AUDIT

SKYNET

CoinMooner

Find Next Moonshot coins

TABLE OF CONTENTS

I. SUMMARY

II. OVERVIEW

III. FINDINGS

A. CONS-1 | deadWallet can be constant

B. CONS-2 | BUSD should be constant

C. SUPP-1 | totalSupply can be constant

D. BUYB-1 | buyBackWalletAddress can be constant

E. LIQUI-1 | LiquidityWalletUpdated event not used

F. UNIS-1 | _uniswapV2Pair is instantiated and then set to state variable

G. LOOP-1 | For loops initialize the i value to 0

H. WALL-1 | Wallet setters do not check address

I. SWAP-1 | swapAndLiquifyEnabled

J. SALE-1| updateMaxAllowedSalePercentang

K. OPEN-1 | isOpen State Variable Initialization

L. TRADE-1 | stopTrade and openTrade

M. UINT-1 | Initializing uint8

N. CONS-3 | BUSD should be constant

O. WRAPP-1 | BUSD address does not need to be wrapped in address

P. DIVID-1 | WithdrawDividend

Q. TRANS-1 | _transfer function

R. GAS-1 | If statement comparing gasLeft with newGasLeft

IV. GLOBAL SECURITY WARNINGS

V. DISCLAIMER

AUDIT SUMMARY

This report was written for SKYNET (SKYNET) in order to find flaws and vulnerabilities in the SKYNET project's source code, as well as any contract dependencies that weren't part of an officially recognized library.

A comprehensive examination has been performed, utilizing Static Analysis, Manual Review, and SKYNET Deployment techniques. The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors
- Assessing the codebase to ensure compliance with current best practices and industry standards
- Ensuring contract logic meets the specifications and intentions of the client
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders
- Through line-by-line manual review of the entire codebase by industry expert

AUDIT OVERVIEW

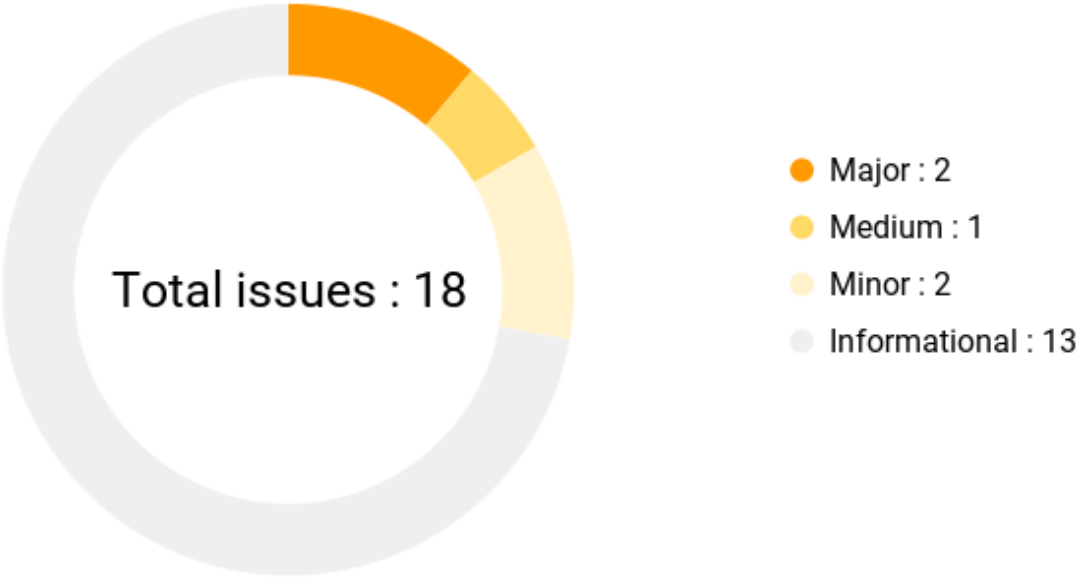
PROJECT SUMMARY

Project name	SKYNEY
Description	SkyNet is a new crypto investment and business solution offering safe investments for traders and business solutions for token owners.
Platform	BNB Smart Chain
Language	Solidity
Codebase	https://bscscan.com/token/0x6f0f9550154c1db781bb20af4ce6075efe331fd9

FINDINGS SUMMARY

Vulnerability	Total
Critical	0
Major	2
Medium	1
Minor	2
Informational	13

AUDIT FINDINGS



Code	Title	Severity
CONS-1	deadWallet can be constant	Informational
CONS-2	BUSD should be constant	Informational
SUPP-1	totalSupply can be constant	Informational
BUYB-1	buyBackWalletAddress can be constant	Informational
LIQUI-1	LiquidityWalletUpdated event not used	Minor
UNIS-1	_uniswapV2Pair is instantiated and then set to state variable	Informational
LOOP-1	For loops initialize the i value to 0	Informational
WALL-1	Wallet setters do not check address	Medium

SWAP-1	swapAndLiquifyEnabled	Informational
SALE-1	updateMaxAllowedSalePercentang	Major
OPEN-1	isOpen State Variable Initialization	Informational
TRADE-1	stopTrade and openTrade	Informational
UINT-1	Initializing uint8	Informational
CONS-3	BUSD should be constant	Informational
WRAPP-1	BUSD address does not need to be wrapped in address	Informational
DIVID-1	WithdrawDividend	Informational
TRANS-1	_transfer function	Major
GAS-1	If statement comparing gasLeft with newGasLeft	Minor

CONS-1 | deadWallet can be constant

Description

If the value for deadWallet will not change then it can be set to a constant, which will save gas when it is used for operations.

Recommendation

Change deadWallet state variable to a constant:

address public constant deadWallet = <address>

CONS-2 | BUSD should be constant

Description

The value for the state variable BUSD is currently set to immutable whereas it should be set to constant. The difference is that constant is used for variables that can never be changed after compilation whereas immutable is used for state variables that can be set within the constructor.

Recommendation

Change BUSD state variable to a constant:

address public constant BUSD = <address>

SUPP-1 | totalSupply can be constant

Description

If the value for totalSupply will not change then it can be set to a constant, which will save gas when it is used for operations. In addition, if totalSupply is supposed to be used to cap the supply then it may be better practice to name it max supply

Recommendation

Change totalSupply state variable to a constant and change name to maxSupply: uint256 public constant maxSupply = <uint256>

BUYB-1 | buyBackWalletAddress can be constant

Description

If the value for buyBackWalletAddress will not change then it can be set to a constant, which will save gas when it is used for operations.

Recommendation

Change buyBackWalletAddress state variable to a constant:

address public constant _buyBackWalletAddress = <address>

LIQUI-1 | LiquidityWalletUpdated event not used

Description

The event LiquidityWalletUpdated is defined but is never used in the contract. If this event will not be used then it should be removed.

Recommendation

Remove LiquidityWalletUpdated from events

UNIS-1 | _uniswapV2Pair is instantiated and then set to state variable

Description

The memory variable _uniswapV2Pair is instantiated to get the address from the call to V2Factory in the constructor and in updateUniswapV2router function. This variable is then used to set the state variable uniswapV2Pair, which could be condensed into a single operation preventing the instantiation of a memory variable.

Recommendation

Condense the uniswapV2Pair state variable setter to one operation:
uniswapV2Pair = IUniswapV2Factor(_uniswapRouter.Factory()).createPair(address(this), _uniswapV2Router.WETH());

LOOP-1 | For loops initialize the i value to 0

Description

The default value for an initialized uint256 is 0, which means when it's used in a for loop the value for i does not need to be initialized to 0.

Recommendation

When the i value in a for loop is initialized to 0, change this to use default value:

```
for (uint256 i ; i < length; i++) {}
```

WALL-1 | Wallet setters do not check address

Description

The setter function for marketingWallet does not check if the address equals a zero address. This does not create any breaking problems until the swapAndLiquify is called, which would mean that the funds for marketing wallet could be sent to address zero.

Recommendation

Add a require statement to check the new address does not equal address zero.

```
require(wallet != address(0), “ZERO_ADDRESS”);
```

SWAP-1 | swapAndLiquifyEnabled

Description

The state variable for swapAndLiquifyEnabled is initialized to false in the LockToken contract. However, the default value for a bool in Solidity is false, meaning the variable can be defined and it will initialize to the same value.

Recommendation

Do not initialize the value for swapAndLiquifyEnabled on LockToken:
bool public swapAndLiquifyEnabled;

SALE-1 | updateMaxAllowedSalePercentang

Description

When the value for maxAllowedSalePercentage is changed with the setter there is no check that this is not a zero value. If the value were to be set to zero then when the third if statement is called in checkForWhale, the allowedToSell value would always return zero and every transaction would be reverted.

Recommendation

Require the input value for _percent is more than 0 for maxAllowedSalePercentage:

require(_percent > 0, “ZERO_VALUE”);

LOCKTOKEN ISSUES

OPEN-1 | isOpen State Variable Initialization

Description

The state variable for isOpen is initialized to false in the LockToken contract. However, the default value for a bool in Solidity is false, meaning the variable can be defined and it will initialize to the same value.

Recommendation

Do not initialize the value for isOpen on LockToken:

```
bool public isOpen;
```

TRADE-1 | stopTrade and openTrade

Description

The stopTrade and openTrade functions could be condensed into a single function which would save gas costs when calling other functions. As for every additional function in the contract the gas is used to traverse it when the function selector is used by the EVM.

Recommendation

Turn both functions into one called changeTradeStatus() which converts the value of isOpen to the opposition value:

```
function changeTradeStatus() external onlyOwner {
        isOpen = !isOpen;
}
```

UINT-1 | Initializing uint8

Description

When the for loop in includeToWhitelist is called, a uint8 is initialized to 0. Firstly, the default value for any uint that is initialized will be 0 meaning this does not need to be defined. Secondly, initializing as uint8 rather than uint256 will use more gas as the variable will occult 256 bits meaning 8 will be filled and the rest will need to be filled with zeros on initialization.

Recommendation

Initialize i in includeToWhitelist as a uint256 with default value:

```
for (uint256 i; i < _users.length; i++) {}
```

DIVIDEND PAYING TOKEN

CONS-3 | BUSD should be constant

Description

The value for the state variable BUSD is currently set to immutable whereas it should be set to constant. The difference is that constant is used for variables that can never be changed after compilation whereas immutable is used for state variables that can be set within the constructor.

Recommendation

Change BUSD state variable to a constant:

address public constant BUSD = <address>

WRAPP-1 | BUSD address does not need to be wrapped in address

Description

The state variable for BUSD receives an address value that is wrapped to be an address. However, this does not need to happen as the address value should be sufficient for the EVM to know this is an address that does not need any conversion.

Recommendation

Unwrap the address being passed to BUSD

address public constant BUSD = <address>

DIVID-1 | WithdrawDividend

Description

Currently withdrawDividend updates the mapping for the user and then transfers the tokens to the user. Often it's best practice to have an update function for the mapping and a withdraw function for the funds, which means user mappings can be updated without reverting risks. If following this approach, the two functions could then be added to a single function for dual functionality.

Recommendation

Break the update logic and the fund transfer logic into two public functions then create a function that contains both.

TRANS-1 | _transfer function

Description

The transfer function has a require statement that checks if false, however, as the default state is false this will always revert. As there is logic that is used to change information for magnifiedDividendCorrection, this should be changed to check the correct require statement rather than reverting on every call.

Recommendation

The require statement should be removed or check a condition that will not always revert.

GAS-1 | If statement comparing gasLeft with newGasLeft

Description

Unclear why this logic would be used as it compares the gasLeft (the effective starting gas or gas at the end of the loop) with newGasLeft (the gas used after operations are completed). As gas is used to complete operations, gasLeft will always be more than newGasLeft, making this logic redundant as it always executes.

Recommendation

Remove the if statement surrounding gas used logic.

Global Warning

SwapAndLiquidity on Skynet contract

The queries highlighted in swapAndLiquidity relate to whether the calculations being used will lead to the expected outcomes. For example, halfLiquidityTokens uses tokens as an input multiplied by liquidity fee divided by swap fee divided by 2. The name would suggest that the outcome should be half of a value however by multiplying and dividing twice it's unlikely the outcome would be within this range.

WIth the token value of 100,000, liquidity fee of 200, and swap fee of 200, the outcome of the calculation would be (((100,000 * 100)/ 400) / 2) or 12,500. Without understanding what type of outcome would like to be achieved from this calculation it is difficult to understand if it is operating correctly.

Similarly, bnbForLiquidity uses an approach that resembles the calculation from halfLiquidityToken and without more context the outcome can not be commented on.

_transfer on Dividend Paying Contract

The logic continued within this function adjusts the values for magnifiedDividendCorrections. However, it is strange for the from addresses values to be added to in a correcting process and the to address to be subtracted from. Without additional context it is difficult to evaluate this function as it's unclear what the purpose is, along with the

additional factor that the require statement will revert before this logic is executed.

Therefore, this logic will not be used and it may be purposeful but if this is the case then the function should be removed and if it is not the case more context would have to be provided to add comments.

_mint and _burn magnifiedDividendCorrection

Similarly to _transfer, there is logic that seems counterintuitive without more context provided. The mint function subtracts from the from address and the burn function adds to the from address, which is rarely common practice.

We would need to understand the purpose of magnifiedDividendCorrection to confirm whether this operation is working as expected.

magnitude usage

The state variable magnitude is used in a series of equations and without more context on what this stat is supposed to be, it is difficult to confirm whether the accumulativeDividendOf operation is acting as expected. Without rehashing the same comments, we need more information to confirm the math is correct.

DISCLAIMER

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to the Company in connection with the Agreement.

This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement.

This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without CoinMooner'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 considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts CoinMooner to perform a security assessment.

This report 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 in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing 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.

CoinMooner's position is that each company and individual are responsible for their own due diligence and continuous security. CoinMooner'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 fun.