SKELETON ECOSYSTEM

SMART CONTRACT AUDIT

SUCCESS KID [SCCS]

BEP 20

0x7D7CE2c7e1d69deBE2219d9F33770fafd1AC29EE

Table of Contents

Global Disclaimer

This document serves as a disclaimer for the crypto smart contract audit conducted by Skeleton Ecosystem. The purpose of the audit was to review the codebase of the smart contracts for potential vulnerabilities and issues. It is important to note the following:

Limited Scope: The audit is based on the code and information available up to the audit completion date. It does not cover external factors, system interactions, or changes made after the audit. The audit itself can not guarantee 100% safaty and can not detect common scam methods like farming and developer sell-out.

No Guarantee of Security: While we have taken reasonable steps to identify vulnerabilities, it is impossible to guarantee the complete absence of security risks or issues. The audit report provides an assessment of the contract’s security as of the audit date.

Continued Development: Smart contracts and blockchain technology are evolving fields. Updates, forks, or changes to the contract post-audit may introduce new risks that were not present during the audit.

Third-party Code: If the smart contract relies on third-party libraries or code, those components were not thoroughly audited unless explicitly stated. Security of these dependencies is the responsibility of their respective developers.

Non-Exhaustive Testing: The audit involved automated analysis, manual review, and testing under controlled conditions. It is possible that certain vulnerabilities or issues may not have been identified.

Risk Evaluation: The audit report includes a risk assessment for identified vulnerabilities. It is recommended that the development team carefully reviews and addresses these risks to mitigate potential exploits.

Not Financial Advice: This audit report is not intended as financial or investment advice. Decisions regarding the use, deployment, or investment in the smart contract should be made based on a comprehensive assessment of the associated risks.

By accessing and using this audit report, you acknowledge and agree to the limitations outlined above. Skeleton Ecosystem and its auditors shall not be held liable for any direct or indirect damages resulting from the use of the audit report or the smart contract itself.

Please consult with legal, technical, and financial professionals before making any decisions related to the smart contract.

Overview

Contract Name	SUCCESS KID
Ticker/Simbol	SCCS
Blockchain	Binance Smart Chain Bep20
Contract Address	0x7D7CE2c7e1d69deBE2219d9F33770fafd1AC29EE
Creator Address	0x7f8816E7bbB128dD77A27F4b55125Af5794410d9
Current Owner Address	Renounced
Contract Explorer	https://bscscan.com/token/0x7d7ce2c7e1d69debe2219d9f33770fafd1ac29ee
Compiler Version	v0.8.0+commit.c7dfd78e
License	MIT License
Optimisation	No with 200 Runs
Total Supply	66,263,286,985.443692 SCCS
Decimals	18

Creation/Audit

Contract Deployed	18 May 2023
Audit Created	27-Aug-23 12:00:00 UTC
Audit Update	V 0.1

Verified Socials

Website	https://successkid.fun/
Telegram	https://t.me/successkidbnb
X	https://twitter.com/SuccessKidBNB

SUCCESS KID BEP20

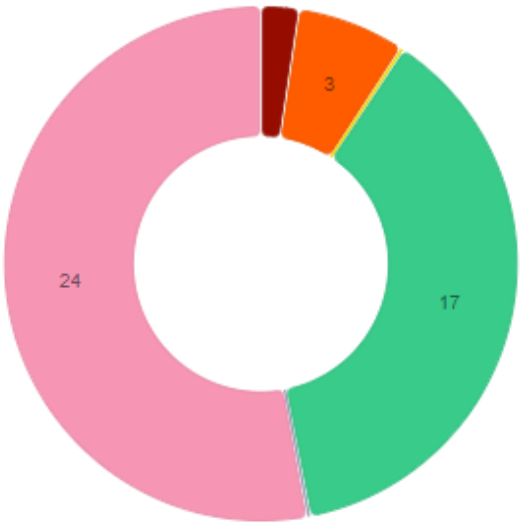
Contract Function Analysis

Pass Attention Item Risky Item

Contract Verified	✅	The contract source code is uploaded to blockchain explorer and is open source, so everybody can read it.
Contract Ownership	✅	The ownership of the contract was sent to dead address. With this the owner eliminates he's rights to modify the contract. The owner can not set any of the functions anymore.
Buy Tax	0 %	Shows the taxes for purchase transactions. Above 10% may be considered a high tax rate. More than 50% tax rate means may not be tradable. Contract renounced so tax rate is fixed. ✅
Sell Tax	10 %	Shows the taxes for sell transactions. Above 10% may be considered a high tax rate. More than 50% tax rate means may not be tradable. Contract renounced so tax rate is fixed. ✅
Honeypot Analyse	✅	Holder is able to buy and sell. If honeypot: The contract blocks sell transfer from holder wallet. Multiple events may cause honeypot. Trading disabled, extremely high tax
Liqudity Status	✅	Locked on 27.08.2023: 99% for 267 days on Unicrypt Note! Initial liqudity tokens scanned. For new LP Lockers allways re-check with skeleton scanner on telegram.
Trading Disable Functions	✅	No trading suspendable function found. If a suspendable code is included, the token maybe neither be bought or sold (honeypot risk). If contract is renounced this function can't be used. ⚠ If there is authorised hidden owner, or there is Retrieve Ownership Function, the trading disable function may be used!
Set Fees function	✅	No Fee Setting function found. The contract owner may contain the authority to modify the transaction tax. If the transaction tax is increased to more than 49%, the tokens may not be able to be traded (honeypot risk). If contract is renounced this function can't be used. ⚠ If there is authorised hidden owner, or there is Retrieve Ownership Function, the set fees function may be used! ✅ Renounced, this function can not be used.
Proxy Contract	✅	Not a Proxy Contract. The proxy contract means contract owner can modifiy the function of the token and possibly effect the price. The Owner is not the creator but the creator may have authorisation to change functions.
Mint Function	✅	No mint function found. Mint function is transparent or non-existent. Hidden mint functions may increase the amount of tokens in circulation and effect the price of the token. Owner can mint new tokens and sell. If contract is renounced this function can't be used.

Balance Modifier Function	✅	No Balance Modifier function found. If there is a function for this, the contract owner can have the authority to modify the balance of tokens at other addresses. For example revoke the bought tokens from the holders wallet. Common form of scam: You buy the token, but it's disappearing from your wallet. ⚠️ If contract is renounced this function still can be used as auto self Destruct
Whitelist Function	✅	No Whitelist Function Found. If there is a function for this Developer can set zero fee or no max wallet size for adresses (for example team wallets can trade without fee. Can cause farming) If there is a whitelist, some addresses may not be able to trade normally (honeypot risk). ✅ Renounced, this function can not be used.
Hidden Owner Analysis	✅	No authorised hidden owner found. For contract with a hidden owner, developer can still manipulate the contract even if the ownership has been abandoned. Fake renounce.
Retrieve Ownership Function	✅	No functions found which can retrieve ownership of the contract. If this function exists, it is possible for the project owner to regain ownership even after relinquishing it. Also known as fake renounce.
Self Destruct Function	✅	No Self Destruct function found. If this function exists and is triggered, the contract will be destroyed, all functions will be unavailable, and all related assets will be erased.
Specific Tax Changing Function	⚠️	Specific Tax Changing Functions found. ✅ Renounced, this function can not be used. If it exists, the contract owner may set a very outrageous tax rate for assigned address to block it from trading. Can assign all wallets at once!
Trading Cooldown Function	⚠️	Trading Cooldown Function found. ✅ Renounced, this function can not be used. If there is a trading cooldown function, the user will not be able to sell the token within a certain time or block after buying. Like a temporary honeypot.
Max Transaction and Holding Modify Function	⚠️	Max Transaction and Holding Modify function found. ✅ Renounced, this function can not be used. If there is a function for this, the maximum trading amount or maximum position can be modified. Can cause honeypot
Transaction Limiting Function	⚠️	Transaction Limiter Function Found. ✅ Renounced, this function can not be used. The number of overall token transactions may be limited (honeypot risk)

Contract Safety and Weakness



1	3	0	17	0	24
Crit	High	Med	Low	Infor	Gas

INCORRECT ACCESS CONTROL	1
UNCHECKED TRANSFER	1
APPROVE FRONT-RUNNING ATTACK	2
MISCONFIGURED BEFORETOKENTRAN...	3
USE OWNABLE2STEP	1
MISSING EVENTS	1
OUTDATED COMPILER VERSION	6
USE OF FLOATING PRAGMA	6
INTERNAL FUNCTIONS NEVER USED	1
DEFINE CONSTRUCTOR AS PAYABLE	2
FUNCTION SHOULD BE EXTERNAL	4
CHEAPER INEQUALITIES IN REQUIRE()	4
GAS OPTIMIZATION FOR STATE VARIA...	1
LONG REQUIRE/REVERT STRINGS	9
STORAGE VARIABLE CACHING IN MEM...	3

⚠️ Incorrect Access Control (1 item)

```
        require(newPair != address(0), "Invalid pair address");
        pair = newPair;
    }

    function publicBurn(uint amount) external {
        super._burn(msg.sender, amount);
    }

    function _transfer(address from, address to, uint256 amount) internal override {
        if (to == pair && pair != address(0)) {
            uint burnPart = amount * burnFee / 100;
            amount -= burnPart;
            super._burn(from, burnPart);
```

Function	Severity	Remedation
Access control plays an important role in segregation of privileges in smart contracts and other applications. If this is misconfigured or not properly validated on sensitive functions, it may lead to loss of funds, tokens and in some cases compromise of the smart contract. The contract SCCS is importing an access control library @openzeppelin/contracts/access/Ownable.sol but the function publicBurn is missing the modifier onlyOwner.	Severity : Critical	It is recommended to go through the contract and observe the functions that are lacking an access control modifier. If they contain sensitive administrative actions, it is advised to add a suitable modifier to the same

⚠ Unchecked Transfer (1 Item)

```
    function _transfer(address from, address to, uint256 amount) internal override {
        if (to == pair && pair != address(0)) {
            uint burnPart = amount * burnFee / 100;
            amount -= burnPart;
            super._burn(from, burnPart);
        }
        super._transfer(from, to, amount);
    }
}
```

Function	Severity	Remedation
Some tokens do not revert the transaction when the transfer or transferFrom fails and returns False. Hence we must check the return value after calling the transfer or transferFrom function.	Severity : High	Use OpenZeppelin SafeERC20's safetransfer and safetransferFrom functions.

⚠ Approve Front Running Attack (2 Items)

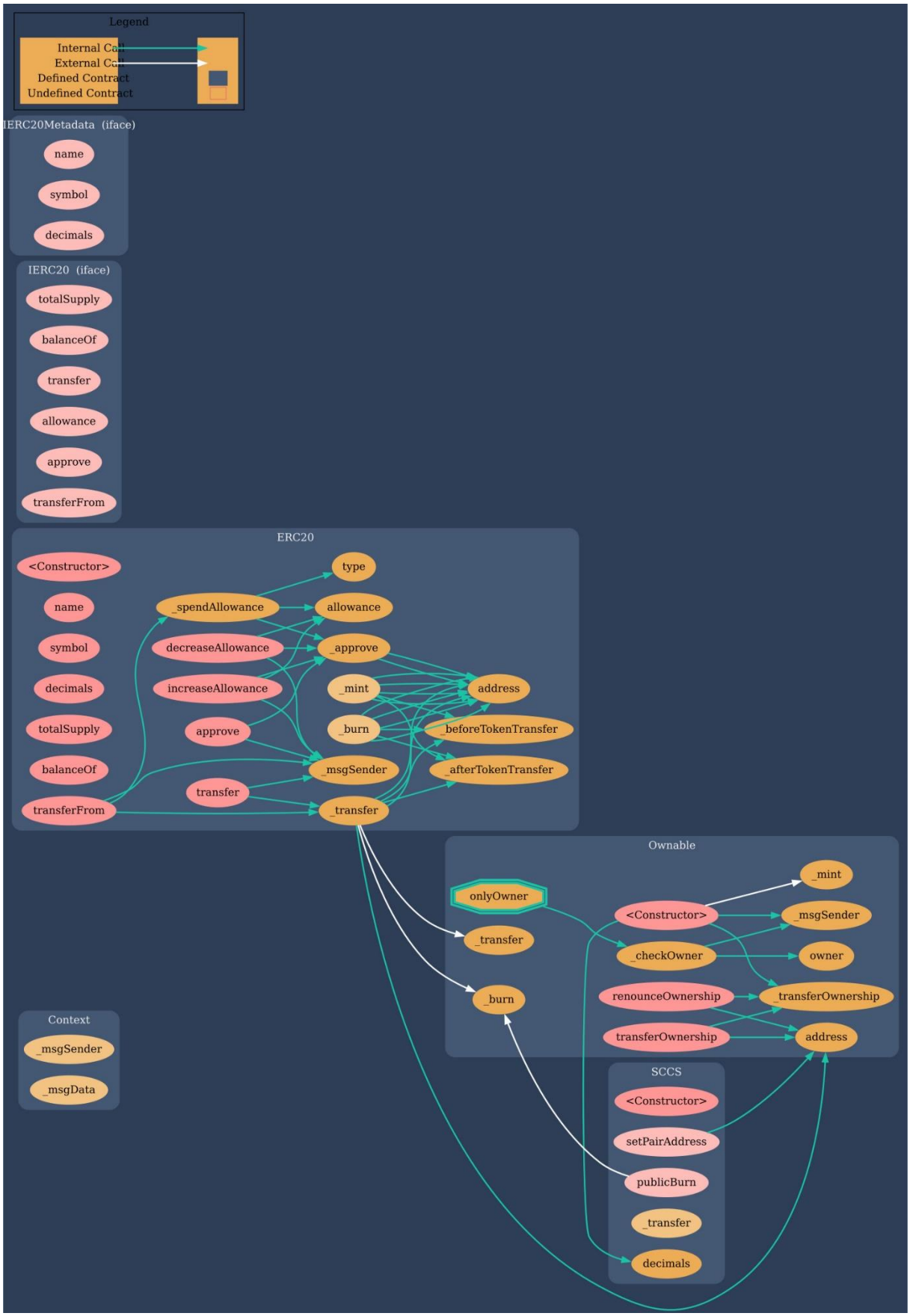
```
        return true;
    }

    function allowance(address owner, address spender) public view virtual override returns (uint256) {
        return _allowances[owner][spender];
    }

    function approve(address spender, uint256 amount) public virtual override returns (bool) {
        address owner = _msgSender();
        _approve(owner, spender, amount);
        return true;
    }

    function transferFrom(
        address from,
        address to,
```

```
        emit Approval(owner, spender, amount);
    }

    function _spendAllowance(
        address owner,
        address spender,
        uint256 amount
    ) internal virtual {
        uint256 currentAllowance = allowance(owner, spender);
        if (currentAllowance != type(uint256).max) {
            require(currentAllowance >= amount, "ERC20: insufficient allowance");
            unchecked {
                _approve(owner, spender, currentAllowance - amount);
            }
        }
    }

    function _beforeTokenTransfer(
        address from,
```

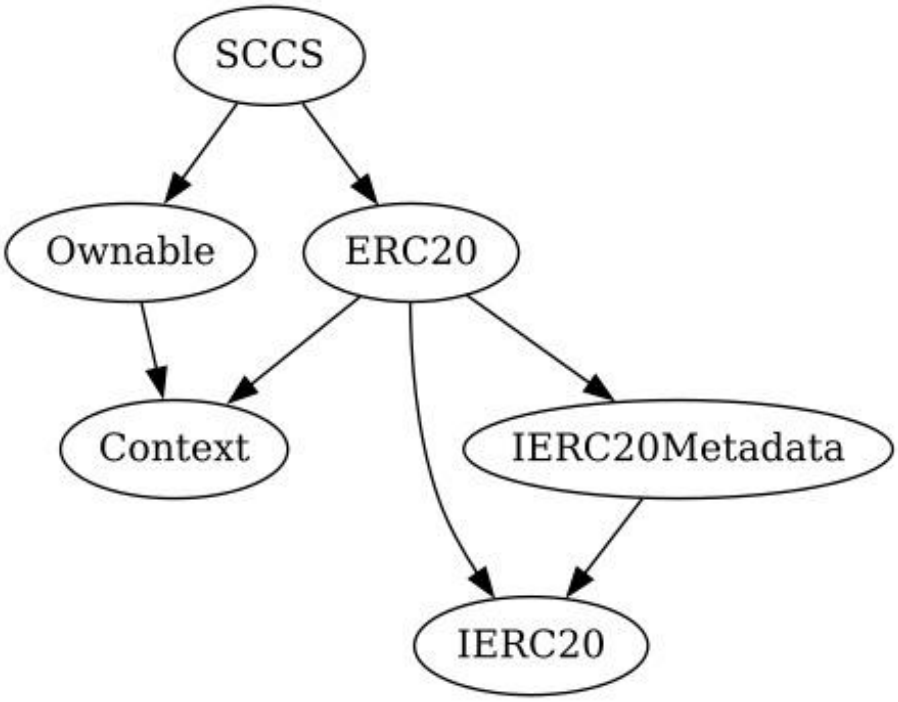
Function	Severity	Remedation
The approve() method overrides current allowance regardless of whether the spender already used it or not, so there is no way to increase or decrease allowance by a certain value atomically unless the token owner is a smart contract, not an account. This can be abused by a token receiver when they try to withdraw certain tokens from the sender's account. Meanwhile, if the sender decides to change the amount and sends another	Severity : High	Only use the approve function of the ERC/BEP standard to change the allowed amount to 0 or from 0 (wait till transaction is mined and approved). Token owner just needs to make sure that the first transaction actually changed allowance from N to 0, i.e., that the spender didn't manage to transfer some of N allowed tokens before the first transaction was mined. Such checking is possible using advanced blockchain explorers such as [Etherscan.io](https://etherscan.io/)

approve transaction, the receiver can notice this transaction before it's mined and can extract tokens from both the transactions, therefore, ending up with tokens from both the transactions. This is a front-running attack affecting the ERC20 Approve function. The function approve can be front-run by abusing the _approve function.		Another way to mitigate the threat is to approve token transfers only to smart contracts with verified source code that does not contain logic for performing attacks like described above, and to accounts owned by the people you may trust.

Contract Flow Graph



Inheritance Graph



Contract Descriptions

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Context	Implementation			
	_msgSender	Internal 🔒		
	_msgData	Internal 🔒		
Ownable	Implementation	Context		
		Public ❗	🛑	NO❗
	owner	Public ❗		NO❗
	_checkOwner	Internal 🔒		
	renounceOwnership	Public ❗	🛑	onlyOwner
	transferOwnership	Public ❗	🛑	onlyOwner
	_transferOwnership	Internal 🔒	🛑	
IERC20	Interface			
	totalSupply	External ❗		NO❗
	balanceOf	External ❗		NO❗
	transfer	External ❗	🛑	NO❗
	allowance	External ❗		NO❗
	approve	External ❗	🛑	NO❗
	transferFrom	External ❗	🛑	NO❗
IERC20Metadata	Interface	IERC20		
	name	External ❗		NO❗
	symbol	External ❗		NO❗
	decimals	External ❗		NO❗

ERC20	Implementation	Context, IERC20, IERC20Metadata		
		Public ❗	🛑	NO❗
	name	Public ❗		NO❗
	symbol	Public ❗		NO❗
	decimals	Public ❗		NO❗
	totalSupply	Public ❗		NO❗
	balanceOf	Public ❗		NO❗
	transfer	Public ❗	🛑	NO❗
	allowance	Public ❗		NO❗
	approve	Public ❗	🛑	NO❗
	transferFrom	Public ❗	🛑	NO❗
	increaseAllowance	Public ❗	🛑	NO❗
	decreaseAllowance	Public ❗	🛑	NO❗
	_transfer	Internal 🔒	🛑	
	_mint	Internal 🔒	🛑	
	_burn	Internal 🔒	🛑	
	_approve	Internal 🔒	🛑	
	_spendAllowance	Internal 🔒	🛑	
	_beforeTokenTransfer	Internal 🔒	🛑	
	_afterTokenTransfer	Internal 🔒	🛑	
SCCS	Implementation	ERC20, Ownable		
		Public ❗	🛑	ERC20
	setPairAddress	External ❗	🛑	onlyOwner
	publicBurn	External ❗	🛑	NO❗
	_transfer	Internal 🔒	🛑	

🛑 Function can modify state

💵 Function is payable

Source:

File Name SHA-1 Hash

c:\Solidity\successkid.sol 6d0937e60e2f453bd40fe2f8e20f124c4bf3eae1

Audit Scope

Audit Method.

Our smart contract audit is an extensive methodical examination and analysis of the smart contract's code that is used to interact with the blockchain. Goal: discover errors, issues and security vulnaribilities in the code. Findings getting reported and improvements getting suggested.

Automatic and Manual Review
We are using automated tools to scan functions and weeknesses of the contract. Transfers, integer over-undeflow checks such as all CWE events.

Tools we use:
Visual Studio Code
CWE
SWC
Solidity Scan
SVD

In manual code review our auditor looking at source code and performing line by line examination. This method helps to clarify developer's coding decisions and business logic.

Skeleton Ecosystem

https://skeletonecosystem.com

https://github.com/SkeletonEcosystem/Audits