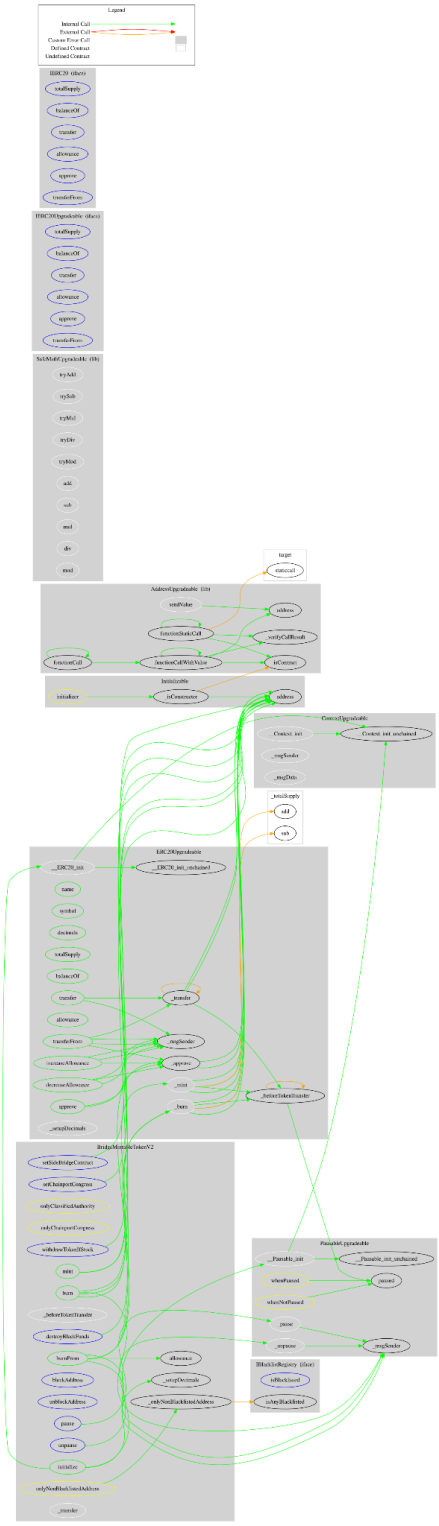
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IBlacklistRegistry	Interface			
	isBlacklisted	External		-
	isAnyBlacklisted	External		-
BridgeMintableTokenV2	Implementation	ERC20Upgradeable, PausableUpgradeable		
	_onlyNonBlacklistedAddress	Internal		
	initialize	Public	✓	initializer
	mint	Public	✓	whenNotPaused onlyClassifiedAuthority onlyNonBlacklistedAddress
	burn	Public	✓	onlyNonBlacklistedAddress
	burnFrom	Public	✓	onlyNonBlacklistedAddress
	_beforeTokenTransfer	Internal	✓	
	pause	External	✓	onlyClassifiedAuthority
	unpause	External	✓	onlyChainportCongress
	blockAddress	External	✓	onlyClassifiedAuthority
	unblockAddress	External	✓	onlyChainportCongress

	setSideBridgeContract	External	✓	onlyChainportCongress
	setChainportCongress	External	✓	onlyChainportCongress
	destroyBlackFunds	External	✓	onlyChainportCongress
	withdrawTokenIfStuck	External	✓	onlyChainportCongress
	_transfer	Internal	✓	onlyNonBlacklistedAddress

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Custom Error Call
Defined Contract
Undefined Contract
IBlackListRegistry
isBlocklisted
isAnyBlacklisted
Pausable
_pause
_unpause
whenPaused
whenNotPaused
ERC20Upgradeable
_transfer
_mint
_burn
_approve
_spendAllowance
BridgeMintableTokenV2
mint
burn
_beforeTokenTransfer
_msgSender
_msgData

Summary

Quidax Token contract implements a token and bridge mechanism. This audit investigates security issues, business logic concerns and potential improvements. There are some functions that can be abused by the authorities, like blacklisting addresses and stopping transactions. A multi-wallet signing pattern will provide security against potential hacks.

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

cyberscope.io