Cyberscope

Audit Report

12 Days Of Xmas

December 2023

Network BSC

Address 0xb5c50edd218569d7a19fab89ca6b85ec30bd1eb0

Audited by © cyberscope

Analysis

Critical Medium Minor / Informative Pass

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

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
Medium	ZD	Zero Division	Unresolved
Medium	PTRP	Potential Transfer Revert Propagation	Unresolved
Minor / Informative	UMU	Unoptimized Multiplier Usage	Unresolved
Minor / Informative	FSA	Fixed Swap Address	Unresolved
Minor / Informative	IFV	Inconsistent Fee Visibility	Unresolved
Minor / Informative	RFU	Redundant Function Usage	Unresolved
Minor / Informative	RES	Redundant Event Statement	Unresolved
Minor / Informative	DDP	Decimal Division Precision	Unresolved
Minor / Informative	PVC	Price Volatility Concern	Unresolved
Minor / Informative	RRS	Redundant Require Statement	Unresolved
Minor / Informative	RSML	Redundant SafeMath Library	Unresolved
Minor / Informative	L02	State Variables could be Declared Constant	Unresolved
Minor / Informative	L04	Conformance to Solidity Naming Conventions	Unresolved
Minor / Informative	L07	Missing Events Arithmetic	Unresolved

Table of Contents

Review

Contract Name	Twelve_Days_Of_Xmas
Compiler Version	v0.8.17+commit.8df45f5f
Optimization	200 runs
Explorer	https://bscscan.com/address/0xb5c50edd218569d7a19fab89ca6b85ec30bd1eb0
Address	0xb5c50edd218569d7a19fab89ca6b85ec30bd1eb0
Network	BSC
Symbol	DOX
Decimals	18
Total Supply	100,000,000

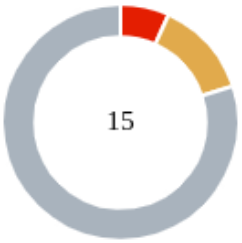
Audit Updates

Initial Audit	07 Dec 2023 https://github.com/cyberscope-io/audits/blob/main/2-dox/v1/audit.pdf
Corrected Phase 2	12 Dec 2023

Source Files

Filename	SHA256
Twelve_Days_Of_Xmas.sol	587c36cb99a1bad1580df8ec156f5265402d9595ca03c4b55dc674551807fbbf

Findings Breakdown



Critical	1
Medium	2
Minor / Informative	12

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	1	0	0	0
Medium	2	0	0	0
Minor / Informative	12	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	contracts/Twelve_Days_Of_Xmas.sol#L268,284,287
Status	Unresolved

Description

The contract is currently designed in a way that allows the contract owner to exert significant control over the sales process. Specifically, the owner has the capability to halt all sales by either setting the total transaction fees to zero which results in the variable `totalBNBFee` being zero, or by designating a contract address as the fee recipient, which could revert to its receive function. This design introduces a critical vulnerability, as it can effectively transform the contract into a honeypot. In such a scenario, users will be able to deposit funds but will be unable to execute sales, leading to a situation where funds are locked within the contract without the possibility of retrieval.

```
uint256 totalBNBFee = totalFee;
...
uint256 amountBNBMarketing = (amountBNB * marketingFee) / totalBNBFee;
uint256 amountBNBDevelopment = (amountBNB * otherFee) / totalBNBFee;
...
payable(marketingFeeReceiver).transfer(amountBNBMarketing);
payable(otherFeeReceiver).transfer(amountBNBDevelopment);
```

Recommendation

It is recommended to implement safeguards within the contract to mitigate this risk. The ability to set a contract address as the fee recipient should be restricted. This can be achieved by adding checks to ensure that the fee recipient is always an externally owned account (EOA) and not a contract. Additionally, the possibility of setting the `totalBNBFee` vairaible to zero should be handled appropriately.Moreover 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 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.

ZD - Zero Division

Criticality	Medium
Location	contracts/Twelve_Days_Of_Xmas.sol#L268,284
Status	Unresolved

Description

The contract uses variables that may be set to zero as denominators. This can lead to unpredictable and potentially harmful results, such as a transaction revert.

Specifically, the `totalBNBFee` variable can be zero if the fees are set to zero.

```
uint256 totalBNBFee = totalFee;
…
uint256 amountBNBMarketing = (amountBNB * marketingFee) /
totalBNBFee;

uint256 amountBNBDevelopment = (amountBNB * otherFee) / totalBNBFee;
```

Recommendation

It is important to handle division by zero appropriately in the code to avoid unintended behavior and to ensure the reliability and safety of the contract. The contract should ensure that the divisor is always non-zero before performing a division operation. It should prevent the variables from being set to zero, or should not allow the execution of the corresponding statements.

PTRP - Potential Transfer Revert Propagation

Criticality	Medium
Location	contracts/Twelve_Days_Of_Xmas.sol#L287
Status	Unresolved

Description

The contract sends funds to the `marketingFeeReceiver` and `otherFeeReceiver` addresses as part of the transfer flow. This address can either be a wallet address or a contract. If the address belongs to a contract then it may revert from incoming payment. As a result, the error will propagate to the token's contract and revert the transfer.

```
payable(marketingFeeReceiver).transfer(amountBNBMarketing);

payable(otherFeeReceiver).transfer(amountBNBDevelopment);
```

Recommendation

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

UMU - Unoptimized Multiplier Usage

Criticality	Minor / Informative
Location	contracts/Twelve_Days_Of_Xmas.sol#L159
Status	Unresolved

Description

The contract is currently implementing three separate multipliers for sell, buy, and transfer transactions, each set to a value of 100. These variables, `sellMultiplier`, `buyMultiplier`, and `transferMultiplier`, are defined as uint256 and are utilized in their respective transaction types. However, throughout the contract's implementation, these multipliers maintain a constant value and do not undergo any changes. This redundancy in defining and using three identical multipliers where their values remain unchanged indicates an area for optimization. The use of multiple variables for the same constant value not only consumes unnecessary storage space but also adds to the complexity of the contract, potentially impacting its efficiency and readability. Simplifying this aspect of the contract could lead to more streamlined code and reduced gas costs, which is a critical factor in smart contract efficiency.

```
uint256 sellMultiplier = 100;

uint256 buyMultiplier = 100;

uint256 transferMultiplier = 100;
```

Recommendation

It is recommended to consolidate the three multipliers into a single variable. Since `sellMultiplier`, `buyMultiplier`, and `transferMultiplier` all hold the same value and function identically, merging them into one variable would streamline the contract's logic and reduce redundancy. This can be achieved by introducing a single `uint256` variable, to replace the three individual multipliers. This change would not only simplify the contract's structure but also potentially lower the gas costs associated with deploying and interacting with the contract.

FSA - Fixed Swap Address

Criticality	Minor / Informative
Location	contracts/Twelve_Days_Of_Xmas.sol#L176
Status	Unresolved

Description

The swap address is assigned once and it can not be changed. It is a common practice in decentralized exchanges to create new swap versions. A contract that cannot change the swap address may not be able to catch up to the upgrade. As a result, the contract will not be able to migrate to a new liquidity pool pair or decentralized exchange.

```
   constructor () Auth(msg.sender) {
        router =
IDEXRouter(0x10ED43C718714eb63d5aA57B78B54704E256024E);
//router address pcs v2

        WBNB = router.WETH();

        pair = IDEXFactory(router.factory()).createPair(WBNB,
address(this));
        ...
        }
```

Recommendation

The team is advised to add the ability to change the pair and router address in order to cover potential liquidity pool migrations. It would be better to support multiple pair addresses so the token will be able to have the same behavior in all the decentralized liquidity pairs.

IFV - Inconsistent Fee Visibility

Criticality	Minor / Informative
Location	contracts/Twelve_Days_Of_Xmas.sol#L154
Status	Unresolved

Description

The contract is utilizing the `marketingFee` and `otherFee` variables to calculate `totalFee`, which represents the fee within the contract. However, there is an inconsistency in the visibility of these fee variables. The `marketingFee` is declared as `public`, making it visible and accessible externally, while `otherFee` is declared as `private`, restricting its visibility to within the contract only. This discrepancy in visibility leads to a lack of transparency, as external entities or users can view only one component of the total fee structure (`marketingFee`), but not the other (`otherFee`). Such inconsistency can cause confusion and hinder the ability of users to fully understand the fee dynamics of the contract.

```
uint256 public marketingFee = 2;

uint256 private otherFee = 2;
```

Recommendation

It is recommended to use consistent visibility for all fee-related variables within the contract. If the intention is to maintain transparency and allow users to view the complete fee structure, both `marketingFee` and `otherFee` should be declared as `public`. Conversely, if the intention is to keep fee details private, then both should be declared as `private`. Aligning the visibility of these variables will ensure clarity and consistency in how fee information is presented to users and external entities, enhancing the contract's transparency and trustworthiness.

RFU - Redundant Function Usage

Criticality	Minor / Informative
Location	contracts/Twelve_Days_Of_Xmas.sol#L301
Status	Unresolved

Description

The contract contains an `update_fees` function solely used for emitting the `UpdateFee` event. This function is called within the `setMultipliers` and `setFees` functions to emit the event after updating fee-related variables. However, the `update_fees` function itself does not perform any additional logic apart from emitting the event. This setup introduces an extra layer of function calls that could be streamlined. Directly emitting the `UpdateFee` event within the `setMultipliers` and `setFees` functions, instead of calling a separate function to do so, would simplify the contract's structure and enhance its readability.

```
    function update_fees() internal {
        emit UpdateFee(
uint8(totalFee.mul(buyMultiplier).div(100)),
            uint8(totalFee.mul(sellMultiplier).div(100)),
            uint8(totalFee.mul(transferMultiplier).div(100))
            );
    }
```

Recommendation

It is recommended to emit the `UpdateFee` event directly within the `setMultipliers` and `setFees` functions, rather than using the intermediary `update_fees` function. This approach reduces the number of function calls, simplifying the contract's logic flow. The direct emission of events in the respective functions where changes occur enhances clarity and makes the contract more straightforward. This modification will not alter the contract's core functionality but will result in a more efficient and cleaner code structure.

RES - Redundant Event Statement

Criticality	Minor / Informative
Location	contracts/Twelve_Days_Of_Xmas.sol#L695
Status	Unresolved

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

The contract contain several `event` statement that are not used in the contract's implemantation.

```
event AutoLiquify(uint256 amountBNB, uint256 amountTokens);
event Wallet_txExempt(address Wallet, bool Status);
event Wallet_holdingExempt(address Wallet, bool Status);
event BalanceClear(uint256 amount);
event clearToken(address TokenAddressCleared, uint256 Amount);
event Set_Wallets_Dev(address DevWallet);
event config_MaxWallet(uint256 maxWallet);
event config_MaxTransaction(uint256 maxWallet);
event config_TradingStatus(bool Status);
event config_LaunchMode(bool Status);
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it. It is recommend removing the unused event statement from the contract..

DDP - Decimal Division Precision

Criticality	Minor / Informative
Location	contracts/Twelve_Days_Of_Xmas.sol#L284
Status	Unresolved

Description

Division of decimal (fixed point) numbers can result in rounding errors due to the way that division is implemented in Solidity. Thus, it may produce issues with precise calculations with decimal numbers.

Solidity represents decimal numbers as integers, with the decimal point implied by the number of decimal places specified in the type (e.g. decimal with 18 decimal places). When a division is performed with decimal numbers, the result is also represented as an integer, with the decimal point implied by the number of decimal places in the type. This can lead to rounding errors, as the result may not be able to be accurately represented as an integer with the specified number of decimal places.

Hence, the splitted shares will not have the exact precision and some funds may not be calculated as expected.

```
uint256 amountBNBMarketing = (amountBNB * marketingFee) /
totalBNBFee;

uint256 amountBNBDevelopment = (amountBNB * otherFee) /
totalBNBFee;
```

Recommendation

The team is advised to take into consideration the rounding results that are produced from the solidity calculations. The contract could calculate the subtraction of the divided funds in the last calculation in order to avoid the division rounding issue.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	contracts/Twelve_Days_Of_Xmas.sol#L266
Status	Unresolved

Description

The contract accumulates tokens from the taxes to swap them for ETH. The variable `swapThreshold` sets a threshold where the contract will trigger the swap functionality. The `swapThreshold` variable can be set up to `10%` of the total supply. As a result 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 swapBack() internal swapping {
    uint256 totalBNBFee = totalFee;
    uint256 amountToSwap = swapThreshold ;

    address[] memory path = new address[](2);

    path[0] = address(this);

    path[1] = WBNB;

    router.swapExactTokensForETHSupportingFeeOnTransferTokens(
    amountToSwap,
    0,
    path,
    address(this),
    block.timestamp
);
```

Recommendation

The contract could ensure that it will not sell more than `2 %` of the total supply 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.

RRS - Redundant Require Statement

Criticality	Minor / Informative
Location	contracts/Twelve_Days_Of_Xmas.sol#L9
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.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	contracts/Twelve_Days_Of_Xmas.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.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	contracts/Twelve_Days_Of_Xmas.sol#L159,160,161
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 sellMultiplier = 100
uint256 buyMultiplier = 100
uint256 transferMultiplier = 100
```

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	Twelve_Days_Of_Xmas.sol#L60,116,136,139,150,293,301,309,317,327,343,344,345,348,350,351,353,354,355,356,357
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.

```
event Authorize_Wallet(address Wallet, bool Status);
function WETH() external pure returns (address);

...
```

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.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	contracts/Twelve_Days_Of_Xmas.sol#L311
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.

```
marketingFee = _marketingFee
```

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.

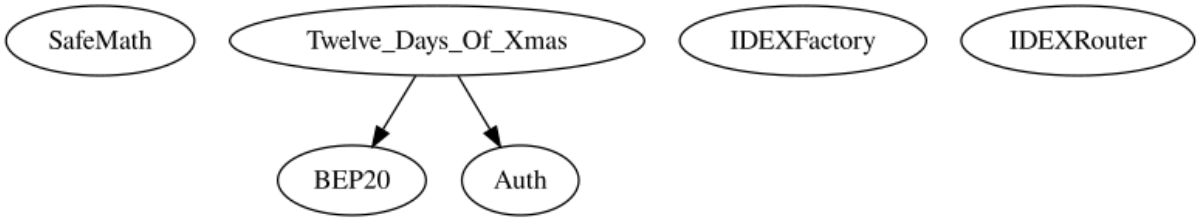
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
SafeMath	Library			
	add	Internal		
	sub	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	div	Internal		
BEP20	Interface			
	getOwner	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
Auth	Implementation			
		Public	✓	-
	isOwner	Public		-

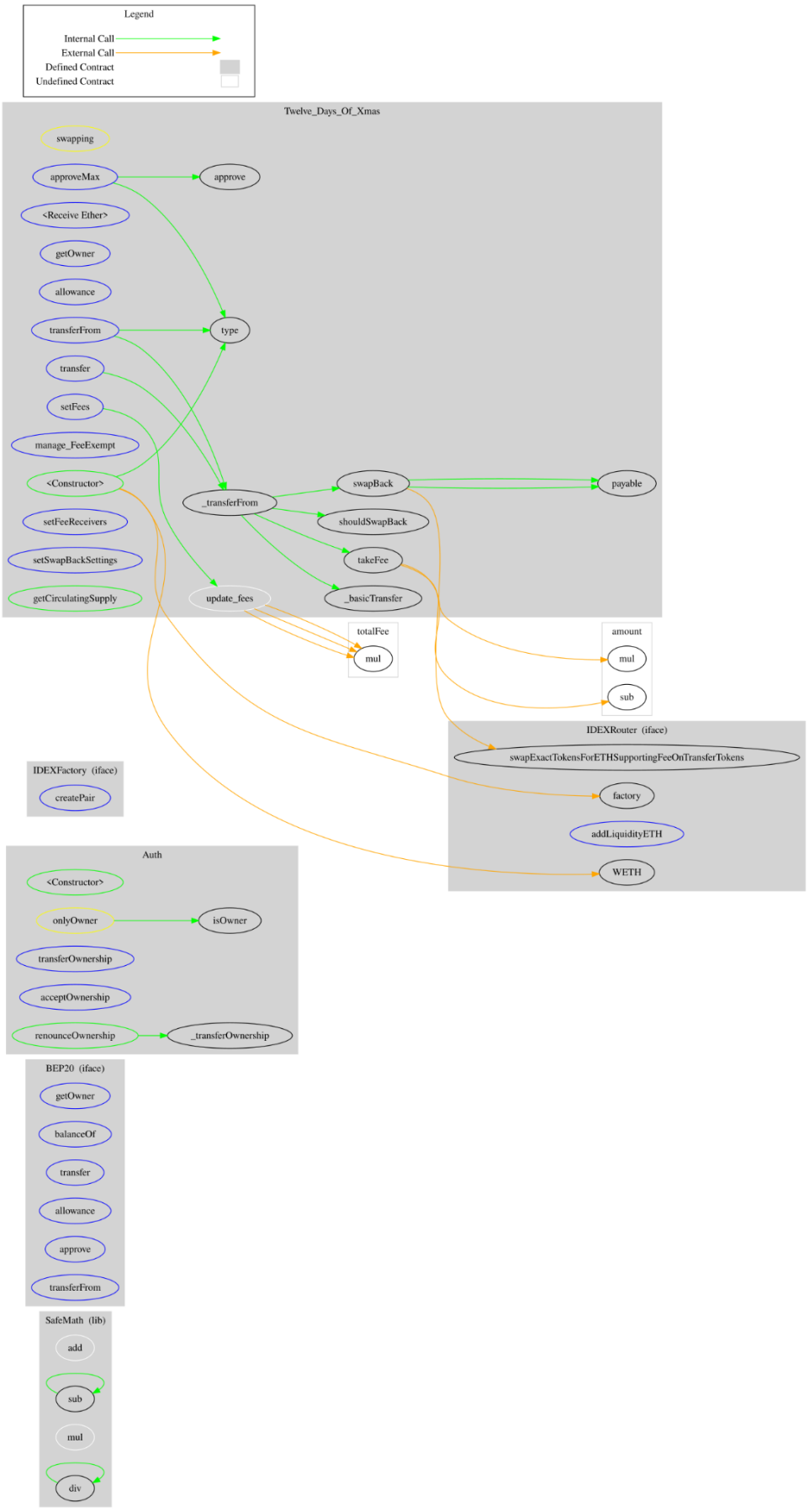
	transferOwnership	External	✓	onlyOwner
	acceptOwnership	External	✓	-
	renounceOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
IDEXFactory	Interface			
	createPair	External	✓	-
IDEXRouter	Interface			
	factory	External		-
	WETH	External		-
	addLiquidityETH	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
Twelve_Days_Of_Xmas	Implementation	BEP20, Auth		
		Public	✓	Auth
		External	Payable	-
	getOwner	External		-
	allowance	External		-
	approve	Public	✓	-
	approveMax	External	✓	-
	transfer	External	✓	-
	transferFrom	External	✓	-

	_transferFrom	Internal	✓	
	_basicTransfer	Internal	✓	
	takeFee	Internal	✓	
	shouldSwapBack	Internal		
	swapBack	Internal	✓	swapping
	manage_FeeExempt	External	✓	onlyOwner
	update_fees	Internal	✓	
	setFees	External	✓	onlyOwner
	setFeeReceivers	External	✓	onlyOwner
	setSwapBackSettings	External	✓	onlyOwner
	getCirculatingSupply	Public		-

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
Twelve_Days_Of_Xmas
swapping
approveMax
approve
<Receive Ether>
getOwner
allowance
transferFrom
type
transfer
setFees
manage_FeeExempt
<Constructor>
_transferFrom
swapBack
payable
shouldSwapBack
setFeeReceivers
takeFee
setSwapBackSettings
getCirculatingSupply
update_fees
_basicTransfer
totalFee
mul
amount
mul
sub
IDEXRouter (iface)
swapExactTokensForETHSupportingFeeOnTransferTokens
IDEXFactory (iface)
createPair
factory
addLiquidityETH
WETH
Auth
<Constructor>
onlyOwner
isOwner
transferOwnership
acceptOwnership
renounceOwnership
_transferOwnership
BEP20 (iface)
getOwner
balanceOf
transfer
allowance
approve
transferFrom
SafeMath (lib)
add
sub
mul
div

Summary

12 Days Of Xmas contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. There are some functions that can be abused by the owner like stop transactions. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats. There is also a limit of max 25% fees.

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