Cyberscope

Audit Report

CrazyyFrogCoin

December 2024

Network	ETH
Address	0xC76d496182b8b01570A6B96cc119bD9D32A336Bf
Audited by	© cyberscope

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Unresolved
●	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
●	UTPD	Unverified Third Party Dependencies	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	MU	Modifiers Usage	Unresolved
●	PTRP	Potential Transfer Revert Propagation	Unresolved
●	RHF	Redundant Hook Functionality	Unresolved
●	UFU	Unnecessary Flag Usage	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L15	Local Scope Variable Shadowing	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved
●	L18	Multiple Pragma Directives	Unresolved
●	L19	Stable Compiler Version	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	CrazyyFrogCoin
Compiler Version	v0.8.28+commit.7893614a
Optimization	200 runs
Explorer	https://etherscan.io/address/0xc76d496182b8b01570a6b96cc119bd9d32a336bf
Address	0xc76d496182b8b01570a6b96cc119bd9d32a336bf
Network	ETH
Symbol	CFC
Decimals	18
Total Supply	111,111,111,111
Badge Eligibility	Must Fix Criticals

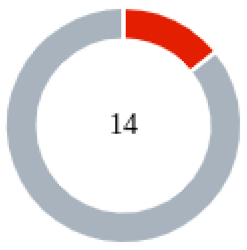
Audit Updates

Initial Audit	23 Dec 2024

Source Files

Filename	SHA256
CrazyyFrogCoin.sol	1450c57e6206b9d9bb2718bd15e033e34c49539f6b5c66289261a083c0ca61bc

Findings Breakdown



🟥	Critical	2
🟧	Medium	0
⬜	Minor / Informative	12

Severity		Unresolved	Acknowledged	Resolved	Other
🟥	Critical	2	0	0	0
🟧	Medium	0	0	0	0
⬜	Minor / Informative	12	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	CrazyyFrogCoin.sol#L3978
Status	Unresolved

Description

The contract owner has the authority to stop transactions, as described in detail in sections `UTPD` and `PTRP`. As a result, the contract might operate as a honeypot.

Recommendation

The team is advised to follow the recommendations outlined in the `UTPD` and `PTRP` findings and implement the necessary steps to mitigate the identified risks, ensuring that the contract does not operate as a honeypot. Renouncing ownership will effectively eliminate the threats, but it is non-reversible.

UTPD - Unverified Third Party Dependencies

Criticality	Critical
Location	CrazyyFrogCoin.sol#L3950,3978
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 setRaffleAddress(address _burnRaffleAddress) public
onlyOwner {
        // Owner can set or update the raffle address after
deployment
        require(_burnRaffleAddress != address(0), "Invalid
raffle address");
        burnRaffleAddress = _burnRaffleAddress;
    }
    ...

if (hookEnabled && to == burnRaffleAddress && value >=
minBurnAmount && burnRaffleAddress != address(0)) {
    IBurnRaffle raffle = IBurnRaffle(burnRaffleAddress);
    (bool success, ) = address(raffle).call(

abi.encodeWithSignature("registerParticipant(address,uint256)",
from, value)
    );
    ...
```

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.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	CrazyyFrogCoin.sol#L3950
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.

```
function setRaffleAddress(address _burnRaffleAddress) public
onlyOwner {
    // Owner can set or update the raffle address after
deployment
    require(_burnRaffleAddress != address(0), "Invalid raffle
address");
    burnRaffleAddress = _burnRaffleAddress;
}
```

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.

MU - Modifiers Usage

Criticality	Minor / Informative
Location	CrazyyFrogCoin.sol#L3957,3963
Status	Unresolved

Description

The contract is using repetitive statements on some methods to validate some preconditions. In Solidity, the form of preconditions is usually represented by the modifiers. Modifiers allow you to define a piece of code that can be reused across multiple functions within a contract. This can be particularly useful when you have several functions that require the same checks to be performed before executing the logic within the function.

```
require(_minBurnAmount > 0, "Minimum burn amount must be
greater than zero");
```

Recommendation

The team is advised to use modifiers since it is a useful tool for reducing code duplication and improving the readability of smart contracts. By using modifiers to perform these checks, it reduces the amount of code that is needed to write, which can make the smart contract more efficient and easier to maintain.

PTRP - Potential Transfer Revert Propagation

Criticality	Minor / Informative
Location	CrazyyFrogCoin.sol#L3978
Status	Unresolved

Description

The contract interacts with a `burnRaffleAddress` as part of the transfer flow. This address can either be a wallet address or a contract. If the address belongs to a contract then it may revert from incoming transactions. As a result, the error will propagate to the token's contract and revert the transfer.

```
if (hookEnabled && to == burnRaffleAddress && value >=
minBurnAmount && burnRaffleAddress != address(0)) {
    IBurnRaffle raffle = IBurnRaffle(burnRaffleAddress);
    (bool success, ) = address(raffle).call(

abi.encodeWithSignature("registerParticipant(address,uint256)",
from, value)
    );
    if (!success) {
        revert("Raffle participation failed");
    }
}
```

Recommendation

The contract should tolerate the potential revert from the underlying contracts when the interaction is part of the main transfer flow. This could be achieved by not allowing set contract addresses or by sending the transaction in a non-revertable way.

RHF - Redundant Hook Functionality

Criticality	Minor / Informative
Location	CrazyyFrogCoin.sol#L3969
Status	Unresolved

Description

The contract is designed with an `enableHook` function that aims to re-enable the `hookEnabled` state. However, `hookEnabled` is already set to `true` by default, and once it is set to `false`, it cannot be reverted back to `true` due to the `require` check for `hookDisabledPermanently`. This renders the `enableHook` function redundant. As it stands, this function does not serve any practical purpose within the current form of the contract, and its presence may lead to confusion for developers or users interacting with the contract.

```
    bool public hookEnabled = true;

     function enableHook() public onlyOwner {
         require(!hookDisabledPermanently, "Hook permanently
disabled");
         hookEnabled = true;
         emit HookEnabled();
     }
```

Recommendation

It is recommended to evaluate the intended functionality of the `enableHook` function and, if unnecessary, remove it to streamline the code and reduce potential misunderstanding or misuse. If re-enabling the hook is required in the future, consider revising the logic to support this functionality.

UFU - Unnecessary Flag Usage

Criticality	Minor / Informative
Location	CrazyyFrogCoin.sol#L3956,3978
Status	Unresolved

Description

The contract is designed with a `hookEnabled` variable intended as a flag to control the execution of specific functionality. However, the custom logic is inherently triggered only when `minBurnAmount` is greater than zero. Since the `setMinBurnAmount` function enforces a requirement that `minBurnAmount` must always be greater than zero, the `minBurnAmount` variable effectively acts as a self-sufficient switch for this functionality. As a result, the `hookEnabled` variable becomes redundant, adding unnecessary complexity to the contract without serving any additional purpose, since the custom logic could be executed if the `minBurnAmount` is greater than zero.

```
    function setMinBurnAmount(uint256 _minBurnAmount) public
onlyOwner {
        require(_minBurnAmount > 0, "Minimum burn amount must
be greater than zero");
        minBurnAmount = _minBurnAmount;
        emit MinBurnAmountUpdated(_minBurnAmount);
    }

    function _update(address from, address to, uint256 value)
internal virtual override {
        ...

        if (hookEnabled && to == burnRaffleAddress && value >=
minBurnAmount && burnRaffleAddress != address(0)) {
        ...
        }
```

Recommendation

It is recommended to remove the `hookEnabled` variable and update the contract logic to rely solely on `minBurnAmount` being greater than zero to trigger the intended functionality. Additionally, the `setMinBurnAmount` function should allow setting the value to zero as an explicit indicator to disable the custom functionality. This optimisation will streamline the contract's logic, reduce storage usage, and eliminate the need for an additional flag, improving the overall efficiency and maintainability of the code.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	CrazyyFrogCoin.sol#L752,3571,3582,3716,3950,3956
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.

```
function DOMAIN_SEPARATOR() external view returns (bytes32);
function _EIP712Name() internal view returns (string memory) {
function _EIP712Version() internal view returns (string memory)
{
function _EIP712Version() internal view returns (string memory)
{
...
```

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.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	CrazyyFrogCoin.sol#L138,812,847,858,874,989,1049,1066,1083,1100,1117,1134,1151,1168,1185,1202,1219,1236,1253,1270,1287,1304,1321,1338,1355,1372,1389,1406,1423,1440,1457,1474,1491,1508,1525,1542,1559,1573,1591,1609,1627,1645,1663,1681,1699,1717,1735,1753,1771,1789,1807,1825,1843,1861,1879,1897,1915,1933,1951,1969,1987,2005,2023,2041,2059,2077,2095,2113,2131,2145,2156,2246,2258,2265,2273,2284,2307,2391,2406,2463,2482,2500,2529,2540,2567,2583,2692,2703,2744,2755,2793,2806,2835,2845,2869,2881,2888,2896,2905,2944,2967,2974,2983,3002,3010,3033,3068,3086,3100,3191,3200,3209,3218,3227,3236,3254,3263,3385,3628,3880,3888,3898
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 _contextSuffixLength() internal view virtual returns
(uint256) {
        return 0;
    }

...
```

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.

L15 - Local Scope Variable Shadowing

Criticality	Minor / Informative
Location	CrazyyFrogCoin.sol#L3675
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.

```
constructor(string memory name) EIP712(name, "1") {}
```

Recommendation

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

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	CrazyyFrogCoin.sol#L3946
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.

```
burnRaffleAddress = _burnRaffleAddress;
minBurnAmount = _minBurnAmount;
```

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	CrazyyFrogCoin.sol#L822,990,2157,2314,2502,2552,2949,3015,3069,3114,3192,3201,3210,3219,3228,3237,3246,3255,3264,3337
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 ("memory-safe") {
                r := mload(add(signature, 0x20))
                s := mload(add(signature, 0x40))
                v := byte(0, mload(add(signature, 0x60)))
            }

...
            mstore(0x00, 0x4e487b71)
            mstore(0x20, code)
            revert(0x1c, 0x24)
        }
    }
}


...
```

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	CrazyyFrogCoin.sol#L7,89,117,147,312,626,667,760,943,1004,2168,2855,2925,3043,3130,3275,3399,3430,3591,3641,3726,3828,3906
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 ^0.8.20;

...
```

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	CrazyyFrogCoin.sol#L7,89,117,147,312,626,667,760,943,1004,2168,2855,2925,3043,3130,3275,3399,3430,3591,3641,3726,3828,3906
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 ^0.8.20;
...
```

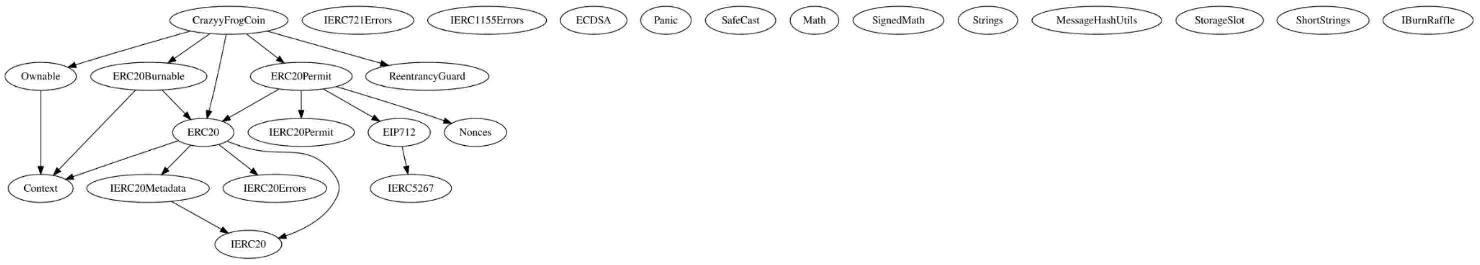
Recommendation

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.

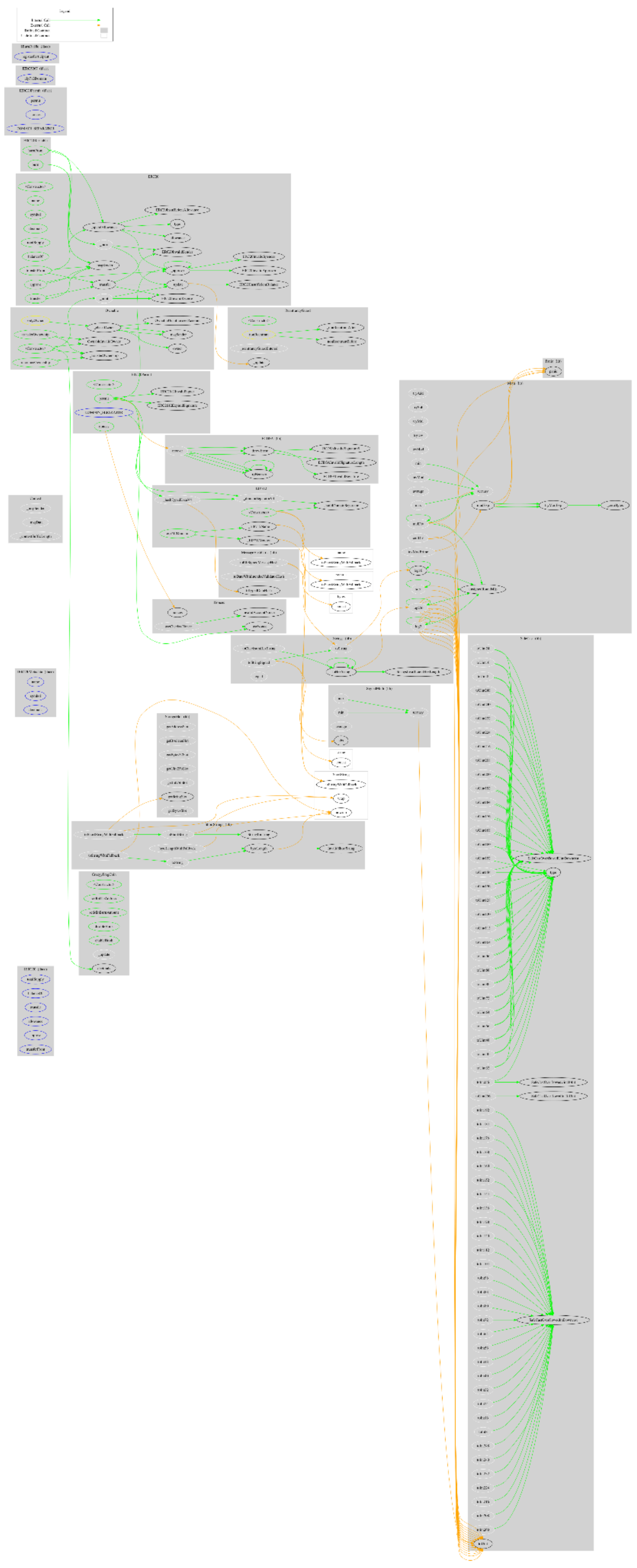
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
CrazyyFrogCoin	Implementation	ERC20, ERC20Burnable, ERC20Permit, Ownable, ReentrancyGuard		
		Public	✓	ERC20 ERC20Permit Ownable
	setRaffleAddress	Public	✓	onlyOwner
	setMinBurnAmount	Public	✓	onlyOwner
	disableHook	Public	✓	onlyOwner
	enableHook	Public	✓	onlyOwner
	_update	Internal	✓	

Inheritance Graph



Flow Graph



Summary

CrazyyFrogCoin contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. There are some functions that can be abused by the owner like stop transactions. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats.

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