Cyberscope

Audit Report

DEX RAIDEN

October 2023

Network	BSC
Address	0x5fb7097e62979907b4e71a5989cf695d635890c0
Audited by	© cyberscope

Analysis

Critical Medium Minor / Informative Pass

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

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
	EIS	Excessively Integer Size	Unresolved
	RRS	Redundant Require Statement	Unresolved
	RSW	Redundant Storage Writes	Unresolved
	MFU	Misleading Function Usages	Unresolved
	MEM	Misleading Error Messages	Unresolved
	TUU	Time Units Usage	Unresolved
	MCC	Misleading Commented Code	Unresolved
	AOI	Arithmetic Operations Inconsistency	Unresolved
	RSD	Redundant Swap Duplication	Unresolved
	PVC	Price Volatility Concern	Unresolved
	MCM	Misleading Comment Messages	Unresolved
	RSML	Redundant SafeMath Library	Unresolved
	RSK	Redundant Storage Keyword	Unresolved
	L02	State Variables could be Declared Constant	Unresolved

●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L05	Unused State Variable	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L15	Local Scope Variable Shadowing	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L19	Stable Compiler Version	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

Table of Contents

Review

Contract Name	DexRaiden
Compiler Version	v0.8.4+commit.c7e474f2
Optimization	100 runs
Explorer	https://bscscan.com/address/0x5fb7097e62979907b4e71a5989cf695d635890c0
Address	0x5fb7097e62979907b4e71a5989cf695d635890c0
Network	BSC
Symbol	DXR
Decimals	18
Total Supply	50,000,000

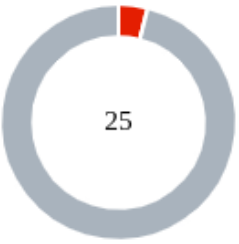
Audit Updates

Initial Audit	20 Oct 2023 https://github.com/cyberscope-io/audits/blob/main/3-dxr/v1/audit.pdf
Corrected Phase 2	27 Oct 2023

Source Files

Filename	SHA256
DexRaiden.sol	5920a6251a7fdcd70c8b3f73436375bae5773659efcb9957fe49a3d21037fcce

Findings Breakdown



Critical	1
Medium	0
Minor / Informative	24

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	1	0	0	0
Medium	0	0	0	0
Minor / Informative	24	0	0	0

BC - Blacklists Addresses

Criticality	Critical
Location	DexRaiden.sol#L736
Status	Unresolved

Description

The contract owner has the authority to stop addresses from transactions. The owner may take advantage of it by calling the `blacklistAddress` function.

```
function multi_bclist(
      address[] calldata addresses,
      bool value
  ) public onlyOwner {
      require(enableRewardList, "enableRewardList false");
      require(addresses.length < 201);
      for (uint256 i; i < addresses.length; ++i) {
          _rewardList[addresses[i]] = value;
      }
  }
```

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.

EIS - Excessively Integer Size

Criticality	Minor / Informative
Location	DexRaiden.sol#L810,817,783
Status	Unresolved

Description

The contract is using a bigger unsigned integer data type that the maximum size that is required. By using an unsigned integer data type larger than necessary, the smart contract consumes more storage space and requires additional computational resources for calculations and operations involving these variables. This can result in higher transaction costs, longer execution times, and potential scalability bottlenecks.

The following are some examples that the contract is using unecessarily more space. The `newValue` could be uint8, the `newValue` could be uint16, `isReward` uint8 etc.

```
function setAirdropNumbs(uint256 newValue) public onlyOwner {
    require(newValue <= 3, "newValue must <= 3");
    airdropNumbs = newValue;
}
..
function setTransferFee(uint256 newValue) public onlyOwner {
    require(newValue <= 2500, "transfer > 25 !");
    transferFee = newValue;
}
..
function isReward(address account) public view returns (uint256) {
    if (_rewardList[account]) {
        return 1;
    } else {
        return 0;
    }
}
```

Recommendation

To address the inefficiency associated with using an oversized unsigned integer data type, it is recommended to accurately determine the required size based on the range of values the variable needs to represent.

RRS - Redundant Require Statement

Criticality	Minor / Informative
Location	DexRaiden.sol#L21
Status	Unresolved

Description

The contract utilizes a `require` statement within the `add` function aiming to prevent overflow errors. This function is designed based on the SafeMath library's principles. In Solidity version 0.8.0 and later, arithmetic operations revert on overflow and underflow, making the overflow check within the function redundant. This redundancy could lead to extra gas costs and increased complexity without providing additional security.

```
function add(uint256 a, uint256 b) internal pure returns (uint256)
{
      uint256 c = a + b;
      require(c >= a, "SafeMath: addition overflow");
      return c;
}
```

Recommendation

It is recommended to remove the `require` statement from the add function since the contract is using a Solidity pragma version equal to or greater than 0.8.0. By doing so, the contract will leverage the built-in overflow and underflow checks provided by the Solidity language itself, simplifying the code and reducing gas consumption. This change will uphold the contract's integrity in handling arithmetic operations while optimizing for efficiency and cost-effectiveness.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	DexRaiden.sol#L691,736
Status	Unresolved

Description

The contract modifies the state of the following variables without checking if their current value is the same as the one given as an argument. As a result, the contract performs redundant storage writes, when the provided parameter matches the current state of the variables, leading to unnecessary gas consumption and inefficiencies in contract execution.

```
    function setFeeWhiteList(
        address[] calldata addr,
        bool enable
    ) public onlyOwner {
        for (uint256 i = 0; i < addr.length; i++) {
            _feeWhiteList[addr[i]] = enable;
        }
    }

    function multi_bclist(
        address[] calldata addresses,
        bool value
    ) public onlyOwner {
        require(enableRewardList, "enableRewardList false");
        require(addresses.length < 201);
        for (uint256 i; i < addresses.length; ++i) {
            _rewardList[addresses[i]] = value;
        }
    }
```

Recommendation

The team is advised to implement additional checks within to prevent redundant storage writes when the provided argument matches the current state of the variables. By incorporating statements to compare the new values with the existing values before

proceeding with any state modification, the contract can avoid unnecessary storage operations, thereby optimizing gas usage.

MFU - Misleading Function Usages

Criticality	Minor / Informative
Location	DexRaiden.sol#L783,822
Status	Unresolved

Description

The contract uses the `isReward` function that checks if a given address is present in the `_rewardList` mapping. However, the naming convention of both the function and the `_rewardList` variable is misleading. Contrary to what their names suggest, there is no functionality related to rewards. Instead, the primary purpose is to preventing certain addresses from transferring tokens, which is associated with blacklisting functionality. Furthermore, the `isReward` function is redundant since the `_transfer` function can directly access the `_rewardList` variable without the need for an intermediary function.

```
   function isReward(address account) public view returns (uint256)
{
        if (_rewardList[account]) {
            return 1;
        } else {
            return 0;
        }
    }
  function _transfer(
        address from,
        address to,
        uint256 amount
    ) internal override {
        require(from != address(0), "ERC20: transfer from the zero
address");
        require(to != address(0), "ERC20: transfer to the zero
address");
        require(isReward(from) <= 0, "isReward > 0 !");
        ...
    }
```

Recommendation

It is recommended to remove the `isReward` function and directly use the `_rewardList` variable for any necessary checks. Additionally, to avoid confusion and enhance code clarity, it would be beneficial to rename the `_rewardList` variable to a more descriptive name, such as `_blacklist`. This will ensure that the variable's name accurately reflects its actual functionality, which is to blacklist or block certain addresses from performing token transfers.

MEM - Misleading Error Messages

Criticality	Minor / Informative
Location	DexRaiden.sol#L741
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.

```
require(addresses.length < 201);
```

Recommendation

The team is suggested to provide a descriptive message to the errors. 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.

TUU - Time Units Usage

Criticality	Minor / Informative
Location	DexRaiden.sol#L1332
Status	Unresolved

Description

The contract is using arbitrary numbers to form time-related values. As a result, it decreases the readability of the codebase and prevents the compiler to optimize the source code.

```
claimWait = 600;
```

Recommendation

It is a good practice to use the time units reserved keywords like `seconds`, `minutes`, `hours`, `days` and `weeks` to process time-related calculations.

It's important to note that these time units are simply a shorthand notation for representing time in seconds, and do not have any effect on the actual passage of time or the execution of the contract. The time units are simply a convenience for expressing time in a more human-readable form.

MCC - Misleading Commented Code

Criticality	Minor / Informative
Location	DexRaiden.sol#L663,857
Status	Unresolved

Description

The contract contains the `_transfer` function, which contains segments of code that are commented out. These commented-out sections do not offer any descriptive or documentation value, making them redundant. Moreover, such segments can be misleading as they might assume that the commented code holds some significance or was part of a previous implementation. Keeping such code can also clutter the contract, making it less readable and potentially confusing.

```
   // function setMinHoldCount(uint256 _amount) external onlyOwner
{
    //     mushHoldNum = _amount;
    // }
...
    function _transfer(
        address from,
        address to,
        uint256 amount
    ) internal override {
        ...
            uint256 marketingTokens = numTokensSellToFund
//contractTokenBalance
                .mul(buy_marketingFee + sell_marketingFee)
                .div(buy_totalFees + sell_totalFees);
            if (marketingTokens > 0)
swapAndSendToFee(marketingTokens);

            uint256 swapTokens = numTokensSellToFund
//contractTokenBalance
                .mul(buy_liquidityFee + sell_liquidityFee)
                .div(buy_totalFees + sell_totalFees);
            if (swapTokens > 0) swapAndLiquify(swapTokens);

            uint256 sellTokens = numTokensSellToFund -
                marketingTokens -
                swapTokens; //balanceOf(address(this));
            if (sellTokens > 0) swapAndSendDividends(sellTokens);

            swapping = false;
        }

        bool takeFee = !swapping;

        // if (enableTransferFee) {
        //     if (
        //         !_swapPairList[from] &&
        //         !_swapPairList[to] &&
        //         !_feeWhiteList[from] &&
        //         !_feeWhiteList[to]
        //     ) {
        //         if (takeFee != false) takeFee = true;
        //     }
        // } else {
        //     if (
        //         !_swapPairList[from] &&
        //         !_swapPairList[to] &&
        //         !_feeWhiteList[from] &&
        //         !_feeWhiteList[to]
        //     ) {
```

```
        //          takeFee = false;
        //      }
        // }
    ...
}
```

Recommendation

It is recommended to remove the unused code segments from the contract instead of merely commenting them out. Cleaning up the contract by eliminating these sections will enhance its clarity, reduce potential points of confusion, and ensure that the codebase remains concise and focused on its actual functionality. This will also make future reviews or audits of the contract more straightforward and less prone to misunderstandings.

AOI - Arithmetic Operations Inconsistency

Criticality	Minor / Informative
Location	DexRaiden.sol#L867
Status	Unresolved

Description

The contract uses both the SafeMath library and native arithmetic operations. The SafeMath library is commonly used to mitigate vulnerabilities related to integer overflow and underflow issues. However, it was observed that the contract also employs native arithmetic operators (such as +, -, *, /) in certain sections of the code.

The combination of SafeMath library and native arithmetic operations can introduce inconsistencies and undermine the intended safety measures. This discrepancy creates an inconsistency in the contract's arithmetic operations, increasing the risk of unintended consequences such as inconsistency in error handling, or unexpected behavior.

```
    uint256 swapTokens = numTokensSellToFund
         .mul(buy_liquidityFee + sell_liquidityFee)
         .div(buy_totalFees + sell_totalFees);
     if (swapTokens > 0) swapAndLiquify(swapTokens);

     uint256 sellTokens = numTokensSellToFund -
         marketingTokens -
         swapTokens;
```

Recommendation

To address this finding and ensure consistency in arithmetic operations, it is recommended to standardize the usage of arithmetic operations throughout the contract. The contract should be modified to either exclusively use SafeMath library functions or entirely rely on native arithmetic operations, depending on the specific requirements and design considerations. This consistency will help maintain the contract's integrity and mitigate potential vulnerabilities arising from inconsistent arithmetic operations.

RSD - Redundant Swap Duplication

Criticality	Minor / Informative
Location	DexRaiden.sol#L860
Status	Unresolved

Description

The contract contains multiple swap methods that individually perform token swaps and transfer promotional amounts to specific addresses and features. This redundant duplication of code introduces unnecessary complexity and increases dramatically the gas consumption. By consolidating these operations into a single swap method, the contract can achieve better code readability, reduce gas costs, and improve overall efficiency.

```
    if (marketingTokens > 0) swapAndSendToFee(marketingTokens);

    uint256 swapTokens = numTokensSellToFund //contractTokenBalance
        .mul(buy_liquidityFee + sell_liquidityFee)
        .div(buy_totalFees + sell_totalFees);
    if (swapTokens > 0) swapAndLiquify(swapTokens);

    uint256 sellTokens = numTokensSellToFund -
        marketingTokens -
        swapTokens; //balanceOf(address(this));
    if (sellTokens > 0) swapAndSendDividends(sellTokens);
```

Recommendation

A more optimized approach could be adopted to perform the token swap operation once for the total amount of tokens and distribute the proportional amounts to the corresponding addresses, eliminating the need for separate swaps.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	DexRaiden.sol#L641,843
Status	Unresolved

Description

The contract accumulates tokens from the taxes to swap them for ETH. The variable `swapTokensAtAmount` sets a threshold where the contract will trigger the swap functionality. If the variable is set to a big number, then the contract will swap a huge amount of tokens for ETH.

It is important to note that the price of the token representing it, can be highly volatile. This means that the value of a price volatility swap involving Ether could fluctuate significantly at the triggered point, potentially leading to significant price volatility for the parties involved.

```
    function setSwapTokensAtAmount(uint256 newValue) public onlyOwner {
        swapTokensAtAmount = newValue;
    }
...
    uint256 contractTokenBalance = balanceOf(address(this));

    bool canSwap = contractTokenBalance >= swapTokensAtAmount;

    uint256 numTokensSellToFund = amount
    if (numTokensSellToFund > contractTokenBalance) {
        numTokensSellToFund = contractTokenBalance;
    }
```

Recommendation

The contract could ensure that it will not sell more than a reasonable amount of tokens in a single transaction. A suggested implementation could check that the maximum amount should be less than a fixed percentage of the exchange reserves. Hence, the contract will guarantee that it cannot accumulate a huge amount of tokens in order to sell them.

MCM - Misleading Comment Messages

Criticality	Minor / Informative
Location	DexRaiden.sol#L668
Status	Unresolved

Description

The contract is using misleading comment messages. These comment messages do not accurately reflect the actual implementation, making it difficult to understand the source code. As a result, the users will not comprehend the source code's actual implementation.

```
    function disableSwapLimit() public onlyOwner {
        // 关闭限购，且无法重启
        enableSwapLimit = false;
    }
```

Recommendation

The team is advised to carefully review the comment in order to reflect the actual implementation. To improve code readability, the team should use more specific and descriptive comment messages.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	DexRaiden.sol
Status	Unresolved

Description

SafeMath is a popular Solidity library that provides a set of functions for performing common arithmetic operations in a way that is resistant to integer overflows and underflows.

Starting with Solidity versions that are greater than or equal to 0.8.0, the arithmetic operations revert to underflow and overflow. As a result, the native functionality of the Solidity operations replaces the SafeMath library. Hence, the usage of the SafeMath library adds complexity, overhead and increases gas consumption unnecessarily.

```
library SafeMath {...}
```

Recommendation

The team is advised to remove the SafeMath library. Since the version of the contract is greater than `0.8.0` then the pure Solidity arithmetic operations produce the same result.

If the previous functionality is required, then the contract could exploit the `unchecked { ... }` statement.

Read more about the breaking change on https://docs.soliditylang.org/en/v0.8.16/080-breaking-changes.html#solidity-v0-8-0-breaking-changes.

RSK - Redundant Storage Keyword

Criticality	Minor / Informative
Location	DexRaiden.sol#L1456,1461,1471,1477
Status	Unresolved

Description

The contract uses the `storage` keyword in a view function. The `storage` keyword is used to persist data on the contract's storage. View functions are functions that do not modify the state of the contract and do not perform any actions that cost gas (such as sending a transaction). As a result, the use of the `storage` keyword in view functions is redundant.

```
Map storage map
```

Recommendation

It is generally considered good practice to avoid using the `storage` keyword in view functions because it is unnecessary and can make the code less readable.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	DexRaiden.sol#L491,530
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.

```
address public deadWallet =
0x000000000000000000000000000000000000dEaD
uint256 public gasForProcessing = 300000
```

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	DexRaiden.sol#L438,486,487,493,497,499,500,501,502,503,505,506,507,508,509,519,532,534,648,649,659,736,1045,1046,1047,1161,1163,1225,1230,1236,1242
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 WETH() external pure returns (address);
IUniswapV2Router02 public _swapRouter
address public _mainPair
address public immutable ETH
mapping(address => bool) public _rewardList
uint256 public buy_marketingFee
uint256 public buy_liquidityFee
uint256 public buy_ETHRewardsFee
uint256 public buy_totalFees
uint256 public buy_burnFee
uint256 public sell_marketingFee
uint256 public sell_liquidityFee
uint256 public sell_ETHRewardsFee
uint256 public sell_totalFees

...
```

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.

L05 - Unused State Variable

Criticality	Minor / Informative
Location	DexRaiden.sol#L1518
Status	Unresolved

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.

```
stant MAX_INT256 = ~(int256(1) << 255);
```

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	DexRaiden.sol#L642,651,660,688,706,812,819
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.

```
swapTokensAtAmount = newValue
maxBuyAmount = _buyamount
maxWalletAmount = _amount
kb = killBlockNumber;
function completeCustoms(uint256[] calldata customs) external
onlyOwner {
...
}
airdropNumbs = newValue;
transferFee = newValue;
```

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	DexRaiden.sol#L1564
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 abs(int256 a) internal pure returns (int256) {
        require(a != MIN_INT256);
        return a < 0 ? -a : a;
    }
```

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.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	DexRaiden.sol#L840,857,862
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 numTokensSellToFund = (amount *
         (buy_totalFees + sell_totalFees)) / 5000;
    ...
    uint256 marketingTokens = numTokensSellToFund
//contractTokenBalance
         .mul(buy_marketingFee + sell_marketingFee)
         .div(buy_totalFees + sell_totalFees);
    ...
    uint256 swapTokens = numTokensSellToFund //contractTokenBalance
         .mul(buy_liquidityFee + sell_liquidityFee)
         .div(buy_totalFees + sell_totalFees);
```

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.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	DexRaiden.sol#L742,943,996,997,998
Status	Unresolved

Description

Using an uninitialized local variable can lead to unpredictable behavior and potentially cause errors in the contract. It's important to always initialize local variables with appropriate values before using them.

```
uint256 i
uint256 burnAmount;
uint256 iterations,
uint256 claims,
uint256 lastProcessedIndex
```

Recommendation

By initializing local variables before using them, the contract ensures that the functions behave as expected and avoid potential issues.

L15 - Local Scope Variable Shadowing

Criticality	Minor / Informative
Location	DexRaiden.sol#L1173,1174,1225,1230,1236,1242
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.

```
string memory _name,
string memory _symbol,
address _owner
```

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	DexRaiden.sol#L184,621,701
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.

```
_owner = msgSender
_mainPair = __mainPair
fundAddress = wallet;
```

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	DexRaiden.sol#L3
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 ^0.8.4;
```

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	DexRaiden.sol#L1019
Status	Unresolved

Description

According to the ERC20 specification, the transfer methods should be checked if the result is successful. Otherwise, the contract may wrongly assume that the transfer has been established.

```
IERC20(ETH).transfer(fundAddress, newBalance);
```

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**
SafeMath	Library			
	add	Internal		
	sub	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	div	Internal		
	mod	Internal		
	mod	Internal		
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner

	transferOwnership	Public	✓	onlyOwner
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	External		-
	totalSupply	Public		-
	balanceOf	Public		-

	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	
	_beforeTokenTransfer	Internal	✓	
IUniswapV2Router01	Interface			
	factory	External		-
	WETH	External		-
	addLiquidityETH	External	Payable	-
IUniswapV2Router02	Interface	IUniswapV2Router01		
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
IUniswapV2Factory	Interface			

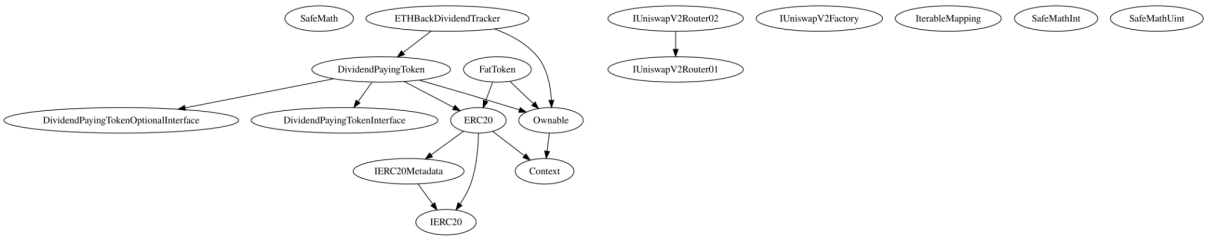
	createPair	External	✓	-
	getPair	External		-
DexRaiden	Implementation	ERC20, Ownable		
		Public	✓	ERC20
	setSwapTokensAtAmount	Public	✓	onlyOwner
		External	Payable	-
	changeSwapLimit	External	✓	onlyOwner
	changeWalletLimit	External	✓	onlyOwner
	disableSwapLimit	Public	✓	onlyOwner
	disableWalletLimit	Public	✓	onlyOwner
	disableChangeTax	Public	✓	onlyOwner
	launch	Public	✓	onlyOwner
	setKillBlock	Public	✓	onlyOwner
	setFeeWhiteList	Public	✓	onlyOwner
	setFundAddress	External	✓	onlyOwner
	completeCustoms	External	✓	onlyOwner
	setSwapPairList	Public	✓	onlyOwner
	multi_bclist	Public	✓	onlyOwner
	_setAutomatedMarketMakerPair	Private	✓	
	excludeFromDividends	External	✓	onlyOwner
	processDividendTracker	External	✓	-
	claim	External	✓	-

	isReward	Public		-
	setSwapAndLiquifyEnabled	Public	✓	onlyOwner
	setEnableTransferFee	Public	✓	onlyOwner
	setAirdropNumbs	Public	✓	onlyOwner
	setTransferFee	Public	✓	onlyOwner
	_transfer	Internal	✓	
	swapAndSendToFee	Private	✓	
	swapAndLiquify	Private	✓	
	swapTokensForEth	Private	✓	
	swapTokensForETH	Private	✓	
	addLiquidity	Private	✓	
	swapAndSendDividends	Private	✓	
DividendPayingTokenOptionalInterface	Interface			
	withdrawableDividendOf	External		-
	withdrawnDividendOf	External		-
	accumulativeDividendOf	External		-
DividendPayingTokenInterface	Interface			
	dividendOf	External		-
	withdrawDividend	External	✓	-

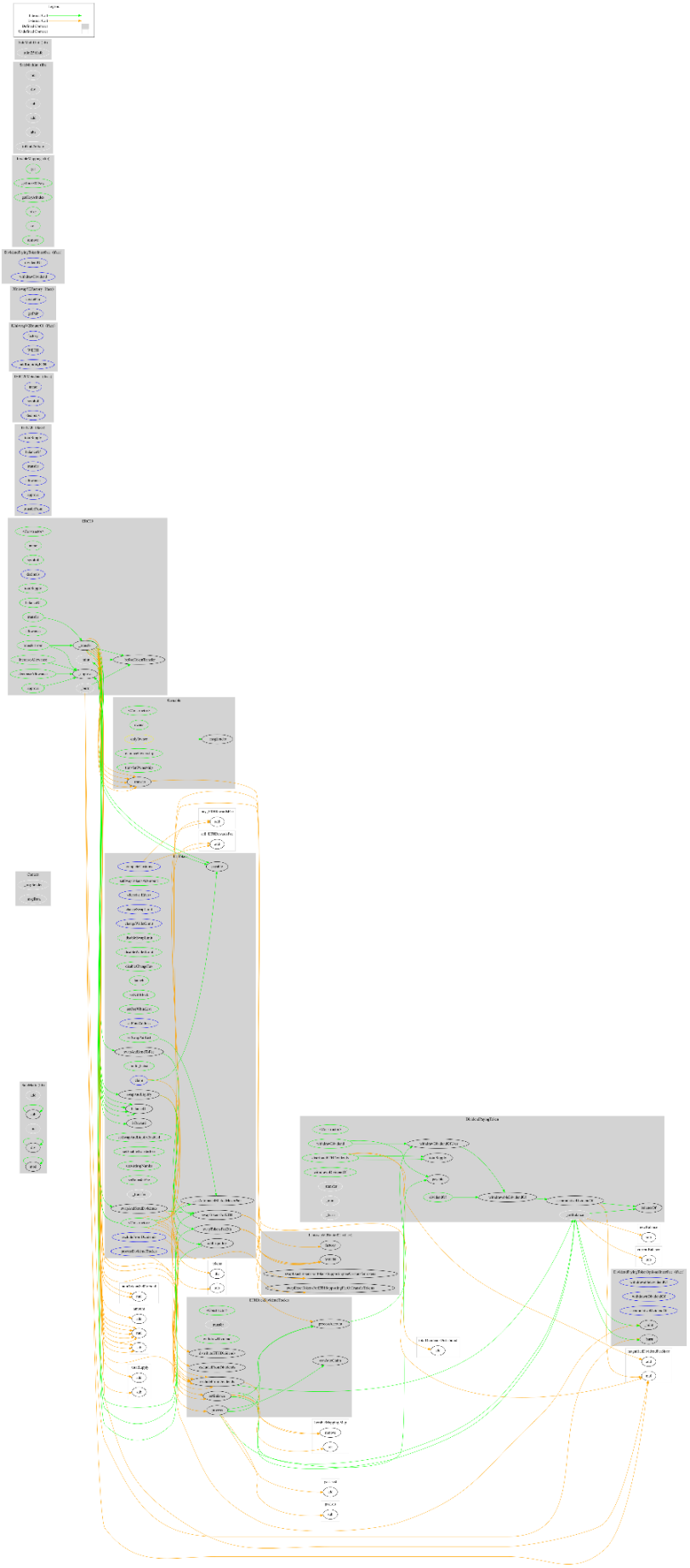
DividendPayingToken	Implementation	ERC20, Ownable, DividendPayingTokenInterface, DividendPayingTokenOptionalInterface		
		Public	✓	ERC20
	distributeETHDividends	Public	✓	onlyOwner
	withdrawDividend	Public	✓	-
	_withdrawDividendOfUser	Internal	✓	
	dividendOf	Public		-
	withdrawableDividendOf	Public		-
	withdrawnDividendOf	Public		-
	accumulativeDividendOf	Public		-
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_setBalance	Internal	✓	
ETHBackDividendTracker	Implementation	Ownable, DividendPayingToken		
		Public	✓	DividendPayingToken
	_transfer	Internal		
	withdrawDividend	Public		-
	excludeFromDividends	External	✓	onlyOwner
	canAutoClaim	Private		

	setBalance	External	✓	onlyOwner
	process	Public	✓	-
	processAccount	Public	✓	onlyOwner
IterableMapping	Library			
	get	Public		-
	getIndexOfKey	Public		-
	getKeyAtIndex	Public		-
	size	Public		-
	set	Public	✓	-
	remove	Public	✓	-
SafeMathInt	Library			
	mul	Internal		
	div	Internal		
	sub	Internal		
	add	Internal		
	abs	Internal		
	toUint256Safe	Internal		
SafeMathUint	Library			
	toInt256Safe	Internal		

Inheritance Graph



Flow Graph



Summary

DEX RAIDEN 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 massively blacklist addresses. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats. There is also a limit of max 25% fees on buy, sell and transfer transactions.

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