Cyberscope

Audit Report

Ratpad

March 2023

SHA256 50ff6a4de5c220ec0f423b061ee6e0079125601bab3c7254a3f59d9553a4b996

Audited by © cyberscope

Table of Contents

Table of Contents 1
Review 4
Audit Updates 4
Source Files 4
Introduction 5
PresaleContract 5
Roles 5
PresaleContract 6
Roles 6
TokenLauncher 7
Roles 7
TokenLockLauncher 8
Roles 8
TokenLock 9
Roles 9
Contract Quality 10
Diagnostics 11
OLD - Override Lock Data 13
Description 13
Recommendation 13
FPR - Finalize Presale Repetition 14
Description 14
Recommendation 14
WTAPF - Withdraw Tokens After Presale Finalization 15
Description 15
Recommendation 15
PTAI - Potential Transfer Amount Inconsistency 16
Description 16
Recommendation 17
IFAP - Invalid Function Access Permissions 18
Description 18
Recommendation 18
MSC - Missing Sanity Checks 20
Description 20
Recommendation 23
RDM - Revert Descriptive Message 24
Description 24
Recommendation 24

MEM - Misleading Error Messages 25
Description 25
Recommendation 25
DD - Data Duplication 26
Description 26
Recommendation 27
RSV - Redundant Struct Variable 28
Description 28
Recommendation 28
MVN - Misleading Variables Naming 29
Description 29
Recommendation 29
DSM - Data Structure Misuse 30
Description 30
Recommendation 30
RAV - Redundant Array Variable 31
Description 31
Recommendation 31
LDAR - Large Data Array Return 32
Description 32
Recommendation 33
DSO - Data Structure Optimization 34
Description 34
Recommendation 34
DPI - Decimals Precision Inconsistency 35
Description 35
Recommendation 35
CR - Code Readability 37
Description 37
Recommendation 37
L12 - Using Variables before Declaration 39
Description 39
Recommendation 39
L17 - Usage of Solidity Assembly 40
Description 40
Recommendation 40
L02 - State Variables could be Declared Constant 41
Description 41
Recommendation 41
L04 - Conformance to Solidity Naming Conventions 42
Description 42
Recommendation 43

L06 - Missing Events Access Control 44
Description 44
Recommendation 44
L07 - Missing Events Arithmetic 45
Description 45
Recommendation 45
L09 - Dead Code Elimination 46
Description 46
Recommendation 46
L11 - Unnecessary Boolean equality 48
Description 48
Recommendation 48
L13 - Divide before Multiply Operation 49
Description 49
Recommendation 49
L16 - Validate Variable Setters 50
Description 50
Recommendation 50
L19 - Stable Compiler Version 51
Description 51
Recommendation 51
L20 - Succeeded Transfer Check 52
Description 52
Recommendation 52
Functions Analysis 53
Inheritance Graph 62
Flow Graph 63
Summary 64
Disclaimer 65
About Cyberscope 66

Review

Testing Deploy	https://testnet.bscscan.com/address/0xbd90939a090d242d6058729312e12e3dbab733aa

Audit Updates

Initial Audit	22 Mar 2023

Source Files

Filename	SHA256
LaunchPad.sol	d181782ba79a2c02b2b4e3edfdda36d3849cf4dd0e2f13d66340e71afc7f288e

Introduction

Cyberscope audited three contracts that include a Presale Launcher Master contract, a Token Launcher Master Contract, and a Lock Launcher Master Contract. These contracts are designed to deploy child contracts dynamically, which include a Presale child contract called PresaleContract, a Token child contract called ERC20, and a Lock child contract called tokenLock.

PresaleContract

The Launchpad contract serves as a factory contract for presales and is responsible for maintaining the essential registries related to presale contracts. In addition, the contract handles the configuration of fees, presale tiers and the fee receiver.

Roles

The contract roles consist of the `admin` roles. The `admin` is responsible for the configuration of fees and the fee receiver.

The users have the authority to:

- Create a new presale.
- Get fees in relation to the user's tier.
- Set lock contract.
- Get pool details.
- Get tier in relation to the address.
- Change contract tiers.

PresaleContract

The contract is a PresaleContract, which handles the presale of tokens before they are released to the public. The presale allows investors to buy tokens before they are available for purchase on a public exchange.

Roles

The contract roles consist of the `admin` and the presale `owner` role.

The `admin` is responsible for configuring badges and admin investing authorization.

The presale `owner` is responsible for editing the pool if it is allowed and finalizing or canceling the presale.

The users have the authority to:

- Claim bought token when presale finalizes.
- View bought tokens.
- Withdraw bought token with fee.
- Buy presale tokens.
- View contribution array.
- Add liquidity to the token.
- View presale badges
- View presale info.
- View if the presale is canceled or finished.

TokenLauncher

The tokenLauncher contract has several functions and events to facilitate the dynamic creation and management of ERC20 tokens.

Roles

The contract roles consist of the `admin` role. The `admin` has the authority to configure fees and change admin.

The `users` have the authority to:

- Create a new token.
- View created tokens.
- View created tokens created by a specified address.

TokenLockLauncher

The tokenLockLauncher contract is a Locker factory contract that enables the creation and management of locked ERC20 tokens.

Roles

The contract roles consist of the `admin` and the `Launcher`.

The `users` have the authority to:

- Lock token.
- View all locked tokens.

TokenLock

The `TokenLock` contract implements a simple locker. Where the contract keeps tokesn locker until the lock time elapses.

Roles

The contract roles consists of the `owner` role. The owner has the authority to Withdraw locked tokens when the lock times elapses.

The `users` can view the Locker data.

Contract Quality

The audit scope is to check for security vulnerabilities, validate the business logic and propose potential optimizations. The contract is missing the fundamental principles of a Solidity smart contract regarding gas consumption, code readability, and data structures. Given these issues, it is not advisable to assume that the contract is in a production-ready state. The development team is strongly encouraged to re-evaluate the business logic and Solidity guidelines to ensure that the contract adheres to established best practices and security measures. It is recommended that the team review the contract's vulnerabilities and optimize it accordingly to minimize the security issues and improve the contract's efficiency. The code's readability should also be improved by simplifying function definitions and using descriptive variable names, as this will enhance the contract's auditability and maintenance.

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	OLD	Overide Lock Data	Unresolved
●	FPR	Finalize Presale Repetition	Unresolved
●	WTAPF	Withdraw Tokens After Presale Finalization	Unresolved
●	PTAI	Potential Transfer Amount Inconsistency	Unresolved
●	IFAP	Invalid Function Access Permissions	Unresolved
●	MSC	Missing Sanity Checks	Unresolved
●	RDM	Revert Descriptive Message	Unresolved
●	MEM	Misleading Error Messages	Unresolved
●	DD	Data Duplication	Unresolved
●	RSV	Redundant Struct Variable	Unresolved
●	MVN	Misleading Variables Naming	Unresolved
●	DSM	Data Structure Misuse	Unresolved
●	RAV	Redundant Array Variable	Unresolved

●	LDAR	Large Data Array Return	Unresolved
●	DSO	Data Structure Optimization	Unresolved
●	DPI	Decimals Precision Inconsistency	Unresolved
●	CR	Code Readability	Unresolved
●	L12	Using Variables before Declaration	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L06	Missing Events Access Control	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L11	Unnecessary Boolean equality	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L19	Stable Compiler Version	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

OLD - Override Lock Data

Criticality	Critical
Location	LaunchPad.sol#L637,671
Status	Unresolved

Description

The Lock data can be overridden after a presale is completed. If someone created a presale with the same address, the lock data will be overridden. This could lead to unintended behavior, potential security vulnerabilities, and unfair treatment of investors who participated in the presale.

```
function createPresale(
    address[] memory _token_owner_admin_currency,
    string[] memory _title_symbol_SocialMedia,
    uint256[] memory _noOfTokens_price_max_min_vesting_month_start_end,
    string memory _hash
) public payable {
    ...
    TokenPresale[_token_owner_admin_currency[0]] = address(tx1);
    TokenPresaleLocked[_token_owner_admin_currency[0]][
    ...
}
```

Recommendation

It is recommended updating the presale function to ensure that it cannot be override old data. This can be achieved by adding a condition that checks whether the presale has ended before and whether the same token address is used. Additionally, we recommend thoroughly testing the updated function to verify that it functions as intended and does not introduce any new issues.

FPR - Finalize Presale Repetition

Criticality	Critical
Location	LaunchPad.sol#L897
Status	Unresolved

Description

The finalize function in a presale contract is used to lock the presale and distribute tokens to the buyers. Once the presale is finalized, the contract is not supposed to accept any more contributions or allow any changes to be made. However, the current implementation allows the owner to call the finalize function multiple times, which can lead to unexpected behavior and result in the function reverting.

```
function finalize() public {...}
```

Recommendation

To address this issue, the finalize function should be modified to include a check that ensures it can only be called once. One way to achieve this could be to add a modifier or a guard mechanism to the finalize function that checks if the presale has been finalized before allowing the function to be executed. Additionally, it is recommended to thoroughly test the contract to ensure that all possible scenarios have been considered.

WTAPF - Withdraw Tokens After Presale Finalization

Criticality	Critical
Location	LaunchPad.sol#L938
Status	Unresolved

Description

In the presale contract, there is a function that allows the owner to cancel the presale by changing the variable `selfInfo.finalized`. This variable is checked in various parts of the contract to ensure that the presale is still active. However, this flag can be set to false even after the presale has been finalized. This can lead to unexpected behavior and loss of funds for investors who have already contributed to the presale.

```
function cancel() public {
    require(
      msg.sender == selfInfo._token_owner_admin_currency[1],
      "Only owner allowed"
    );
    IERC20 Token = IERC20(selfInfo._token_owner_admin_currency[0]);
    Token.transfer(
      selfInfo._token_owner_admin_currency[1],
      Token.balanceOf(address(this))
    );
    selfInfo.finalized = 1;
}
```

Recommendation

To prevent unexpected behavior and loss of funds, it is recommended to add additional checks in the presale contract to prevent the owner from canceling the presale after it has been finalized. One way to achieve this could be to add a modifier or a guard mechanism to the cancel function that checks if the presale has been finalized before allowing the function to be executed. Additionally, it is recommended to thoroughly test the contract to ensure that all possible scenarios have been considered.

PTAI - Potential Transfer Amount Inconsistency

Criticality	Critical
Location	LaunchPad.sol#L611,795,814,927,929,941,1129
Status	Unresolved

Description

The `transfer()` and `transferFrom()` functions are used to transfer a specified amount of tokens to an address. The fee or tax is an amount that is charged to the sender of an ERC20 token when tokens are transferred to another address. According to the specification, the transferred amount could potentially be less than the expected amount. This may produce inconsistency between the expected and the actual behavior.

The following example depicts the diversion between the expected and actual amount.

Tax	Amount	Expected	Actual
No Tax	100	100	100
10% Tax	100	100	90

```
_Token.transferFrom(
    msg.sender,
    address(tx1),
    (_noOfTokens_price_max_min_vesting_month_start_end[0] +
    _noOfTokens_price_max_min_vesting_month_start_end[18])
);
Token.transfer(msg.sender, Entitlement);
Token.transfer(msg.sender, netEntitlement);
Token.transfer(selfInfo._token_owner_admin_currency[1], BalanceTokens);
Token.transfer(
    selfInfo._token_owner_admin_currency[1],
    Token.balanceOf(address(this))
);
Token.transferFrom(msg.sender, address(tx1), _amount);
```

Recommendation

The team is advised to take into consideration the actual amount that has been transferred instead of the expected.

It is important to note that an ERC20 transfer tax is not a standard feature of the ERC20 specification, and it is not universally implemented by all ERC20 contracts. Therefore, the contract could produce the actual amount by calculating the difference between the transfer call.

```
Actual Transferred Amount = Balance After Transfer - Balance Before Transfer
```

IFAP - Invalid Function Access Permissions

Criticality	Critical
Location	LaunchPad.sol#L671,689
Status	Unresolved

Description

The contract has a function that can be called from any address, including addresses that are not authorized to change the contract's core configuration variables. This function should be restricted to only authorized addresses to prevent any unintended changes to the contract's state and behavior. Allowing unrestricted access to such functions can lead to vulnerabilities in the smart contract.

```
function changeTiers(uint256 min1, uint256 min2) public {
    tier1ratboyMin = min1;
    tier2ratboyMin = min2;
}

function setLockContract(
    address _token,
    uint256 _amount,
    string memory _title,
    uint256 _time,
    address _contract,
    bool _LP
) public {
    tokenLockStruct memory tx1 = tokenLockStruct(
        _title,
        _amount,
        _contract,
        _time,
        _LP
    );
    address add = TokenPresale[_token];
    TokenLockContracts[_token].push(tx1);
    PresaleLockContracts[add].push(tx1);
}
```

Recommendation

The function should be updated to restrict access only to authorized addresses. The contract owner or authorized parties should be granted permission to access and modify core configuration variables. By limiting access to the function, the contract can maintain its intended state and behavior, preventing potential vulnerabilities. Access control should be implemented to ensure that only authorized parties can modify sensitive variables. This can be achieved using a modifier or by checking the caller's address within the function before allowing it to execute.

MSC - Missing Sanity Checks

Criticality	Critical
Location	LaunchPad.sol
Status	Unresolved

Description

The contract is processing variables that have not been properly sanitized and checked that they form the proper shape. These variables may produce vulnerability issues.

The variables `feeOnFinalization`, `withdrawFee`, and fee are not properly sanitized. The values could exceed the denominators' values and potentially create unwanted behavior.

```
//contract/Launchpad
function changeFeeForPoolCreation(
    uint256 _feeFinalization,
    uint256 tier1Fee,
    uint256 tier2fee,
    uint256 tier3fee
) public onlyAdmin {
    feeOnFinalization = _feeFinalization;
    ...
}
PresaleContract tx1 = new PresaleContract(
    ...
    feeOnFinalization,
    ...
);

//contract/PresaleContract
constructor(
    address[] memory _token_owner_admin_currency,
    string[] memory _title_symbol_SocialMedia,
    uint256[] memory _noOfTokens_price_max_min_vesting_month_start_end,
    string memory _hash,
    uint256 _fee,
    address _receiver,
    address _admin
) {
    ...
    fee = _fee;
    ...
}
uint256 adminShare = totalBalance * fee / 100;

//contract/Launchpad
function changeWithdrawFee(uint256 _fee) public onlyAdmin {
    withdrawFee = _fee;
}
//contract/PresaleContract
function withdraw() public payable {
    ...
    uint256 withdrawFee = Factory.withdrawFee();
    ...
    uint256 _fee = DeemedCancelled() ? 0 : (amount * withdrawFee) /
100;
    ...,
}
```

The variable `feeReceiverAddress` is not properly sanitized. It could be set to zero address.

```
//contract/Launchpad
function changeReceiver(address _receiver) public onlyAdmin {
    feeReceiverAddress = _receiver;
}
```

The `time` variable is not properly sanitized. The `time` could be set to now.

```
//contract/tokenLockLauncher
tokenLock tx1 = new tokenLock(
    ...
    _time,
    block.timestamp,
    ...
);

contract tokenLock {
    struct Info {
        ...
        uint256 Now;
        uint256 Time;
        ...
    }
}
```

The `min` and `max`, `start` and `end` and the time variables of the `_noOfTokens_price_max_min_vesting_month_start_end` array are not properly sanitized.

The addresses of the `_token_owner_admin_currency` are not properly sanitized.

```
//contract/Presale-Launchpad
_noOfTokens_price_max_min_vesting_month_start_end
The addresses of the `_token_owner_admin_currency` are not properly
sanitized.
```

The arguments `min1` and `min2` are not properly sanitized.

```
function changeTiers(uint256 min1, uint256 min2) public {
    tier1ratboyMin = min1;
    tier2ratboyMin = min2;
}
```

Recommendation

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

- The fee nominators should be lower than 100.
- All the addresses should not be set to zero addresses.
- The `time` variable should be greater than the current timestamp.
- The contract should examine and properly sanitize the tier configuration.

RDM - Revert Descriptive Message

Criticality	Minor / Informative
Location	LaunchPad.sol#L817,889
Status	Unresolved

Description

The `revert()` function is used to halt the execution of a contract and revert any changes made to the contract's state. The contract does not provide a descriptive message to the `revert()` function.

```
require(msg.value >=
selfInfo._noOfTokens_price_max_min_vesting_month_start_end[3] &&
msg.value <=
selfInfo._noOfTokens_price_max_min_vesting_month_start_end[2]
                    ||

selfInfo._noOfTokens_price_max_min_vesting_month_start_end[5] <=
block.timestamp,"Some Error Occured"
                );
require(msg.sender == selfInfo._token_owner_admin_currency[1],"Some
error occured");
```

Recommendation

The team is suggested to provide a descriptive message to the `revert()` function. This message can be used to provide additional context about the error that occurred or to explain why the contract execution was halted. This can be useful for debugging and for providing more information to users that interact with the contract.

MEM - Misleading Error Messages

Criticality	Minor / Informative
Location	LaunchPad.sol#L845
Status	Unresolved

Description

The contract is using misleading error messages. These error messages do not accurately reflect the problem, making it difficult to identify and fix the issue. As a result, the users will not be able to find the root cause of the error.

A presale could either be finalized or canceled.

```
require(selfInfo.finalized == 0,"Presale is finalized");
```

Recommendation

The team is advised to carefully review the source code in order to address these issues. To accelerate the debugging process and mitigate these issues, the team should use more specific and descriptive error messages.

DD - Data Duplication

Criticality	Minor / Informative
Location	LaunchPad.sol#L1117,1189,626
Status	Unresolved

Description

The TopCarde contract duplicates `WKDNFTToken` variables. This can result in unnecessary code complexity, potential inconsistencies in the data, and difficulty in maintaining the codebase.

```
//tokenLock
IERC20 _token = IERC20(_Token);
Data.name = _token.name();
Data.symbol = _token.symbol();
Data.decimals = _token.decimals();
//tokenLockLauncher
IERC20 Token = IERC20(_token);
string memory _symbol = Token.symbol();
string memory _name = Token.name();
uint256 _decimals = Token.decimals();
```

The Launchpad contract duplicates `Presale` data on mapping. All the necessary data is already saved on the presale contract. This can result in unnecessary code complexity, potential inconsistencies in the data, and difficulty in maintaining the codebase.

```
PresaleContract tx1 = new PresaleContract(
    _token_owner_admin_currency,
    _title_symbol_SocialMedia,
    _noOfTokens_price_max_min_vesting_month_start_end,
    _hash,

    feeOnFinalization,
    feeReceiverAddress,
    admin
);

Presale memory tx2 = Presale(
    PresaleIndex,
    address(tx1),
    _token_owner_admin_currency,
    _title_symbol_SocialMedia,
    _noOfTokens_price_max_min_vesting_month_start_end,
    _hash,
    0,
    0
);
PresaleArray.push(tx2);
```

Recommendation

To prevent the duplication of values in multiple contracts, one option is to directly access the variables from the other contract. Another approach is to create a separate contract that contains the shared data or variables, which can then be inherited by other contracts that need access to them.

It is recommended to remove redundant data structures.

RSV - Redundant Struct Variable

Criticality	Minor / Informative
Location	LaunchPad.sol#L727
Status	Unresolved

Description

The contract contains a struct with a variable that is redundant and not used in any of the contract's logic. This variable does not provide any meaningful functionality to the contract and adds unnecessary complexity to the codebase. Additionally, this variable can consume memory and increase the gas cost of contract execution.

The `cycle` is not utilized in the contract's implementation. Hence, it is redundant.

```
struct Presale {
    ...
    uint256 cycle;
    ...
}
```

Recommendation

It is recommended to remove the redundant variable from the contract to simplify the codebase and reduce unnecessary gas costs. Before removing the variable, it is important to ensure that it is not used anywhere else in the contract, including any external dependencies. By removing the redundant variable, the contract can become more streamlined and efficient, potentially reducing the chances of errors and making it easier to maintain and audit in the future.

MVN - Misleading Variables Naming

Criticality	Minor / Informative
Location	LaunchPad.sol#L706
Status	Unresolved

Description

Variables can have misleading names if their names do not accurately reflect the value they contain or the purpose they serve. The contract uses some variable names that are too generic or do not clearly convey the information stored in the variable. Misleading variable names can lead to confusion, making the code more difficult to read and understand.

The variable `noOfClaims` is used to check whether a user has performed a claim or not. Hence, it is not the number of claims.

```
mapping(address => uint256) public noOfClaims;
```

The variable `_busd` is used to depicts the native blockchain currency.

```
uint256 _busd = msg.value;
```

Recommendation

It's always a good practice for the contract to contain variable names that are specific and descriptive. The team is advised to keep in mind the readability of the code.

- The variable `noOfClaims` could be renamed to `hasClaimed`.
- The variable `_busd` could be renamed to `_bnb`.

DSM - Data Structure Misuse

Criticality	Minor / Informative
Location	LaunchPad.sol#L706
Status	Unresolved

Description

The contract uses the valuable `noOfClaims` as an array. The business logic of the contract does not require number of claims. It is only used to check whether a user has performed a claim or not. Therefore, it can be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

```
mapping(address => uint256) public noOfClaims;
```

Recommendation

The contract could use mapping of address to boolean. That way it will improve the efficiency and performance of the source code.

RAV - Redundant Array Variable

Criticality	Minor / Informative
Location	LaunchPad.sol#L532
Status	Unresolved

Description

The contract contains an array with a variable that is redundant and not used in any of the contract's logic. This variable does not provide any meaningful functionality to the contract and adds unnecessary complexity to the codebase. Additionally, this variable can consume memory and increase the gas cost of contract execution.

The indexes 2 and 3 of the `_token_owner_admin_currency` array are never used in the contract.

```
address[] _token_owner_admin_currency;
```

Recommendation

It is recommended to remove the redundant variable from the contract to simplify the codebase and reduce unnecessary gas costs. Before removing the variable, it is important to ensure that it is not used anywhere else in the contract, including any external dependencies. By removing the redundant variable, the contract can become more streamlined and efficient, potentially reducing the chances of errors and making it easier to maintain and audit in the future.

LDAR - Large Data Array Return

Criticality	Minor / Informative
Location	LaunchPad.sol#L
Status	Unresolved

Description

The contract contains a function that returns a large array of data, which could make it difficult to use in practice. If this function is called frequently, it could cause performance issues.

The array `PresaleArray`, `arr1` might return a large array of data.

```
function getPoolDetails()
    public
    view
    returns (Presale[] memory, IGOData[] memory)
{
    IGOData[] memory arr1 = new IGOData[](PresaleArray.length);
    for (uint256 i = 0; i < PresaleArray.length; i++) {
      PresaleContract tx1 =
PresaleContract(payable(PresaleArray[i]._address));
      (
        ,
        ,
        uint256 purchasedTokens,
        uint256 investedAmount,
        uint256 finalized,
        address LP,
        ,
        bool adminAllowed
      ) = tx1.selfInfo();
      IGOData memory tx2 = IGOData(
        investedAmount,
        purchasedTokens,
        finalized,
        LP,
        adminAllowed
      );
      arr1[i] = tx2;
    }
    return (PresaleArray, arr1);
}
```

Recommendation

It is recommended to consider implementing pagination to break the large array into smaller, more manageable chunks. This approach can reduce the computational burden on the function and make it easier to use in practice. By implementing pagination, the contract can improve its usability and overall performance.

DSO - Data Structure Optimization

Criticality	Minor / Informative
Location	LaunchPad.sol#L702,984,988
Status	Unresolved

Description

The contract utilizes the `Badges` variable as an array, but the business logic of the contract does not require the array to be dynamic. This results in unnecessary memory usage, slower execution times, and more operations being performed than necessary. Therefore, optimizing this code segment could help reduce these issues.

```
bool public Badges;

function addBadges(bool[] memory _badges) public onlyAdmin {
    Badges = _badges;
}

function getBadges() public view returns (bool[] memory) {
    return Badges;
}
```

Recommendation

The contract could use a fixed data structure. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

DPI - Decimals Precision Inconsistency

Criticality	Minor / Informative
Location	LaunchPad.sol#L884
Status	Unresolved

Description

The decimals field of a contract's ERC20 token can be used to specify the number of decimal places that the token uses. For example, if decimals is set to `8`, it means that the smallest unit of the token is `0.00000001`, and if decimals are set to `18`, it means that the smallest unit of the token is `0.000000000000000001`.

However, there is an inconsistency in the way that the decimals field is handled in some ERC20 contracts. The ERC20 specification does not specify how the decimals field should be implemented, and as a result, some contracts use different precision numbers.

This inconsistency can cause problems when interacting with these contracts, as it is not always clear how the decimals field should be interpreted. For example, if a contract expects the decimals field to be 18 digits, but the contract being interacted with uses 8 digits, the result of the interaction may not be what was expected.

```
uint256 tokens = ((investedAmount * (10 ** decimals)) *
    selfInfo._noOfTokens_price_max_min_vesting_month_start_end[1]) /
    (10 ** 18);
```

Recommendation

To avoid these issues, it is important to carefully review the implementation of the decimals field of the underlying tokens. The team is advised to normalize each decimal to one single source of truth. A recommended way is to scale all the decimals to the greatest token's decimal. Hence, the contract will not lose precision in the calculations.

The following example depicts 3 tokens with different decimals precision.

ERC20	Decimals

Token 1	6
Token 2	9
Token 3	18

All the decimals could be normalized to 18 since it represents the ERC20 token with the greatest digits.

CR - Code Readability

Criticality	Minor / Informative
Location	LaunchPad.sol
Status	Unresolved

Description

There are array variables that contain multiple different variables of different contexts. This can result in difficulties to understand the purpose of these variables and how they're used within the contract.

When there are multiple array variables that contain multiple different variables of different contexts, it can be difficult to discern the purpose and usage of each variable. This can increase the likelihood of errors and omissions during the audit, potentially leading to security vulnerabilities and other issues.

```
address[] memory _token_owner_admin_currency
string[] memory _title_symbol_SocialMedia
uint256[] memory _noOfTokens_price_max_min_vesting_month_start_end

require(
    (msg.value >=
    selfInfo._noOfTokens_price_max_min_vesting_month_start_end[3] &&
    msg.value <=
    selfInfo._noOfTokens_price_max_min_vesting_month_start_end[2]) ||
    selfInfo._noOfTokens_price_max_min_vesting_month_start_end[5] <=
    block.timestamp,
    "Some Error Occured"
);
```

Recommendation

To improve the readability and clarity of the Solidity contract, it's recommended to use different variables of different and group variables with a similar context only. By doing so, auditors and developers can better understand the purpose and usage of each variable, reducing the likelihood of errors and omissions during the audit.

In addition, it's recommended to use clear and descriptive variable names, making it easier to understand the purpose and usage of each variable. By using descriptive

variable names, developers can help to ensure that the Solidity contract is more readable and easier to audit.

L12 - Using Variables before Declaration

Criticality	Minor / Informative
Location	ERC20.sol#L1724,1726
Status	Unresolved

Description

The contract is using a variable before the declaration. This is usually happening either if it has not been declared yet or if the variable has been declared in a different scope. It is not a good practice to use a local variable before it has been declared.

```
bytes4 retval
bytes memory reason
```

Recommendation

By declaring local variables before using them, contract ensures that it operates correctly. It's important to be aware of this rule when working with local variables, as using a variable before it has been declared can lead to unexpected behavior and can be difficult to debug.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	ERC20.sol#L988,1732
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 {
                    let returndata_size := mload(returndata)
                    revert(add(32, returndata), returndata_size)
                }

assembly {
                        revert(add(32, reason), mload(reason))
                    }
```

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.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	LaunchPad.sol#L550,739
Status	Unresolved

Description

State variables can be declared as constant using the constant keyword. This means that the value of the state variable cannot be changed after it has been set. Additionally, the constant variables decrease gas consumption of the corresponding transaction.

```
uint256 minSuperRat = 1
bool public publicSale = false
```

Recommendation

Constant state variables can be useful when the contract wants to ensure that the value of a state variable cannot be changed by any function in the contract. This can be useful for storing values that are important to the contract's behavior, such as the contract's address or the maximum number of times a certain function can be called. The team is advised to add the constant keyword to state variables that never change.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	LaunchPad.sol#L66,477,534,535,536,537,538,539,540,546,548,554,557,591,602,606,610,615,616,617,618,659,668,669,670,671,672,673,717,740,745,750,758,805,810,811,842,877,982,994,1009,1014,1016,1018,1030,1035,1042,1043,1044,1045,1068,1083,1085,1087,1103,1111,1112,1113,1114,1115,1116,1117,1160,1174
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.

```
uint8 private Decimals
function WETH() external pure returns (address);
uint256 public PresaleIndex
Presale[] public PresaleArray
mapping(address => uint256) public PresaleMapping
mapping(address => address) public TokenPresale
mapping(address => mapping(address => bool)) public TokenPresaleLocked
mapping(address => tokenLockStruct[]) public TokenLockContracts
mapping(address => tokenLockStruct[]) public PresaleLockContracts
IERC721 public Superrat =
IERC721(0xf4C91AB5B5c40ba93540c0703954fC148C49f293)
tokenLockLauncher public LockLauncher
event presaleCreated(address indexed presaleAddress);

...
```

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/v0.8.17/style-guide.html#naming-convention.

L06 - Missing Events Access Control

Criticality	Minor / Informative
Location	LaunchPad.sol#L611
Status	Unresolved

Description

Events are a way to record and log information about changes or actions that occur within a contract. They are often used to notify external parties or clients about events that have occurred within the contract, such as the transfer of tokens or the completion of a task. There are functions that have no event emitted, so it is difficult to track off-chain changes.

```
admin = _admin
```

Recommendation

To avoid this issue, it's important to carefully design and implement the events in a contract, and to ensure that all required events are included. It's also a good idea to test the contract to ensure that all events are being properly triggered and logged.

By including all required events in the contract and thoroughly testing the contract's functionality, the contract ensures that it performs as intended and does not have any missing events that could cause issues.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	LaunchPad.sol#L597,1032
Status	Unresolved

Description

Events are a way to record and log information about changes or actions that occur within a contract. They are often used to notify external parties or clients about events that have occurred within the contract, such as the transfer of tokens or the completion of a task.

It's important to carefully design and implement the events in a contract, and to ensure that all required events are included. It's also a good idea to test the contract to ensure that all events are being properly triggered and logged.

```
feeForPoolTier1 = tier1Fee
fee = _fee
```

Recommendation

By including all required events in the contract and thoroughly testing the contract's functionality, the contract ensures that it performs as intended and does not have any missing events that could cause issues with its arithmetic.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	LaunchPad.sol#L352
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 _burn(address account, uint256 amount) internal virtual {
        require(account != address(0), "ERC20: burn from the zero
address");

        _beforeTokenTransfer(account, address(0), amount);

        uint256 accountBalance = _balances[account];
...
            _totalSupply -= amount;
        }

        emit Transfer(account, address(0), amount);

        _afterTokenTransfer(account, address(0), amount);
    }
```

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.

L11 - Unnecessary Boolean equality

Criticality	Minor / Informative
Location	LaunchPad.sol#L759
Status	Unresolved

Description

Boolean equality is unnecessary when comparing two boolean values. This is because a boolean value is either true or false, and there is no need to compare two values that are already known to be either true or false.

it's important to be aware of the types of variables and expressions that are being used in the contract's code, as this can affect the contract's behavior and performance. The comparison to boolean constants is redundant. Boolean constants can be used directly and do not need to be compared to true or false.

```
require(selfInfo.adminAllowed == true,"Presale not authorized")
```

Recommendation

Using the boolean value itself is clearer and more concise, and it is generally considered good practice to avoid unnecessary boolean equalities in Solidity code.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	LaunchPad.sol#L829,834,851,857,911,913
Status	Unresolved

Description

It is important to be aware of the order of operations when performing arithmetic calculations. This is especially important when working with large numbers, as the order of operations can affect the final result of the calculation. Performing divisions before multiplications may cause loss of prediction.

```
uint256 liquidity = ( remainder *
selfInfo._noOfTokens_price_max_min_vesting_month_start_end[11]) / 100
uint256 tokens = liquidity * (10**tDecimals) *
selfInfo._noOfTokens_price_max_min_vesting_month_start_end[10] /
(10**18)
```

Recommendation

To avoid this issue, it is recommended to carefully consider the order of operations when performing arithmetic calculations in Solidity. It's generally a good idea to use parentheses to specify the order of operations. The basic rule is that the multiplications should be prior to the divisions.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	LaunchPad.sol#L607,611,798,802,1037,1038,1106
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.

```
feeReceiverAddress = _receiver
admin = _admin
feeReceiver = _receiver
feeReceiver = receiver
admin = _user
```

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.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	LaunchPad.sol#L3,15,55
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 ^0.8.0;
```

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	LaunchPad.sol#L650,825,838,928,933,942,1132,1203
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.

```
_Token.transferFrom(
            msg.sender,
            address(tx1),

(_noOfTokens_price_max_min_vesting_month_start_end[0]+_noOfTokens_price_m
ax_min_vesting_month_start_end[18])
        )
Token.transfer(msg.sender, Entitlement)
...
Token.transfer(
                selfInfo._token_owner_admin_currency[1],
                BalanceTokens
            )
Token.transfer(0x000000000000000000000000000000000000dEaD, BalanceTokens)
Token.transfer(selfInfo._token_owner_admin_currency[1],Token.balanceOf(ad
dress(this)))
Token.transferFrom(msg.sender, address(tx1), _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**
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	✓	
	_spendAllowance	Internal	✓	
	_beforeTokenTransfer	Internal	✓	
	_afterTokenTransfer	Internal	✓	
ERC20Burnable	Implementation	Context, ERC20		
	burn	Public	✓	-
	burnFrom	Public	✓	-
IERC20A	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	name	External		-
	symbol	External		-
	decimals	External		-

	transferFrom	External	✓	-
Strings	Library			
	toString	Internal		
	toHexString	Internal		
	toHexString	Internal		
	toHexString	Internal		
Address	Library			
	isContract	Internal		
	sendValue	Internal	✓	
	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionStaticCall	Internal		
	functionStaticCall	Internal		
	functionDelegateCall	Internal	✓	
	functionDelegateCall	Internal	✓	
	verifyCallResult	Internal		
IERC721Receiver	Interface			
	onERC721Received	External	✓	-
IERC165	Interface			
	supportsInterface	External		-
ERC165	Implementation	IERC165		
	supportsInterface	Public		-

IERC721	Interface	IERC165		
	balanceOf	External		-
	ownerOf	External		-
	safeTransferFrom	External	✓	-
	safeTransferFrom	External	✓	-
	transferFrom	External	✓	-
	approve	External	✓	-
	setApprovalForAll	External	✓	-
	getApproved	External		-
	isApprovedForAll	External		-
IERC721Enumerable	Interface	IERC721		
	totalSupply	External		-
	tokenOfOwnerByIndex	External		-
	tokenByIndex	External		-
IERC721Metadata	Interface	IERC721		
	name	External		-
	symbol	External		-
	tokenURI	External		-
ERC721	Implementation	Context, ERC165, IERC721, IERC721Metadata		
		Public	✓	-
	supportsInterface	Public		-
	balanceOf	Public		-
	ownerOf	Public		-

	name	Public		-
	symbol	Public		-
	tokenURI	Public		-
	_baseURI	Internal		
	approve	Public	✓	-
	getApproved	Public		-
	setApprovalForAll	Public	✓	-
	isApprovedForAll	Public		-
	transferFrom	Public	✓	-
	InternaltransferFrom	Internal	✓	
	safeTransferFrom	Public	✓	-
	safeTransferFrom	Public	✓	-
	_safeTransfer	Internal	✓	
	_exists	Internal		
	_isApprovedOrOwner	Internal		
	_safeMint	Internal	✓	
	_safeMint	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_transfer	Internal	✓	
	_approve	Internal	✓	
	modifyTransfer	Public	✓	-
	_setApprovalForAll	Internal	✓	
	_checkOnERC721Received	Private	✓	
	_beforeTokenTransfer	Internal	✓	
	_afterTokenTransfer	Internal	✓	
IUniswapV2Factory	Interface			
	feeTo	External		-

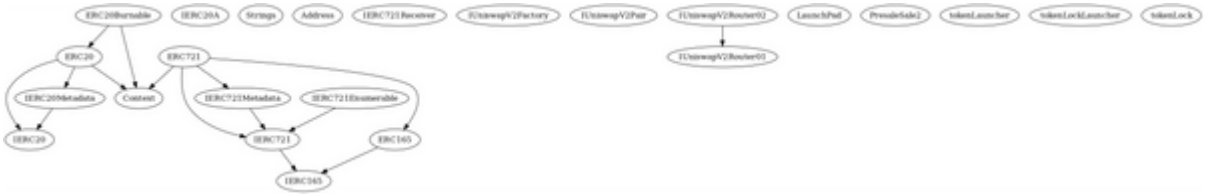
	feeToSetter	External		-
	getPair	External		-
	allPairs	External		-
	allPairsLength	External		-
	createPair	External	✓	-
	setFeeTo	External	✓	-
	setFeeToSetter	External	✓	-
IUniswapV2Pair	Interface			
	name	External		-
	symbol	External		-
	decimals	External		-
	totalSupply	External		-
	balanceOf	External		-
	allowance	External		-
	approve	External	✓	-
	transfer	External	✓	-
	transferFrom	External	✓	-
	DOMAIN_SEPARATOR	External		-
	PERMIT_TYPEHASH	External		-
	nonces	External		-
	permit	External	✓	-
	MINIMUM_LIQUIDITY	External		-
	factory	External		-
	token0	External		-
	token1	External		-
	getReserves	External		-
	price0CumulativeLast	External		-
	price1CumulativeLast	External		-

	kLast	External		-
	mint	External	✓	-
	burn	External	✓	-
	swap	External	✓	-
	skim	External	✓	-
	sync	External	✓	-
	initialize	External	✓	-
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		-

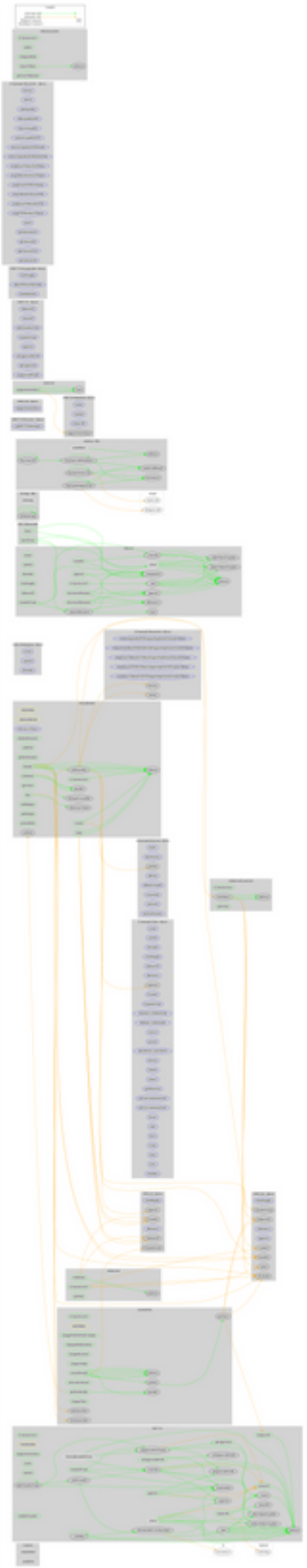
IUniswapV2Router02	Interface	IUniswapV2Router01		
	removeLiquidityETHSupportingFeeOnTransferTokens	External	✓	-
	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	✓	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
LaunchPad	Implementation			
		Public	✓	-
	changeFeeForPoolCreation	Public	✓	onlyAdmin
	changeWithdrawFee	Public	✓	onlyAdmin
	changeReceiver	Public	✓	onlyAdmin
	changeAdmin	Public	✓	onlyAdmin
	createPresale	Public	Payable	-
	getFee	Public		-
	setLockContract	Public	✓	-
	getPoolDetails	Public		-
	getTiers	Public		-
	changeTiers	Public	✓	-
PresaleSale2	Implementation			
		External	Payable	-
		Public	✓	-
	adminAllowance	Public	✓	onlyAdmin
	editPool	Public	✓	_adminAllowed
	claim	Public	✓	-
	getEntitlement	Public		-

	withdraw	Public	Payable	-
	Buy	Public	Payable	_adminAllowed
	getUsers	Public		-
	finalize	Public	✓	-
	cancel	Public	✓	-
	addLiquidity	Public	✓	-
	addBadges	Public	✓	onlyAdmin
	getBadges	Public		-
	getSelfInfo	Public		-
	DeemedCancelled	Public		-
	whiteListCheck	Public		-
tokenLauncher	Implementation			
		Public	✓	-
	setfee	Public	✓	-
	changeAdmin	Public	✓	-
	launchToken	Public	✓	-
	getUserTokenList	Public		-
tokenLockLauncher	Implementation			
		Public	✓	-
	lockToken	Public	✓	-
	getArray	Public		-
tokenLock	Implementation			
		Public	✓	-
	withdraw	Public	✓	-
	getData	Public		-

Inheritance Graph



Flow Graph



Summary

Ratpad contract implements a token, locker and utility 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