Cyberscope

Audit Report

DreamAi

March 2023

Table of Contents

Review

Contract Name	DreamAi
Compiler Version	v0.8.19+commit.7dd6d404
Optimization	200 runs
Explorer	https://bscscan.com/address/0x013ce4c18aa43e7a8f0e8917d8fc01471a9c54f9
Address	0x013ce4c18aa43e7a8f0e8917d8fc01471a9c54f9
Network	BSC
Symbol	DRM
Decimals	9
Total Supply	100,000,000

Audit Updates

Initial Audit	25 Mar 2023

Source Files

Filename	SHA256
DreamAi.sol	dd272cb6afba7521c55c98e7a97ec365f7d8515e982e87ee2d63aaf26b50ad7b

Analysis

● Critical   ● Medium   ● Minor / Informative   ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Passed
●	OCTD	Transfers Contract's Tokens	Passed
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Passed
●	ULTW	Transfers Liquidity to Team Wallet	Passed
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

🔴 Critical   🟠 Medium   ⚪ Minor / Informative

Severity	Code	Description	Status
⚪	US	Untrusted Source	Unresolved
⚪	PTRP	Potential Transfer Revert Propagation	Unresolved
⚪	PVC	Price Volatility Concern	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
⚪	L11	Unnecessary Boolean equality	Unresolved
⚪	L12	Using Variables before Declaration	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

US - Untrusted Source

Criticality	Minor / Informative
Location	DreamAi.sol#L1073
Status	Unresolved

Description

The contract uses an external contract in order to determine the transaction's flow. The external contract is untrusted. As a result, it may produce security issues and harm the transactions.

```
function updateDividendTracker(address newAddress) public onlyOwner {
    //...
}
```

Recommendation

The contract should use a trusted external source. A trusted source could be either a commonly recognized or an audited contract. The pointing addresses should not be able to change after the initialization.

PTRP - Potential Transfer Revert Propagation

Criticality	Minor / Informative
Location	DreamAi.sol#L1411,1416
Status	Unresolved

Description

The contract sends funds to a `marketingWallet` , `developmentWallet` 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.

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 achived by not allowing set contract addresses or by sending the funds in a non-revertable way.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	DreamAi.sol#L1162
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.

```solidity
function setSwapTokensAtAmount(uint256 amount) external onlyOwner {
    swapTokensAtAmount = amount * 10**9;
}
```

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 total supply. Hence, the contract will guarantee that it cannot accumulate a huge amount of tokens in order to sell them.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	DreamAi.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 on 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 the 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	DreamAi.sol#L922,926,933,939
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	DreamAi.sol#L1002,1019
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.

```
string private currentRewardToken
uint256 private launchtax = 99
```

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	DreamAi.sol#L657,742,837,844,851,861,996,1021,1162,1167,1172,1454,1531
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);
uint256 constant internal magnitude = 2**128
address _owner
address public constant deadWallet =
0x000000000000000000000000000000000000dEaD
mapping(address => bool) public _isExcludedFromFees
uint256 _rewards
uint256 _marketing
uint256 _development
uint256 _liquidity
bool _enabled

...
```

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	DreamAi.sol#L524,1002,1017
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.

```solidity
int256 private constant MAX_INT256 = ~(int256(1) << 255)
string private currentRewardToken
mapping(address => uint256) private lastSell
```

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	DreamAi.sol#L1159,1185
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 = amount * 10**9
antiBotBlocks = numberOfBlocks
```

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	DreamAi.sol#L570,871
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.

```solidity
function abs(int256 a) internal pure returns (int256) {
        require(a != MIN_INT256);
        return a < 0 ? -a : a;
    }

function _transfer(address from, address to, uint256 value) internal virtual
override {
        require(false);

        int256 _magCorrection =
magnifiedDividendPerShare.mul(value).toInt256Safe();
        magnifiedDividendCorrections[from] =
magnifiedDividendCorrections[from].add(_magCorrection);
        magnifiedDividendCorrections[to] =
magnifiedDividendCorrections[to].sub(_magCorrection);
    }
```

Recommendation

To avoid creating dead code, it's important to carefully consider the logic and flow of the contract and to remove any code that is not needed or that is never executed. This can help improve the clarity and efficiency of the contract.

L11 - Unnecessary Boolean equality

Criticality	Minor / Informative
Location	DreamAi.sol#L1496
Status	Unresolved

Description

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

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

```
value == true
```

Recommendation

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

L12 - Using Variables before Declaration

Criticality	Minor / Informative
Location	DreamAi.sol#L797,1369,1370,1371
Status	Unresolved

Description

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

```
(bool success) = swapBnbForCustomToken(user, _withdrawableDividend)
uint256 iterations
uint256 claims
uint256 lastProcessedIndex
```

Recommendation

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

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	DreamAi.sol#L1396,1397,1405,1410,1415
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.

```solidity
uint256 unitBalance = deltaBalance / (denominator - sellTaxes.liquidity)
uint256 dividends = unitBalance * 2 * sellTaxes.rewards
```

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	DreamAi.sol#L807,1344,1369,1370,1371
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.

```
bool success
uint256 swapAmt
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	DreamAi.sol#L765,837,844,851,861
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	DreamAi.sol#L819
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.

```
rewardToken = newToken
```

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	DreamAi.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 ^0.8.19;
```

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	DreamAi.sol#L1104
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(tokenAddress).transfer(msg.sender,IERC20(tokenAddress).balanceOf(address(this)))
```

Recommendation

The contract should check if the result of the transfer methods is successful. The team is advised to check the SafeERC20 library from the Openzeppelin library.

Functions Analysis

Contract	Type	Bases		
	Function Name	Visibility	Mutability	Modifiers
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-
ERC20	Implementation	Context, IERC20, IERC20Metadata		

		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	_transfer	Internal	✓	
	_tokengeneration	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	
	_beforeTokenTransfer	Internal	✓	
SafeMath	Library			
	add	Internal		
	sub	Internal		
	sub	Internal		
	mul	Internal		

	div	Internal		
	div	Internal		
	mod	Internal		
	mod	Internal		
SafeMathInt	Library			
	mul	Internal		
	div	Internal		
	sub	Internal		
	add	Internal		
	abs	Internal		
	toUint256Safe	Internal		
SafeMathUint	Library			
	toInt256Safe	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
IPair	Interface			

	sync	External	✓	-
IFactory	Interface			
	createPair	External	✓	-
	getPair	External		-
IRouter	Interface			
	factory	External		-
	WETH	External		-
	addLiquidityETH	External	Payable	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
DividendPaying TokenInterface	Interface			
	dividendOf	External		-
	distributeDividends	External	Payable	-
	withdrawableDividendOf	External		-
	withdrawnDividendOf	External		-
	accumulativeDividendOf	External		-
DividendPaying Token	Implementation	ERC20, DividendPayingTokenInterf		

			ace, Ownable		
			Public	✓	ERC20
			External	Payable	-
	distributeDividends	Public	Payable	-
	_withdrawDividendOfUser	Internal	✓	
	setRewardToken	External	✓	onlyOwner
	swapBnbForCustomToken	Internal	✓	
	dividendOf	Public		-
	withdrawableDividendOf	Public		-
	withdrawnDividendOf	Public		-
	accumulativeDividendOf	Public		-
	_transfer	Internal	✓	
	_tokengeneration	Internal	✓	
	_burn	Internal	✓	
	_setBalance	Internal	✓	
IterableMapping	Library			
	get	Public		-
	getIndexOfKey	Public		-
	getKeyAtIndex	Public		-
	size	Public		-
	set	Public	✓	-
	remove	Public	✓	-

Address	Library			
	sendValue	Internal	✓	
DreamAi	Implementation	ERC20, Ownable		
		Public	✓	ERC20
		External	Payable	-
	updateDividendTracker	Public	✓	onlyOwner
	processDividendTracker	External	✓	-
	claim	External	✓	-
	rescueBEP20Tokens	External	✓	onlyOwner
	forceSend	External	✓	-
	excludeFromFees	Public	✓	onlyOwner
	excludeMultipleAccountsFromFees	Public	✓	onlyOwner
	excludeFromDividends	External	✓	onlyOwner
	setMarketingWallet	External	✓	onlyOwner
	setdevelopmentWallet	External	✓	onlyOwner
	setSwapTokensAtAmount	External	✓	onlyOwner
	setBuyTaxes	External	✓	onlyOwner
	setSellTaxes	External	✓	onlyOwner
	setSwapEnabled	External	✓	onlyOwner
	enableTradingEnabled	External	✓	onlyOwner
	setAntiBotBlocks	External	✓	onlyOwner

	setMinBalanceForDividends	External	✓	onlyOwner
	_setAutomatedMarketMakerPair	Private	✓	
	setGasForProcessing	External	✓	onlyOwner
	setClaimWait	External	✓	onlyOwner
	getClaimWait	External		-
	getTotalDividendsDistributed	External		-
	isExcludedFromFees	Public		-
	withdrawableDividendOf	Public		-
	getCurrentRewardToken	External		-
	dividendTokenBalanceOf	Public		-
	getAccountDividendsInfo	External		-
	getAccountDividendsInfoAtIndex	External		-
	getLastProcessedIndex	External		-
	getNumberOfDividendTokenHolders	External		-
	_transfer	Internal	✓	
	swapAndLiquify	Private	✓	
	swapTokensForBNB	Private	✓	
	addLiquidity	Private	✓	
DreamAi_DividendTracker	Implementation	Ownable, DividendPayingToken		
		Public	✓	DividendPayingToken
	_transfer	Internal		
	setMinBalanceForDividends	External	✓	onlyOwner

|				
excludeFromDividends	External	✓	onlyOwner
updateClaimWait	External	✓	onlyOwner
getLastProcessedIndex	External		-
getNumberOfTokenHolders	External		-
getCurrentRewardToken	External		-
getAccount	Public		-
getAccountAtIndex	Public		-
canAutoClaim	Private		
setBalance	Public	✓	onlyOwner
process	Public	✓	-
processAccount	Public	✓	onlyOwner

Inheritance Graph
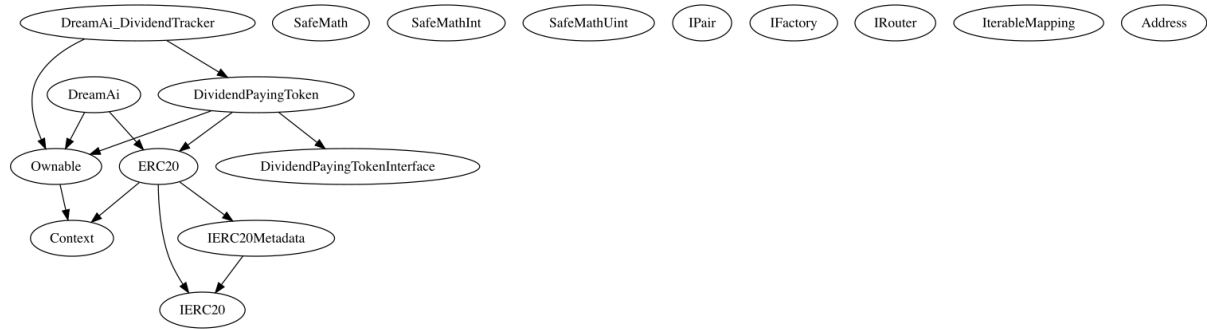
Flow Graph
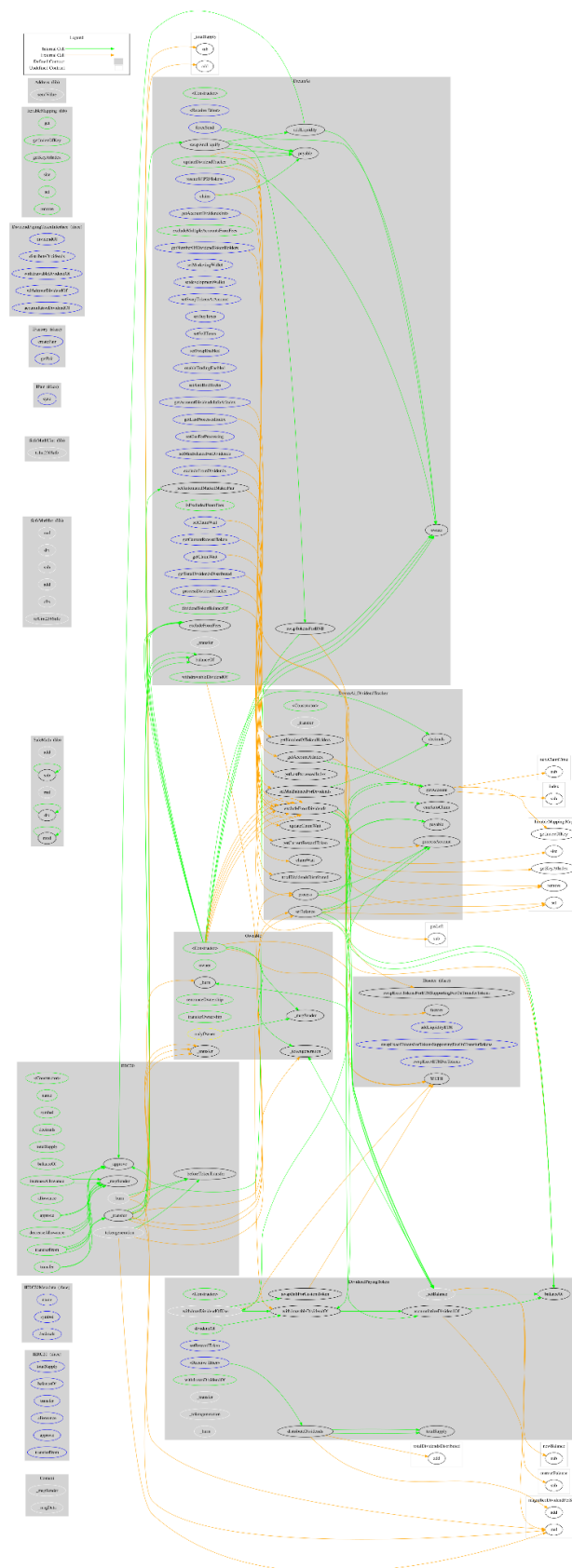
Summary

DreamAi contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. DreamAi is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler error or critical issues. The contract Owner can access some admin functions that can not be used in a malicious way to disturb the users' transactions. There is also a limit of max 12% 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