

SKELETON ECOSYSTEM

SMART CONTRACT AUDIT



Evo Chain Trust Fund ECTF BEP20

0x99c0c24a22ffe4f9d25db3cf518c33c4cd595a6



Table of Contents

Table of Contents	1
Disclaimer	2
Overview	3
Creation/Audit Date	3
Verified Socials	3
Contract Functions Analysis	4
Contract Safety and Weakness	7
Detected Vulnerability Description	11
Contract Flow Graph	15
Contract Interaction Graph	16
Inheritance Graph	17
Contract Descriptions	18
Audit Scope	22

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% safety 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	ECTF
Ticker/Symbol	ECTF
Blockchain	Binance Smart Chain BEP20
Contract Address	0x99c0c24a22ffe4f9d25db3cf518c33c4cd595a69
Creator Address	0xE9EbE90F90Aa6AED578161c898B25BC8F4c85E66
Current Owner Address	0xE9EbE90F90Aa6AED578161c898B25BC8F4c85E66
Contract Explorer	https://bscscan.com/address/0x99C0c24A22fFe4F9d25Db3CF518C33C4cd595A69#code
Compiler Version	v0.8.24+commit.e11b9ed9
License	MIT
Optimisation	No with 200 Runs
Total Supply	10,000,000 ECTF
Decimals	18




Creation/Audit

Contract Deployed	02.03.2024
Audit Created	14.03.2024
Audit Update	V 1.0

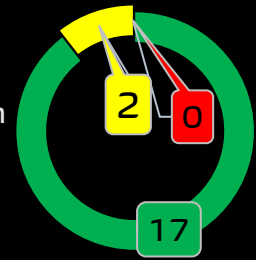
Verified Socials



Website	https://ectf.world/
Telegram	https://t.me/evotrustfund
Twitter (X)	https://x.com/evotrustfund

Contract Function Analysis

 Pass
  Attention Item
  Risky Item

■ Pass
 ■ Attention
 ■ Risk



Contract Verified		The contract source code is uploaded to blockchain explorer and is open source, so everybody can read it.
Contract Ownership		0xE9EbE90F90Aa6AED578161c898B25BC8F4c85E66
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. Fee can be set!
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. Fee can be set!
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
Liquidity Status		Liquidity status on 13.03.2024 Lp Locked: 99.99% Mudra Locker for 117 days.
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		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).
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		<p>No Balance Modifier function found.</p> <p>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.</p>
Blacklist Function		<p>No Blacklist Setting function found.</p> <p>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.</p>
Whitelist Function		<p>No Whitelist Setting function found</p> <p>If there is a function for this Developer can set zero fee or no max wallet size for addresses (for example team wallets can trade without fee. Can cause farming)</p>
Hidden Owner Analysis		<p>No Hidden or multi owner with authorisation</p> <p>For contract with a hidden owner, developer can still manipulate the contract even if the ownership has been abandoned.</p>
Retrieve Ownership Function		<p>No Functions found which can retrieve ownership of the contract.</p> <p>If this function exists, it is possible for the project owner to regain ownership even after relinquishing it. Also known as fake renounce.</p>
Self Destruct Function		<p>No Self Destruct function found.</p> <p>If this function exists and is triggered, the contract will be destroyed, all functions will be unavailable, and all related assets will be erased.</p>
Specific Tax Changing Function		<p>No Specific Tax Changing Functions found.</p> <p>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!</p>
Trading Cooldown Function		<p>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.</p>
Max Transaction and Holding Modify Function		<p>Max Transaction and Holding Modify function found.</p> <p>If there is a function for this, the maximum trading amount or maximum position can be modified. Can cause honeypot</p>
Transaction Limiting Function		<p>No Transaction Limiter Function Found.</p> <p>The number of overall token transactions may be limited (honeypot risk)</p>

Details of Risk - Attention Items

⚠ Set Fee

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).

```
fttrace | funcSig
50 function updateFees(uint256 newBuyFee!, uint256 newSellFee!) external onlyOwner {
51     require(newBuyFee! <= 5 && newSellFee! <= 5, 'Attempting to set fee higher than initial fee.');// smaller than or equal to initial fee
52     buyFee = newBuyFee!;
53     sellFee = newSellFee!;
54 }
55
```

⚠ Max Transaction and Holding Modify function

[Min 0.001% - Max 2%]

Reduced Risk

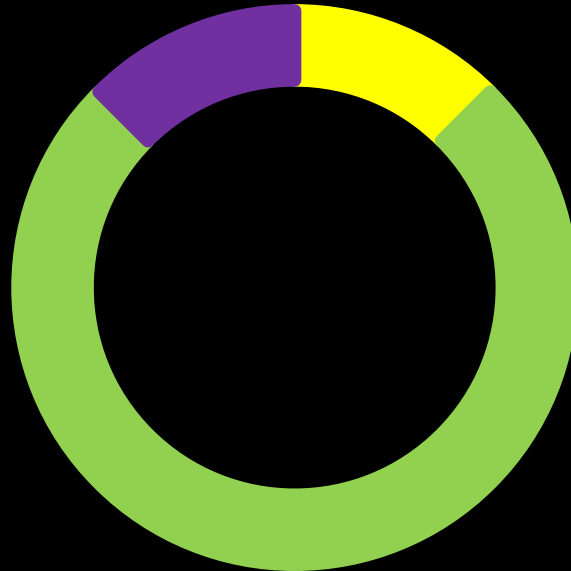
If there is a function for this, the maximum trading amount or maximum position can be modified. Can cause honeypot

```
fttrace | funcSig
444 function updateFeeThreshold(uint256 newThreshold!) external onlyOwner {
445     require(newThreshold! >= totalSupply().mul(1).div(100000), "Swap threshold cannot be lower than 0.001% total supply.");
446     require(newThreshold! <= totalSupply().mul(2).div(100), "Swap threshold cannot be higher than 2% total supply.");
447     feeThreshold = newThreshold!;
448 }
449
```

Contract Security

Total Findings: 8

- High 0
- Medium 1
- Low 6
- Info 1



■ **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: (1)

- Authorisation through Tx. Origin

Low severity issues: (6)

- Missing Events
- Long number literals
- Outdated compiler Version
- Unchecked Array Length
- Approve Front Running Attack
- Reentrancy

Informational severity issues: (1)

- Public Functions Should be Declared External

Contract Weakness Classification

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

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	Passed	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	High	Low	Low
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	High	Medium	Medium
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	low	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.

⚠️ Approve Front running Attack (2)

Item: 1	Location:	Line 96-100	Severity:	Low
---------	-----------	-------------	-----------	-----

Function	<p>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.</p> <p>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.</p> <p>The function approve can be front-run by abusing the _approve function.</p>
Remedation	<ol style="list-style-type: none"> 1. Introduce mechanisms that limit the maximum acceptable gas price for transactions. This can help prevent front-runners from drastically increasing the gas fees to prioritize their transactions. 2. Use transaction taxes to prevent against front-run attack

```

95 |
96 |     function approve(address spender, uint256 amount) public virtual override returns (bool) {
97 |         address owner = _msgSender();
98 |         _approve(owner, spender, amount);
99 |         return true;
100 |     }
101 |

```

Item: 2	Location:	Line 193-205	Severity: ■ Low
---------	-----------	--------------	--

Function	<p>The <code>_spendAllowance()</code> 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.</p> <p>This can be abused by a token receiver when they try to withdraw certain tokens from the sender's account.</p> <p>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.</p> <p>The function approve can be front-run by abusing the <code>_approve</code> function.</p>
Remedation	<ol style="list-style-type: none"> Introduce mechanisms that limit the maximum acceptable gas price for transactions. This can help prevent front-runners from drastically increasing the gas fees to prioritize their transactions. Use transaction taxes to prevent against front-run attack

```

ftrace | funcSig
193     function _spendAllowance(
194         address owner!,
195         address spender!,
196         uint256 amount!
197     ) internal virtual {
198         uint256 currentAllowance = allowance(owner!, spender!);
199         if (currentAllowance != type(uint256).max) {
200             require(currentAllowance >= amount!, "ERC20: insufficient allowance");
201             unchecked {
202                 _approve(owner!, spender!, currentAllowance - amount!);
203             }
204         }
205     }
206
  
```

⚠️ Authorisation through Tx. Origin (2 Items)

Item: 1	Location:	Line 355	Severity:	■ Medium
Item: 2	Location:	Line 359	Severity:	■ Medium

Function	In Solidity, tx.origin is a global variable that returns the address of the account that sent the transaction. Using the variable for authorization could make a contract vulnerable. For example, if an authorized account calls a malicious contract which triggers it to call the vulnerable contract that passes an authorization check since tx.origin returns the original sender of the transaction which in this case is the authorized account.
Remediation	The best way to prevent Tx Origin attacks is not to use the tx.origin for authentication purposes. Instead, it is advisable to use msg.sender

```

355     _isExcludedFromLimits[tx.origin] = true;
356     _isExcludedFromLimits[address(this)] = true;
357     _isExcludedFromLimits[address(0xdead)] = true;
358
359     _mint(tx.origin, totalSupply.sub(lpSupply));
360     _mint(msg.sender, lpSupply);
  
```

⚠ Reentrancy (1 Item)

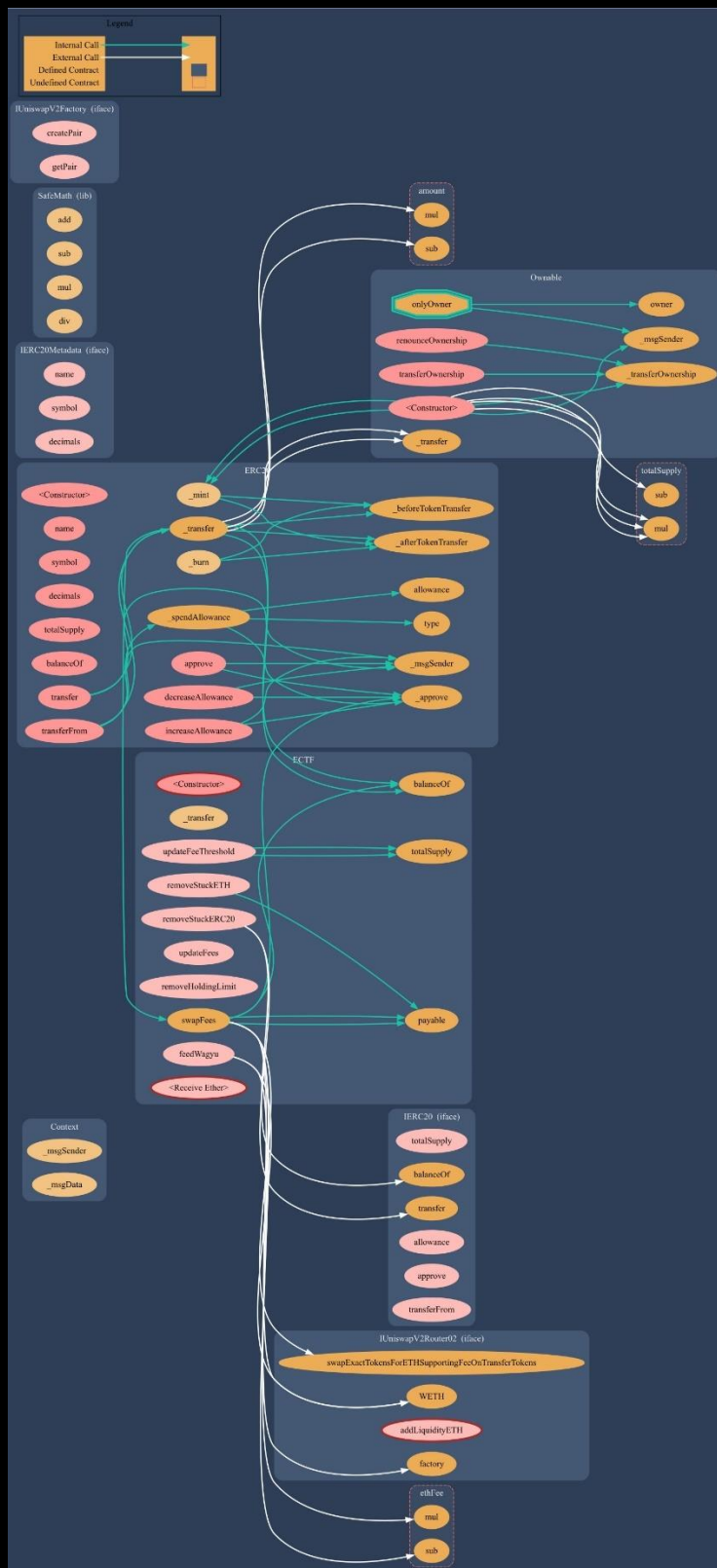
Item: 1	Location:	Line 408-437	Severity:	Low
---------	-----------	--------------	-----------	-----

Function	In a Re-entrancy attack, a malicious contract calls back into the calling contract before the first invocation of the function is finished. This may cause the different invocations of the function to interact in undesirable ways, especially in cases where the function is updating state variables after the external calls. This may lead to loss of funds, improper value updates, token loss, etc.
Remedation	<ul style="list-style-type: none"> Ensure all state changes happen before calling external contracts, i.e., update balances or code internally before calling external code Use function modifiers that prevent reentrancy

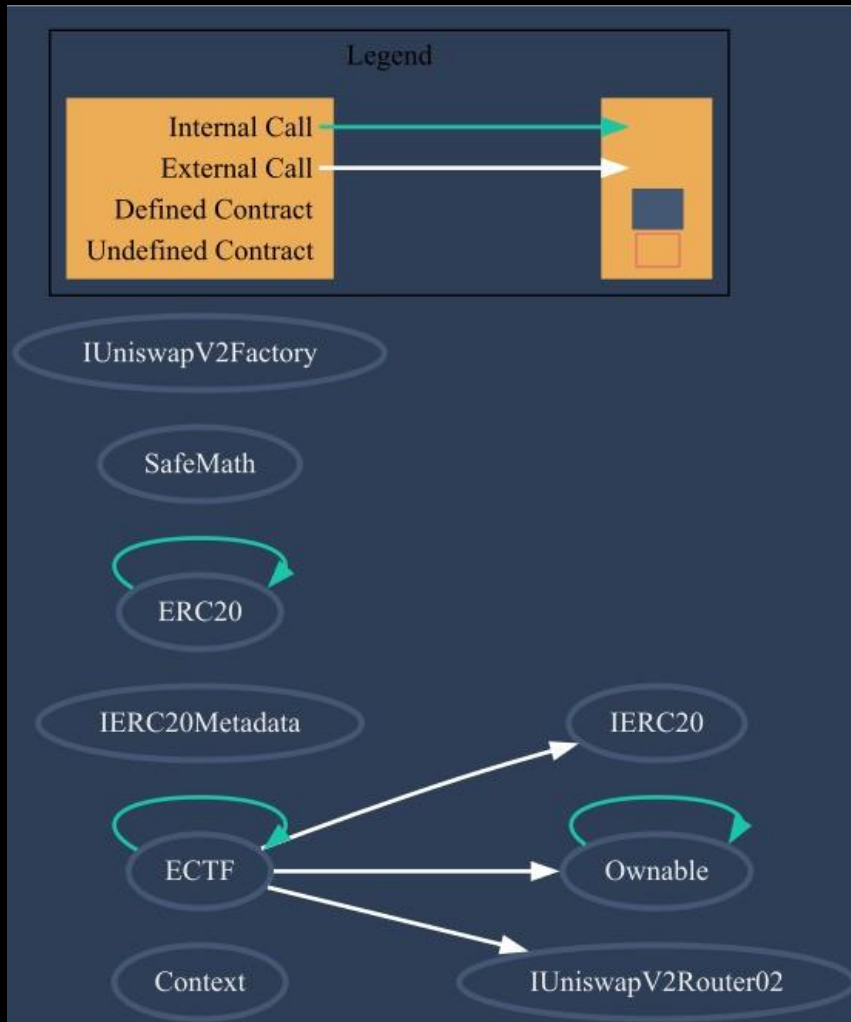
```

408      function swapFees() public {
409          uint256 contractBalance = balanceOf(address(this));
410          if (contractBalance == 0) return;
411          if (contractBalance > feeThreshold) contractBalance = feeThreshold;
412
413          uint256 initETHBal = address(this).balance;
414
415          address[] memory path = new address[](2);
416          path[0] = address(this);
417          path[1] = _router.WETH();
418
419          _approve(address(this), address(_router), contractBalance);
420
421          _router.swapExactTokensForETHSupportingFeeOnTransferTokens(
422              contractBalance,
423              0,
424              path,
425              address(this),
426              block.timestamp
427          );
428
429          uint256 ethFee = address(this).balance.sub(initETHBal);
430          uint256 revFee = ethFee.mul(20).div(100);
431
432          ethFee = ethFee.sub(revFee);
433          payable(teamAddr).transfer(ethFee);
434          payable(revshareAddr).transfer(revFee);
435
436          emit RevFee(revFee);
437      }
438  
```

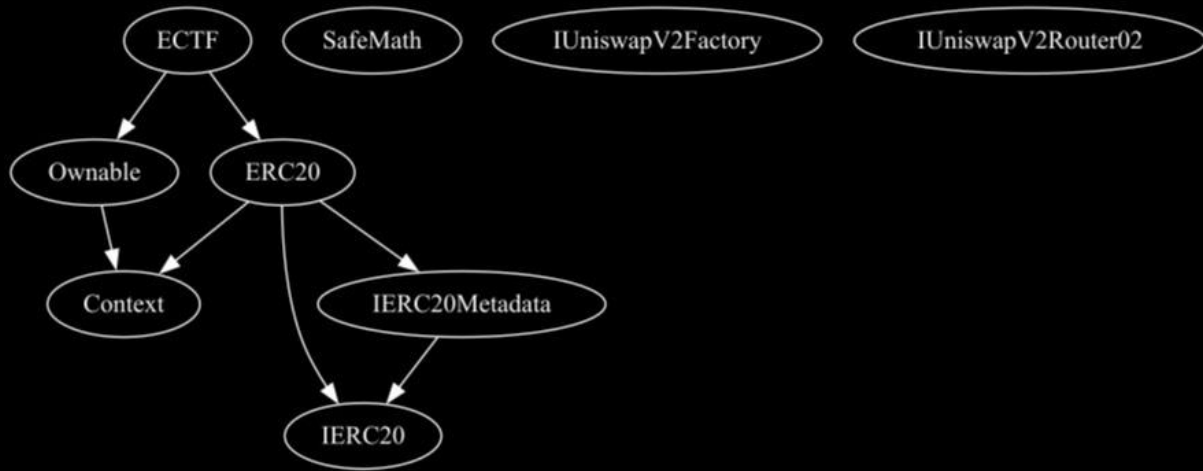
Contract Flow Graph



























Contract Interaction Graph















































































Inheritance Graph







Contract Functions

Contract	Type	Bases		
L	Function Name	Visibility	Mutability	Modifiers
Context	Implementation			
L	_msgSender	Internal 		
L	_msgData	Internal 		
IERC20	Interface			
L	totalSupply	External 		NO 
L	balanceOf	External 		NO 
L	transfer	External 		NO 
L	allowance	External 		NO 
L	approve	External 		NO 
L	transferFrom	External 		NO 
IERC20Metadata	Interface	IERC20		
L	name	External 		NO 
L	symbol	External 		NO 
L	decimals	External 		NO 
ERC20	Implementation	Context, IERC20, IERC20Metadata		
L		Public 		NO 
L	name	Public 		NO 
L	symbol	Public 		NO 
L	decimals	Public 		NO 
L	totalSupply	Public 		NO 

Contract	Type	Bases		
L	balanceOf	Public 		NO 
L	transfer	Public 		NO 
L	allowance	Public 		NO 
L	approve	Public 		NO 
L	transferFrom	Public 		NO 
L	increaseAllowance	Public 		NO 
L	decreaseAllowance	Public 		NO 
L	_transfer	Internal 		
L	_mint	Internal 		
L	_burn	Internal 		
L	_approve	Internal 		
L	_spendAllowance	Internal 		
L	_beforeTokenTransfer	Internal 		
L	_afterTokenTransfer	Internal 		
Ownable	Implementation	Context		
L		Public 		NO 
L	owner	Public 		NO 
L	renounceOwnership	Public 		onlyOwner
L	transferOwnership	Public 		onlyOwner
L	_transferOwnership	Internal 		
SafeMath	Library			

Contract	Type	Bases		
L	add	Internal 		
L	sub	Internal 		
L	mul	Internal 		
L	div	Internal 		
L	sub	Internal 		
L	div	Internal 		
IUniswapV2Factory	Interface			
L	createPair	External 		NO 
L	getPair	External 		NO 
IUniswapV2Router02	Interface			
L	factory	External 		NO 
L	WETH	External 		NO 
L	addLiquidityETH	External 		NO 
L	swapExactTokensForETHSupportingFeeOnTransferTokens	External 		NO 
ECTF	Implementation	ERC20, Ownable		
L		Public 		ERC20
L	_transfer	Internal 		
L	swapFees	Public 		NO 
L	feedWagyu	External 		onlyOwner
L	updateFeeThreshold	External 		onlyOwner

Contract	Type	Bases		
L	updateFees	External 		onlyOwner
L	removeHoldingLimit	External 		onlyOwner
L	removeStuckETH	External 		onlyOwner
L	removeStuckERC20	External 		onlyOwner
L		External 		NO 



Function
can modify
state



Function
is payable

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 vulnerabilities in the code. Findings getting reported and improvements getting suggested.

Automatic and Manual Review

We are using automated tools to scan functions and weaknesses 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>

