SKELETON ECOSYSTEM

SMART CONTRACT AUDIT

TAU CETI
[TAU]
BEP 20

0x3ED1be864a7D08a3e3e72B28c567DEd1E5eE70c7

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	Tau Ceti
Ticker/Simbol	TAU
Blockchain	Binance Smart Chain BEP20
Contract Address	0x3ED1be864a7D08a3e3e72B28c567DEd1E5eE70c7
Creator Address	0x7715592be525a8cE67Ec14c2b13Bf50c9Ee10Ba4
Current Owner Address	0xFAc9364dDcD8995A4b6531Ac117E3304A2B6e186
Contract Explorer	https://bscscan.com/token/0x3ED1be864a7D08a3e3e72B28c567DEd1E5eE70c7#code
Compiler Version	v0.8.19+commit.7dd6d404
License	None
Optimisation	Yes with 200 Runs
Total Supply	10,000,000 **TAU**
Decimals	18

Creation/Audit

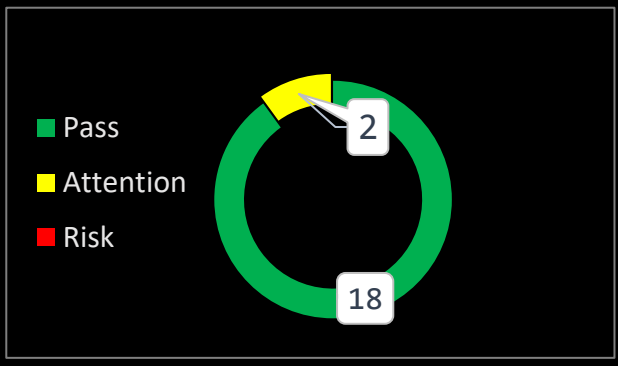
Contract Deployed	19 Sept 2023
Audit Created	25 Sept 2023
Audit Update	V 1.0

Verified Socials

Website	https://tau-ceti.info/
Telegram	https://t.me/TauCetiGlobal
Twitter (X)	https://x.com/TauCetiGlobal

SKELETON ECOSYSTEM
SMART CONTRACT AUDIT REPORT

TAU CETI BEP20



Contract Function Analysis

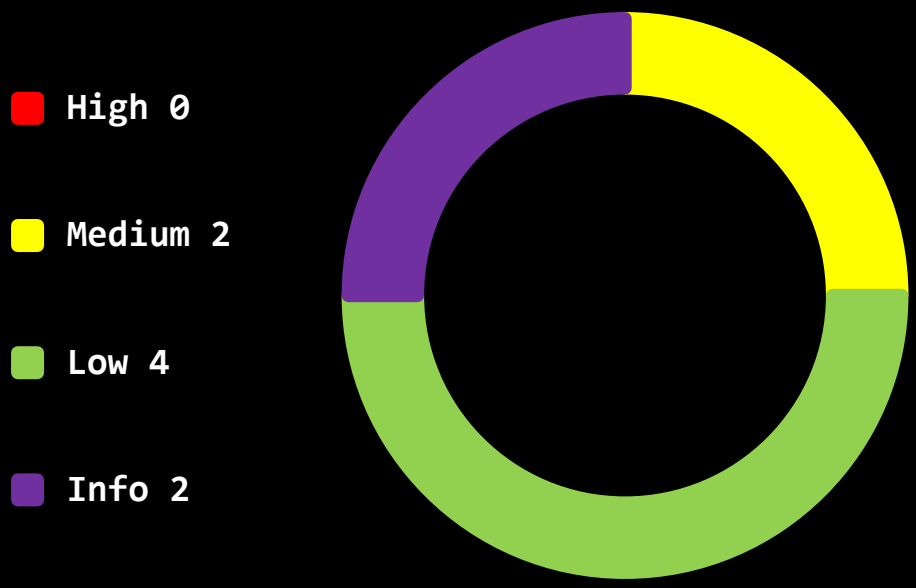
Pass Attention Item Risky Item

Contract Verified	✓	The contract source code is uploaded to blockchain explorer and is open source, so everybody can read it.
Contract Ownership	⚠	0xFAc9364dDcD8995A4b6531Ac117E3304A2B6e186
Buy Tax	5 %	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.
Sell Tax	5 %	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.
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		No Liqudity added yet. Presale phase!
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
Set Fees function	⚠ 25% max	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). Max fee setting option: 25%
Proxy Contract	✓	Not a proxy contract!
Mint Function	✓	No Mint Function detected 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.

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.
Blacklist Function	✅	No blacklist function found If there is a blacklist, some addresses may not be able to trade normally. Example: you buy the token and right after your Wallet getting blacklisted. Like so you will be unable to sell. Honeypot Risk.
Whitelist Function	⚠️	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)
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	✅	No Specific Tax Changing Functions found. 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	✅	No Trading Cooldown Function found. 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	✅	No Max Transaction and Holding Modify function found. If there is a function for this, the maximum trading amount or maximum position can be modified. Can cause honeypot
Transaction Limiting Function	✅	No Transaction Limiter Function Found. The number of overall token transactions may be limited (honeypot risk)

Contract Security

Total Findings: 8



High Severity Issues: High possibility to cause problems, need to be resolved.

Medium Severity Issue: Will likely cause problems, recommended to resolve.

Low Severity Issues: Won’t cause problems, but for improvement purposes could be adjusted.

Informational Severity Issues: Not harmful in any way, information for the developer team.

Contract Security

List of Found Issues

High severity Issues: (0)

Medium severity issues: (2)

- Approve of front running attack
- Unchecked Transfer

Low severity issues: (4)

- Numeric Notation Best Practices
- Use of Floating Pragma
- Numeric Notation Best Practices
- Missing Events

Informational severity issues: (2)

- Hard Coded Address
- Public Functions Should be Declared External

Contract Weakness Classisication

THE SMART CONTRACT WEAKNESS CLASSIFICATION REGISTRY (SWC REGISTRY) IS AN IMPLEMENTATION OF THE WEAKNESS CLASSIFICATION SCHEME PROPOSED IN EIP-1470. IT IS LOOSELY ALIGNED TO THE TERMINOLOGIES AND STRUCTURE USED IN THE COMMON WEAKNESS ENUMERATION (CWE) WHILE OVERLAYING A WIDE RANGE OF WEAKNESS VARIANTS THAT ARE SPECIFIC TO SMART CONTRACTS.

ID	Description	AI	Manual	Result
SWC-100	Function Default Visibility	Passed	Passed	Passed
SWC-101	Integer Overflow and Underflow	Passed	Passed	Passed
SWC-102	Outdated Compiler Version	Low	Passed	Passed
SWC-103	Floating Pragma	Low	Passed	Passed
SWC-104	Unchecked Call Return Value	Passed	Passed	Passed
SWC-105	Unprotected Ether Withdrawal	Passed	Passed	Passed
SWC-106	Unprotected SELFDESTRUCT Instruction	Passed	Passed	Passed
SWC-107	Reentrancy	Passed	Passed	Passed
SWC-108	State Variable Default Visibility	Passed	Passed	Passed
SWC-109	Uninitialized Storage Pointer	Passed	Passed	Passed
SWC-110	Assert Violation	Passed	Passed	Passed
SWC-111	Use of Deprecated Solidity Functions	Passed	Passed	Passed
SWC-112	Delegatecall to Untrusted Callee	Passed	Passed	Passed
SWC-113	DoS with Failed Call	Passed	Passed	Passed
SWC-114	Transaction Order Dependence	Passed	Passed	Passed
SWC-115	Authorization through tx.origin	Passed	Passed	Passed
SWC-116	Block values as a proxy for time	Passed	Passed	Passed
SWC-117	Signature Malleability	Passed	Passed	Passed
SWC-118	Incorrect Constructor Name	Passed	Passed	Passed

SWC-119	Shadowing State Variables	**Passed**	**Passed**	**Passed**
SWC-120	Weak Sources of Randomness from Chain Attributes	**Passed**	**Passed**	**Passed**
SWC-121	Missing Protection against Signature Replay Attacks	**Passed**	**Passed**	**Passed**
SWC-122	Lack of Proper Signature Verification	**Passed**	**Passed**	**Passed**
SWC-123	Requirement Violation	**Passed**	**Passed**	**Passed**
SWC-124	Write to Arbitrary Storage Location	**Passed**	**Passed**	**Passed**
SWC-125	Incorrect Inheritance Order	**Passed**	**Passed**	**Passed**
SWC-126	Insufficient Gas Griefing	**Passed**	**Passed**	**Passed**
SWC-127	Arbitrary Jump with Function Type Variable	**Passed**	**Passed**	**Passed**
SWC-128	DoS With Block Gas Limit	**Passed**	**Passed**	**Passed**
SWC-129	Typographical Error	**Passed**	**Passed**	**Passed**
SWC-130	Right-To-Left-Override control character (U+202E)	**Passed**	**Passed**	**Passed**
SWC-131	Presence of unused variables	**Passed**	**Passed**	**Passed**
SWC-132	Unexpected Ether balance	**Passed**	**Passed**	**Passed**
SWC-133	Hash Collisions With Multiple Variable Length Arguments	**Passed**	**Passed**	**Passed**
SWC-134	Message call with hardcoded gas amount	**Passed**	**Passed**	**Passed**
SWC-135	Code With No Effects	**Passed**	**Passed**	**Passed**
SWC-136	Unencrypted Private Data On-Chain	**Passed**	**Passed**	**Passed**

Detected High and Medium Severity Vulnerability Description

⚠ Unchecked Transfer (2 Items)

Item: 1	Location:	token.sol Line 177	Severity:	Medium

Function	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.
Remedation	Use OpenZeppelin SafeERC20's safetransfer and safetransferFrom functions.

```
176
        if (fees > 0) {
177
            super._transfer(from, address(this), fees);
178
```

Item: 2	Location:	token.sol Line 181	Severity:	Medium

Function	ome 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.
Remedation	Use OpenZeppelin SafeERC20's safetransfer and safetransferFrom functions.

```
181
        super._transfer(from, to, amount);
182


183
```

⚠ Approve of front running attack (2 Items)

Item: 1	Location:	erc20.sol Line 136-140	Severity:	Medium

Function	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 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.
Remedation	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/) 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.

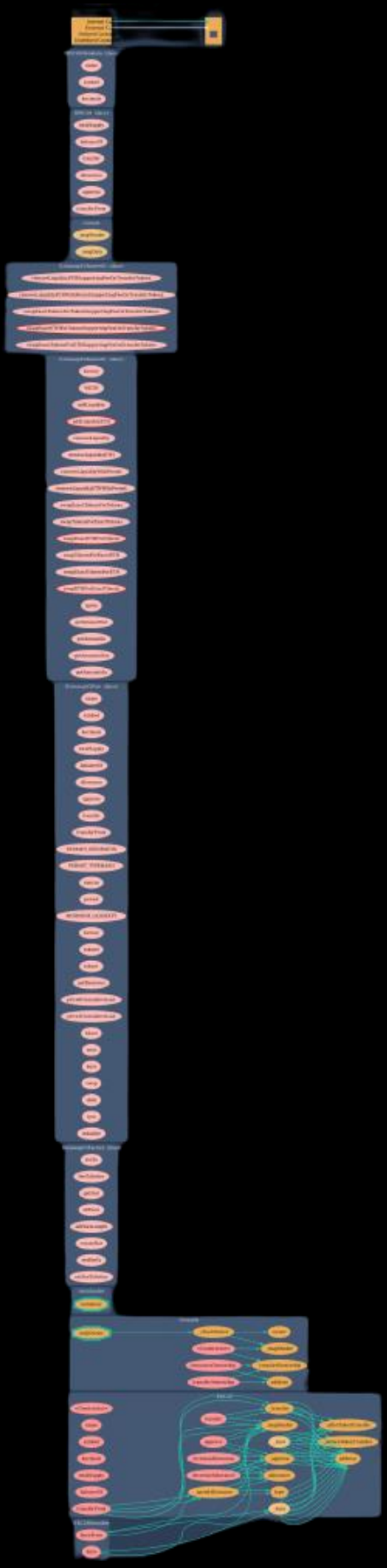
```
136
    function approve(address spender, uint256 amount) public virtual override returns (bool) {
137
        address owner = _msgSender();
138
        _approve(owner, spender, amount);
139
        return true;
140
    }
```

Item: 2	Location:	erc20.sol Line 324-330	Severity:	Medium

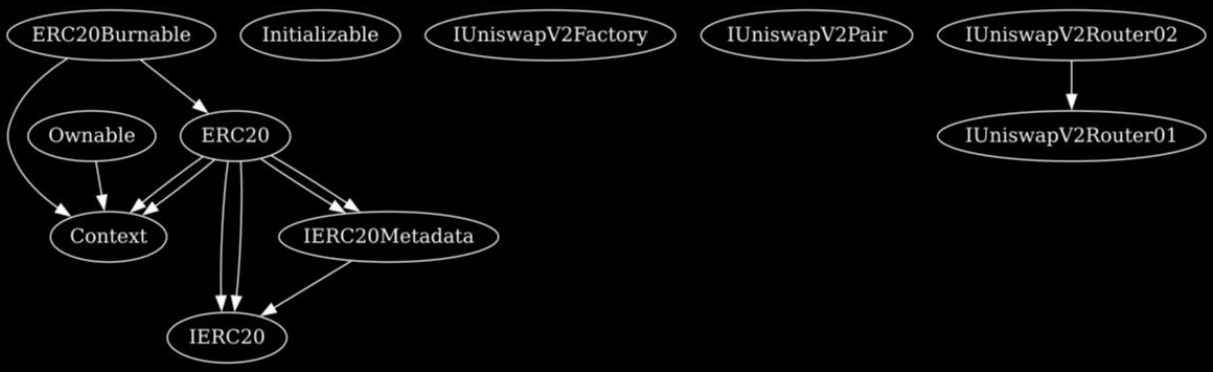
Function	The _spendAllowance() method overrides current allowance regardless of whether the spender already used it or not, so there is no way to increase or decrease allowance by a certain value atomically unless the token owner is a smart contract, not an account. This can be abused by a token receiver when they try to withdraw certain tokens from the sender's account. Meanwhile, if the sender decides to change the amount and sends another 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 _spendAllowance can be front-run by abusing the _approve function.
Remedation	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/) 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.

```
324
    function _spendAllowance(address owner, address spender, uint256 amount) internal virtual {
325
        uint256 currentAllowance = allowance(owner, spender);
326
        if (currentAllowance != type(uint256).max) {
327
            require(currentAllowance >= amount, "ERC20: insufficient allowance");
328
            unchecked {
329
                _approve(owner, spender, currentAllowance - amount);
330
```

Contract Flow Graph



Inheritance Graph



Contract Functions

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
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 🔒	🛑	
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 🔒	🛑	
ERC20Burnable	Implementation	Context, ERC20		
	burn	Public ❗	🛑	NO❗
	burnFrom	Public ❗	🛑	NO❗
Ownable	Implementation	Context		
		Public ❗	🛑	NO❗
	owner	Public ❗		NO❗
	_checkOwner	Internal 🔒		
	renounceOwnership	Public ❗	🛑	onlyOwner
	transferOwnership	Public ❗	🛑	onlyOwner
	_transferOwnership	Internal 🔒	🛑	

Initializable	Implementation			
IUniswapV2Factory	Interface			
	feeTo	External !		NO!
	feeToSetter	External !		NO!
	getPair	External !		NO!
	allPairs	External !		NO!
	allPairsLength	External !		NO!
	createPair	External !	🛑	NO!
	setFeeTo	External !	🛑	NO!
	setFeeToSetter	External !	🛑	NO!
IUniswapV2Pair	Interface			
	name	External !		NO!
	symbol	External !		NO!
	decimals	External !		NO!
	totalSupply	External !		NO!
	balanceOf	External !		NO!
	allowance	External !		NO!
	approve	External !	🛑	NO!
	transfer	External !	🛑	NO!
	transferFrom	External !	🛑	NO!
	DOMAIN_SEPARATOR	External !		NO!
	PERMIT_TYPEHASH	External !		NO!
	nonces	External !		NO!
	permit	External !	🛑	NO!
	MINIMUM_LIQUIDITY	External !		NO!
	factory	External !		NO!
	token0	External !		NO!
	token1	External !		NO!
	getReserves	External !		NO!
	price0CumulativeLast	External !		NO!
	price1CumulativeLast	External !		NO!

	kLast	External !		NO!
	mint	External !	🛑	NO!
	burn	External !	🛑	NO!
	swap	External !	🛑	NO!
	skim	External !	🛑	NO!
	sync	External !	🛑	NO!
	initialize	External !	🛑	NO!
IUniswapV2Router01	Interface			
	factory	External !		NO!
	WETH	External !		NO!
	addLiquidity	External !	🛑	NO!
	addLiquidityETH	External !	💵	NO!
	removeLiquidity	External !	🛑	NO!
	removeLiquidityETH	External !	🛑	NO!
	removeLiquidityWithPermit	External !	🛑	NO!
	removeLiquidityETHWithPermit	External !	🛑	NO!
	swapExactTokensForTokens	External !	🛑	NO!
	swapTokensForExactTokens	External !	🛑	NO!
	swapExactETHForTokens	External !	💵	NO!
	swapTokensForExactETH	External !	🛑	NO!
	swapExactTokensForETH	External !	🛑	NO!
	swapETHForExactTokens	External !	💵	NO!
	quote	External !		NO!
	getAmountOut	External !		NO!
	getAmountIn	External !		NO!

	getAmountsOut	External ❗		NO❗
	getAmountsIn	External ❗		NO❗
IUniswapV2Router02	Interface	IUniswapV2Router01		
	removeLiquidityETHSupportingFeeOnTransferTokens	External ❗	🛑	NO❗
	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External ❗	🛑	NO❗
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External ❗	🛑	NO❗
	swapExactETHForTokensSupportingFeeOnTransferTokens	External ❗	💵	NO❗
	swapExactTokensForETHSupportingFeeOnTransferTokens	External ❗	🛑	NO❗
Context	Implementation			
	_msgSender	Internal 🔒		
	_msgData	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!

🛑 Function can modify state

💵 Function is payable

Source:

File Name	SHA-1 Hash
c:\Solidity\tauceti.sol	162fa0d42ad300ea3f1f4bf6b5714d0640761f60

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