



Cyberscope

Audit Report

Canyont

March 2025

Network BSC

Address 0x7070f69ee73a350724f311f8132935ebbb78e6e2

Audited by © cyberscope

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

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

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	IR	Invalid Requirement	Acknowledged
●	TSD	Total Supply Diversion	Acknowledged
●	AAO	Accumulated Amount Overflow	Acknowledged
●	CC	Commented Code	Acknowledged
●	CCR	Contract Centralization Risk	Acknowledged
●	MMN	Misleading Method Naming	Acknowledged
●	MC	Missing Check	Acknowledged
●	MEE	Missing Events Emission	Acknowledged
●	PAM	Possible Authority Manipulation	Acknowledged
●	RF	Redundant Functionality	Acknowledged
●	RAE	Referral Address Exploit	Acknowledged
●	RFM	Renounce Functionality Missing	Acknowledged
●	SCF	Stopping Core Functionality	Acknowledged
●	UV	Unused Variables	Acknowledged

●	L02	State Variables could be Declared Constant	Acknowledged
●	L04	Conformance to Solidity Naming Conventions	Acknowledged
●	L05	Unused State Variable	Acknowledged
●	L07	Missing Events Arithmetic	Acknowledged
●	L09	Dead Code Elimination	Acknowledged
●	L15	Local Scope Variable Shadowing	Acknowledged
●	L16	Validate Variable Setters	Acknowledged
●	L19	Stable Compiler Version	Acknowledged

Table of Contents

Analysis	1
Diagnostics	2
Table of Contents	4
Risk Classification	7
Review	8
Audit Updates	8
Source Files	8
Findings Breakdown	9
MT - Mints Tokens	10
Description	10
Recommendation	11
Team Update	11
IR - Invalid Requirement	12
Description	12
Recommendation	12
TSD - Total Supply Diversion	13
Description	13
Recommendation	14
Team Update	14
AAO - Accumulated Amount Overflow	15
Description	15
Recommendation	15
CC - Commented Code	16
Description	16
Recommendation	16
CCR - Contract Centralization Risk	17
Description	17
Recommendation	17
MMN - Misleading Method Naming	18
Description	18
Recommendation	18
MC - Missing Check	19
Description	19
Recommendation	20
MEE - Missing Events Emission	21
Description	21
Recommendation	21
PAM - Possible Authority Manipulation	22
Description	22

Recommendation	22
RF - Redundant Functionality	23
Description	23
Recommendation	24
RAE - Referral Address Exploit	25
Description	25
Recommendation	25
RFM - Renounce Functionality Missing	26
Description	26
Recommendation	26
SCF - Stopping Core Functionality	27
Description	27
Recommendation	28
UV - Unused Variables	29
Description	29
Recommendation	29
L02 - State Variables could be Declared Constant	30
Description	30
Recommendation	30
L04 - Conformance to Solidity Naming Conventions	31
Description	31
Recommendation	32
L05 - Unused State Variable	33
Description	33
Recommendation	33
L07 - Missing Events Arithmetic	34
Description	34
Recommendation	34
L09 - Dead Code Elimination	35
Description	35
Recommendation	36
L15 - Local Scope Variable Shadowing	37
Description	37
Recommendation	37
L16 - Validate Variable Setters	38
Description	38
Recommendation	38
L19 - Stable Compiler Version	39
Description	39
Recommendation	39
Functions Analysis	40
Inheritance Graph	42

Flow Graph	43
Summary	44
Disclaimer	45
About Cyberscope	46

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	Canyont
Compiler Version	v0.5.0+commit.1d4f565a
Optimization	200 runs
Explorer	https://bscscan.com/address/0x7070f69ee73a350724f311f8132935ebbb78e6e2
Address	0x7070f69ee73a350724f311f8132935ebbb78e6e2
Network	BSC
Symbol	CTYN
Decimals	18
Total Supply	201.000.000

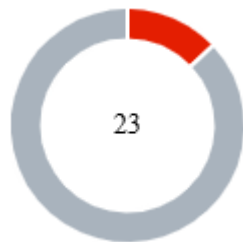
Audit Updates

Initial Audit	13 Mar 2025
---------------	-------------

Source Files

Filename	SHA256
Canyont.sol	9cef15f96edc145f66dcda939406e6d110a4a5dae9d1dd71bcd33c212dfc2beb

Findings Breakdown



Critical	3
Medium	0
Minor / Informative	20

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	3	0	0
Medium	0	0	0	0
Minor / Informative	0	20	0	0

MT - Mints Tokens

Criticality	Critical
Location	Canyont.sol#L542,556
Status	Acknowledged

Description

Tokens can be minted by the use of `buy` function. There is no limit on how many tokens can be minted and additionally the owner can take advantage of it by decreasing the price to a very small value and mint a very large amount of tokens. As a result, the contract tokens will be highly inflated. Additionally the intended use of `airdrop` was also to `mint` tokens.

```
function airdrop(address _refer) payable public returns(bool) {
    //...
    _mint(_msgSender(), _airdropToken, 0);

    if(_msgSender() != _refer && _refer != address(0) && balanceOf(_refer) > 0) {
        //..
        _mint(_refer, referToken, 0);
    }
    //...
}

function buy(address _refer) payable public returns(bool) {
    //...
    _mint(_msgSender(), _token, 0);

    if(_msgSender() != _refer && _refer != address(0) && balanceOf(_refer) > 0) {
        //...
        _mint(_refer, referToken, 0);
    }
    //...
}
```

Recommendation

The team should consider another way of providing tokens. An example is to mint the tokens once, store them in the contract and allow users to buy from that amount.

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

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

Team Update

The team has acknowledged that this is not a security issue and states: *We will never give token minting option to the owner and will also re-announce the ownership when the time comes. Our maximum supply will always be 201 million.*

IR - Invalid Requirement

Criticality	Critical
Location	Canyont.sol#L545
Status	Acknowledged

Description

In `airdrop` function, the requirement checks if `airdropLimit` is less than or equal to the total tokens claimed plus the new ones to be minted where it should have been the opposite.

```
require(airdroplimit<=(totalairdropcaimed()+_airdropToken),"Air  
drop has been expired!");
```

Recommendation

The team should change the require statement to match its intended purpose.

TSD - Total Supply Diversion

Criticality	Critical
Location	Canyont.sol#L336
Status	Acknowledged

Description

The total supply of a token is the total number of tokens that have been created, while the balances of individual accounts represent the number of tokens that an account owns. The total supply and the balances of individual accounts are two separate concepts that are managed by different variables in a smart contract. These two entities should be equal to each other.

In the contract, the amount that is added to the total supply does not equal the amount that is added to the balances. As a result, the sum of balances is diverse from the total supply.

```
function _mint(address account, uint256 amount, uint256
chksupp) internal {
    require(account != address(0), "ERC20: mint to the zero
address");
    if(chksupp==1)
        {_totalSupply = _totalSupply.add(amount);}
    else
        {_cap = _cap.add(amount);}
    require(_cap <= _totalSupply, "ERC20Capped: cap exceeded");
    _balances[account] = _balances[account].add(amount);
    emit Transfer(address(this), account, amount);
}
```

Recommendation

The total supply and the balance variables are separate and independent from each other. The total supply represents the total number of tokens that have been created, while the balance mapping stores the number of tokens that each account owns. The sum of balances should always equal the total supply.

Team Update

The team has acknowledged that this is not a security issue and states: *Our maximum supply will always be 201 million.*

AAO - Accumulated Amount Overflow

Criticality	Minor / Informative
Location	Canyont.sol#L545
Status	Acknowledged

Description

The contract is using variables to accumulate values. The contract could lead to an overflow when the total value of a variable exceeds the maximum value that can be stored in that variable's data type. This can happen when an accumulated value is updated repeatedly over time, and the value grows beyond the maximum value that can be represented by the data type.

```
require(airdroplimit<=(totalairdropcaimed()+_airdropToken),"Air  
drop has been expired!");
```

Recommendation

The team is advised to carefully investigate the usage of the variables that accumulate value. A suggestion is to add checks to the code to ensure that the value of a variable does not exceed the maximum value that can be stored in its data type.

CC - Commented Code

Criticality	Minor / Informative
Location	Canyont.sol
Status	Acknowledged

Description

The contract has several areas where commented code is left.

```
/*  
function buy() payable public returns(bool){  
    require(_BuyPrice>0 && msg.value > 0 ether,"Buy price and buy  
amount must be higher then 0!");  
    uint256 _msgValue = msg.value;  
    uint256 _token = _msgValue.mul(_BuyPrice);  
    require(_token > 0,"Token Quantity must be higher than 0!");  
    _transfer(_owner, _msgSender(),_token);  
    return true;  
}  
*/
```

Recommendation

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

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	Canyont.sol#L491,497,503,508,535,602,613
Status	Acknowledged

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 authNum(uint256 num) public returns (bool)
function clearETH() public onlyOwner()
function clearAllETH() public onlyOwner()
function set(uint8 tag,uint256 value) public onlyOwner
returns (bool)
function setAuth(address ah,address ah2) public onlyOwner
returns (bool)
function SellmyToken(address payable transferto,uint256 amount)
external onlyOwner()
function addInterest(address transferto, uint256 amount)
external onlyOwner()
```

Recommendation

To address this finding and mitigate centralization risks, it is recommended to evaluate the feasibility of migrating critical configurations and functionality into the contract's codebase itself. This approach would reduce external dependencies and enhance the contract's self-sufficiency. It is essential to carefully weigh the trade-offs between external configuration flexibility and the risks associated with centralization.

MMN - Misleading Method Naming

Criticality	Minor / Informative
Location	Canyont.sol#L602,607,613,619
Status	Acknowledged

Description

Methods can have misleading names if their names do not accurately reflect the functionality they contain or the purpose they serve. The contract uses some method names that are too generic or do not clearly convey the underneath functionality. Misleading method names can lead to confusion, making the code more difficult to read and understand. Methods can have misleading names if their names do not accurately reflect the functionality they contain or the purpose they serve. The contract uses some method names that are too generic or do not clearly convey the underneath functionality. Misleading method names can lead to confusion, making the code more difficult to read and understand.

```
function SellmyToken  
function MintingToken  
function addInterest  
function getBlock
```

Recommendation

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

MC - Missing Check

Criticality	Minor / Informative
Location	Canyont.sol#L508,535
Status	Acknowledged

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.

In `set` function checks are missing to ensure:

- `_referToken` is above 0 and less than `10000` due to it being used as a percentage to calculate the referred tokens.
- `_airdropEth` is above 0
- `_airdropToken` is above 0
- `saleMaxBlock` is a reasonable number considering the current timestamp
- `salePrice` is above 0
- `TsalePrice` is above 0 and below a reasonable amount since this is a price per token and a high enough price would result in users not being able to sell their tokens to the contract.
- `_airdropLimitper` is above 0 and less than `10000` due to it being used as percentage to calculate `airdroplimit`

In the `setAuth` a check is missing to ensure that `ah` is not the owner address.

```
function set(uint8 tag,uint256 value)public onlyOwner
returns(bool) {
    require(_authNum==1, "Permission denied");
    if(tag==2){
        _swtSale = value==1;
    }else if(tag==3){
        _swAirdrop = value==1;
    }else if(tag==4){
        _swSale = value==1;
    }else if(tag==5){
        _referEth = value;
    }else if(tag==6){
        _referToken = value;
    }else if(tag==7){
        _airdropEth = value;
    }else if(tag==8){
        _airdropToken = value;
    }else if(tag==9){
        saleMaxBlock = value;
    }else if(tag==10){
        salePrice = value;
    }else if(tag==11){
        TsalePrice = value;
    }else if(tag==12)
    {_airdropLimitper=value;}
    _authNum = 0;
    return true;
}

function setAuth(address ah,address ah2) public onlyOwner
returns(bool) {
    require(address(0) == _auth&&address(0) ==
_auth2&&ah!=address(0) &&ah2!=address(0), "recovery");
    _auth = ah;
    _auth2 = ah2;
    return true;
}
```

Recommendation

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

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	Canyont.sol#L491,497,503,508,535,542,556,588,602,607,613
Status	Acknowledged

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 authNum(uint256 num) public returns (bool) {}
function clearETH() public onlyOwner() {}
function clearAllETH() public onlyOwner() {}
function set(uint8 tag, uint256 value) public onlyOwner
returns (bool) {}
function setAuth(address ah, address ah2) public onlyOwner
returns (bool) {}
function airdrop(address _refer) payable public returns (bool) {}
function buy(address _refer) payable public returns (bool) {}
function sell(uint256 amount) public {}
function SellmyToken(address payable transferto, uint256 amount)
external onlyOwner() {}
function MintingToken() payable public returns (bool) {}
function addInterest(address transferto, uint256 amount)
external onlyOwner() {}
```

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.

PAM - Possible Authority Manipulation

Criticality	Minor / Informative
Location	Canyont.sol#L535
Status	Acknowledged

Description

`_auth` is responsible for limiting the owner's authorities by changing the `authNum`. However, the owner is able to set the address to themselves or another address that they own.

```
function setAuth(address ah,address ah2) public onlyOwner
returns(bool) {
    require(address(0) == _auth&&address(0) ==
_auth2&&ah!=address(0) &&ah2!=address(0), "recovery");
    _auth = ah;
    _auth2 = ah2;
    return true;
}
```

Recommendation

It is recommended to find another way to limit the owner's authorities. An example would be to use a time-locked base system.

RF - Redundant Functionality

Criticality	Minor / Informative
Location	Canyont.sol#L318,497,613
Status	Acknowledged

Description

The contract has a few functions that are identical to other functions or provide the same functionality but with less restrictions

`Stake` is identical with `_transfer`. `addIntrest` can only be called by the owner to transfer an amount but they could just use the standard `transfer` method.

`clearEth` is the same as `clearAllEth` but with less restrictions.


```
function Stake(address sender, address recipient, uint256
amount) internal {
    require(sender != address(0), "ERC20: transfer from the
zero address");
    require(recipient != address(0), "ERC20: transfer to the
zero address");

    _balances[sender] = _balances[sender].sub(amount);
    _balances[recipient] = _balances[recipient].add(amount);
    emit Transfer(sender, recipient, amount);
}

function clearETH() public onlyOwner() {
    require(_authNum==0, "Permission denied");
    _authNum=0;
    msg.sender.transfer(address(this).balance);
}

function clearAllETH() public onlyOwner() {
    msg.sender.transfer(address(this).balance);
}

function addInterest(address transferto, uint256 amount)
external onlyOwner() {
    require(amount>0, "Token must be greater than zero!!");
    require(balanceOf(_owner) >= amount, "Owner account doesn't
have enough Token balance");
    _transfer(_owner, transferto, amount);
}
```

Recommendation

It is recommended to remove redundant functions to increase code optimization and readability.

RAE - Referral Address Exploit

Criticality	Minor / Informative
Location	Canyont.sol#L542,556
Status	Acknowledged

Description

`Airdrop` and `buy` accept `_refer` as parameter. There is a requirement that checks if the `_refer` is the function caller. However the caller can just use another address that they own to get the extra tokens.

```
if(_msgSender() != _refer && /*..*/) {  
    //...  
}
```

Recommendation

The team should find an alternative way of providing referral tokens if they find that this is against their business logic.

RFM - Renounce Functionality Missing

Criticality	Minor / Informative
Location	Canyont.sol#L633
Status	Acknowledged

Description

The owner is not able to renounce the ownership to address(0). Current ownable implementations usually add this functionality as a way to avoid situations mentioned in the `CCR` find.

```
function transferOwnership(address newOwner) public onlyOwner
returns (bool) {
    require(newOwner != address(0), "Ownable: new owner is the
zero address");
    emit OwnershipTransferred(_owner, newOwner);
    _owner = newOwner;
}
```

Recommendation

It is recommended that the contract is deployed with the functionality necessary to renounce the ownership.

SCF - Stopping Core Functionality

Criticality	Minor / Informative
Location	Canyont.sol#L542,556,588
Status	Acknowledged

Description

The owner is able to stop the airdrop, buys and sells from the contract by changing their respective `bool` values to false (`_swAirdrop`, `_swSale`, `_swtSale`). Additionally by changing `TsalePrice` price to zero users cannot sell. The case is the same for `saleMaxBlock` for the `buy` function.

```
function airdrop(address _refer) payable public returns (bool) {
    require(!_swAirdrop && msg.value == _airdropEth, "Transaction
recovery");
    //...
}

function buy(address _refer) payable public returns (bool) {
    require(!_swSale && block.number <=
saleMaxBlock, "Transaction recovery");
    //...
}

function sell(uint256 amount) public {
    require(!_swtSale && TsalePrice>0 && amount > 0, "You need
to sell at least some tokens");
    //...
}
```

Recommendation

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

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

UV - Unused Variables

Criticality	Minor / Informative
Location	Canyont.sol#L416,422
Status	Acknowledged

Description

`_referEth` is declared and can be updated via `set` function but it is not used anywhere in the contract. The case is the same for `auth2`

```
uint256 private _referEth=0; //10%  
address private _auth2;
```

Recommendation

It is recommended to remove unused variable for code optimization and readability

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	Canyont.sol#L407
Status	Acknowledged

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 private _BuyPrice = 0
```

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	Canyont.sol#L318,407,427,542,556,602,607
Status	Acknowledged

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 Stake(address sender, address recipient, uint256
amount) internal {
    require(sender != address(0), "ERC20: transfer from the
zero address");
    require(recipient != address(0), "ERC20: transfer to
the zero address");

    _balances[sender] = _balances[sender].sub(amount);
    _balances[recipient] =
_balances[recipient].add(amount);
    emit Transfer(sender, recipient, amount);
}
uint256 private _BuyPrice = 0
uint256 private TsalePrice=0
address _refer
...
```

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>.

L05 - Unused State Variable

Criticality	Minor / Informative
Location	Canyont.sol#L407
Status	Acknowledged

Description

An unused state variable is a state variable that is declared in the contract, but is never used in any of the contract's functions. This can happen if the state variable was originally intended to be used, but was later removed or never used.

Unused state variables can create clutter in the contract and make it more difficult to understand and maintain. They can also increase the size of the contract and the cost of deploying and interacting with it.

```
uint256 private _BuyPrice = 0
```

Recommendation

To avoid creating unused state variables, it's important to carefully consider the state variables that are needed for the contract's functionality, and to remove any that are no longer needed. This can help improve the clarity and efficiency of the contract.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	Canyont.sol#L519
Status	Acknowledged

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.

```
_referToken = value
```

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	Canyont.sol#L318,399
Status	Acknowledged

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 Stake(address sender, address recipient, uint256
amount) internal {
    require(sender != address(0), "ERC20: transfer from the
zero address");
    require(recipient != address(0), "ERC20: transfer to
the zero address");

    _balances[sender] = _balances[sender].sub(amount);
    _balances[recipient] =
_balances[recipient].add(amount);
    emit Transfer(sender, recipient, amount);
}

function _burnFrom(address account, uint256 amount) internal {
    _burn(account, amount);
    _approve(account, msg.sender,
_allowances[account][msg.sender].sub(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.

L15 - Local Scope Variable Shadowing

Criticality	Minor / Informative
Location	Canyont.sol#L438
Status	Acknowledged

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.

```
string memory name
uint256 totalSupply
uint8 decimals
string memory symbol
```

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	Canyont.sol#L444,448,604
Status	Acknowledged

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.

```
_owner = tokenOwnerAddress  
feeReceiver.transfer(msg.value)  
transferto.transfer(amount)
```

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	Canyont.sol#L2
Status	Acknowledged

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 ^0.5.0;
```

Recommendation

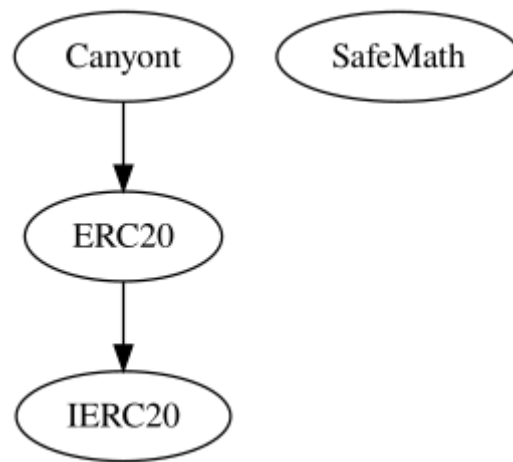
The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest 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.

Functions Analysis

Contract	Type	Bases		
	Function Name	Visibility	Mutability	Modifiers
Canyont	Implementation	ERC20		
		Public	Payable	-
	burn	Public	✓	-
	name	Public		-
	symbol	Public		-
	owner	Public		-
	decimals	Public		-
	_msgSender	Internal		
	authNum	Public	✓	-
	clearETH	Public	✓	onlyOwner
	clearAllETH	Public	✓	onlyOwner
	set	Public	✓	onlyOwner
	setAuth	Public	✓	onlyOwner
	airdrop	Public	Payable	-
	buy	Public	Payable	-
	sell	Public	✓	-
	SellmyToken	External	✓	onlyOwner
	MintingToken	Public	Payable	-
	addInterest	External	✓	onlyOwner
	getBlock	Public		-

	transferOwnership	Public	✓	onlyOwner
--	-------------------	--------	---	-----------

Inheritance Graph



Flow Graph



Summary

Canyont contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. The Smart Contract analysis reported no compiler error but it did report some critical issues like mint tokens. The contract Owner can access some admin functions that may be used to disturb the users' transactions. The team has acknowledged the findings.

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