SKELETON ECOSYSTEM

SMART CONTRACT AUDIT

SOLANA ETF
$SOLETF
BEP20

0xBdB653c41E4B5e9cE70FE237386b714FEC2fC

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	**SolanaETF**
Ticker/Simbol	SOLETF
Blockchain	Binance Smart Chain BEP20
Contract Address	0xBdB653c41E4B5e9cE70FE237386b714FEC2fCFcc
Creator Address	0x2D09F4DDA02591A3253fFfd59583940cb08c6e3f
Current Owner Address	0x0000000000000000000000000000000000000000
Contract Explorer	https://bscscan.com/token/0xBdB653c41E4B5e9cE70FE237386b714FEC2fCFcc#code
Compiler Version	v0.8.19+commit.7dd6d404
License	NONE
Optimisation	Yes with 1873 Runs
Total Supply	69,000,000,000,000,000 **SOLETF**
Decimals	9

Creation/Audit

Contract Deployed	31.05.2024
Audit Created	02.06.2024
Audit Update	V 1.0

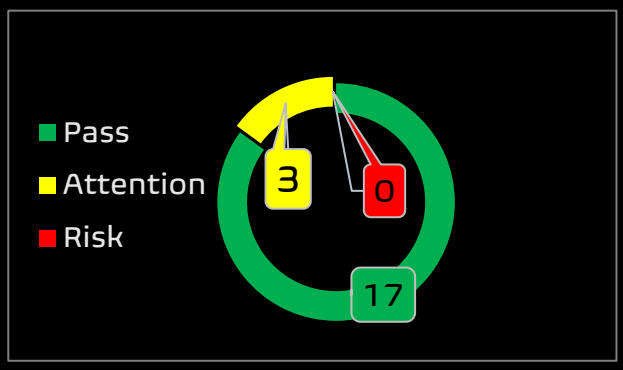
Verified Socials

Website	https://www.coinscope.co/coin/soletf
Telegram	https://t.me/Solanuh
Twitter (X)	https://x.com/SOLETFX1

SOLANA ETF 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		0x0000000000000000000000000000000000000000 Sometimes referred to as the "zero address" or "dead address" and is not owned by anyone.
Buy Tax	5 % final	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	21 % final	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! Fee Structure: 10% marketing – 10% reward – 1% burn
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	✓	Liqudity status on 02.06.2024 100.00% Locked on Mudra Locker for 31 days. https://bscscan.com/tx/0x779177562ff08f19cfe2ee22534152eb75aa4e3a880aa79bd239881c1713dbbb
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	✅	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	⚠️	Blacklist Setting function found. Exclude wallets from receiving dividends only. No Blacklist from trading If there is a function for this, some wallets can be blacklisted and will not receive the reward token.
Whitelist Function	⚠️	Whitelist Setting function found. Contract renounced, function can not be triggered by owner. 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 Hidden or multi owner with authorisation For contract with a hidden owner, developer can still manipulate the contract even if the ownership has been abandoned.
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	✅	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)

Details of Risk - Attention Items

Removing Risk of contract function based on renounced ownership

Transaction Receipt Event Logs

57

Address 0xbdb653c41e4b5e9ce70fe237386b714fec2fcfcc

Name OwnershipTransferred (index_topic_1 address previousOwner, index_topic_2 address newOwner) View Source

Topics
0 0x8be0079c531659141344cd1fd0a4f28419497f9722a3daafe3b4186f6b6457e0
1: previousOwner Dec → 0x2D09F4DDA02591A3253fFfd59583940cb08c6e3f
2: newOwner Dec → 0x0000000000000000000000000000000000000000

Data 0x

Following detected contract functions serve as informational purposes about the contract. The owner has no more authorisation to trigger the following functions.

⚠️ Relative high Sell Taxes

(Sell Tax 21%) and can not be modified.

Found Tax Structure explanation:

10% Reward (Solana Token BEP20)

10% Marketing

1% AutoBurn

```
943    burnFeeOnSell        = 1;
944    marketingFeeOnSell    = 10;
945    rewardsFeeOnSell     = 10;
946
947    totalSellFee         = burnFeeOnSell + marketingFeeOnSell + rewardsFeeOnSell;
948
```

⚠ Whitelist

Contract renounced, function can not be triggered by owner.

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)

```
ftrace | funcSig
function excludeFromFees(address account, bool excluded) external onlyOwner {
    require(_isExcludedFromFees[account] != excluded, "Account is already set to that state");
    _isExcludedFromFees[account] = excluded;

    emit ExcludeFromFees(account, excluded);
}

ftrace | funcSig
function isExcludedFromFees(address account) public view returns(bool) {
    return _isExcludedFromFees[account];
}
```

⚠ Blacklist

(Exclude wallets from receiving dividends only. No Blacklist from trading)

Contract renounced, function can not be triggered by owner.

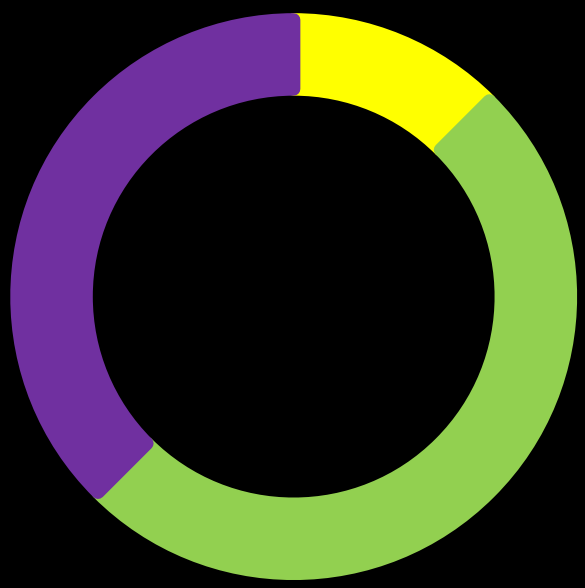
If there is a function for this, some wallets can be blacklisted and will not receive the reward token.

```
ftrace | funcSig
function excludeFromDividends(address account) external onlyOwner {
    require(!excludedFromDividends[account]);
    excludedFromDividends[account] = true;

    _setBalance(account, 0);
    tokenHoldersMap.remove(account);

    emit ExcludeFromDividends(account);
}
```

Contract Security

Total Findings: 8

- High 0
- Medium 1
- Low 4
- Info 3



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)

- Use of TX.orifin

Low severity issues: (4)

- Missing Events
- Long number literals
- Outdated Compiler Version
- Approve of front running attack

Informational severity issues: (3)

- Public Functions Should be Declared External
- State Variables Should be Declared Constant
- Code With No Effects

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

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	low	low
SWC-103	Floating Pragma	Passed	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	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.

⚠ Authorisation by using TX.origin (2 Item)

Item: 1	Location:	Line 1126	Severity:	Medium
Item: 2	Location:	Line: 1217	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.
Remedation	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

```
1124
1125        try dividendTracker.process(gas) returns (uint256 iterations, uint256 claims, uint256 lastProcessedIndex) {
1126            emit ProcessedDividendTracker(iterations, claims, lastProcessedIndex, true, gas, tx.origin);
1127        }
```

```
            ftrace | funcSig
1215    function processDividendTracker(uint256 gas!) external {
1216        (uint256 iterations, uint256 claims, uint256 lastProcessedIndex) = dividendTracker.process(gas!);
1217        emit ProcessedDividendTracker(iterations, claims, lastProcessedIndex, false, gas!, tx.origin);
1218    }
```

⚠ Approve of front running attack. Also known as Sandwich Bot attack. (2 Item)

Item: 1	Location:	Line 475-478	Severity:	Low

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	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

```
ftrace | funcSig
475 function approve(address spender, uint256 amount) public virtual override returns (bool) {
476     _approve(_msgSender(), spender, amount);
477     return true;
478 }
479
```

Item: 2	Location:	Line 480-488	Severity:	Low

Function	The transferFrom() 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	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

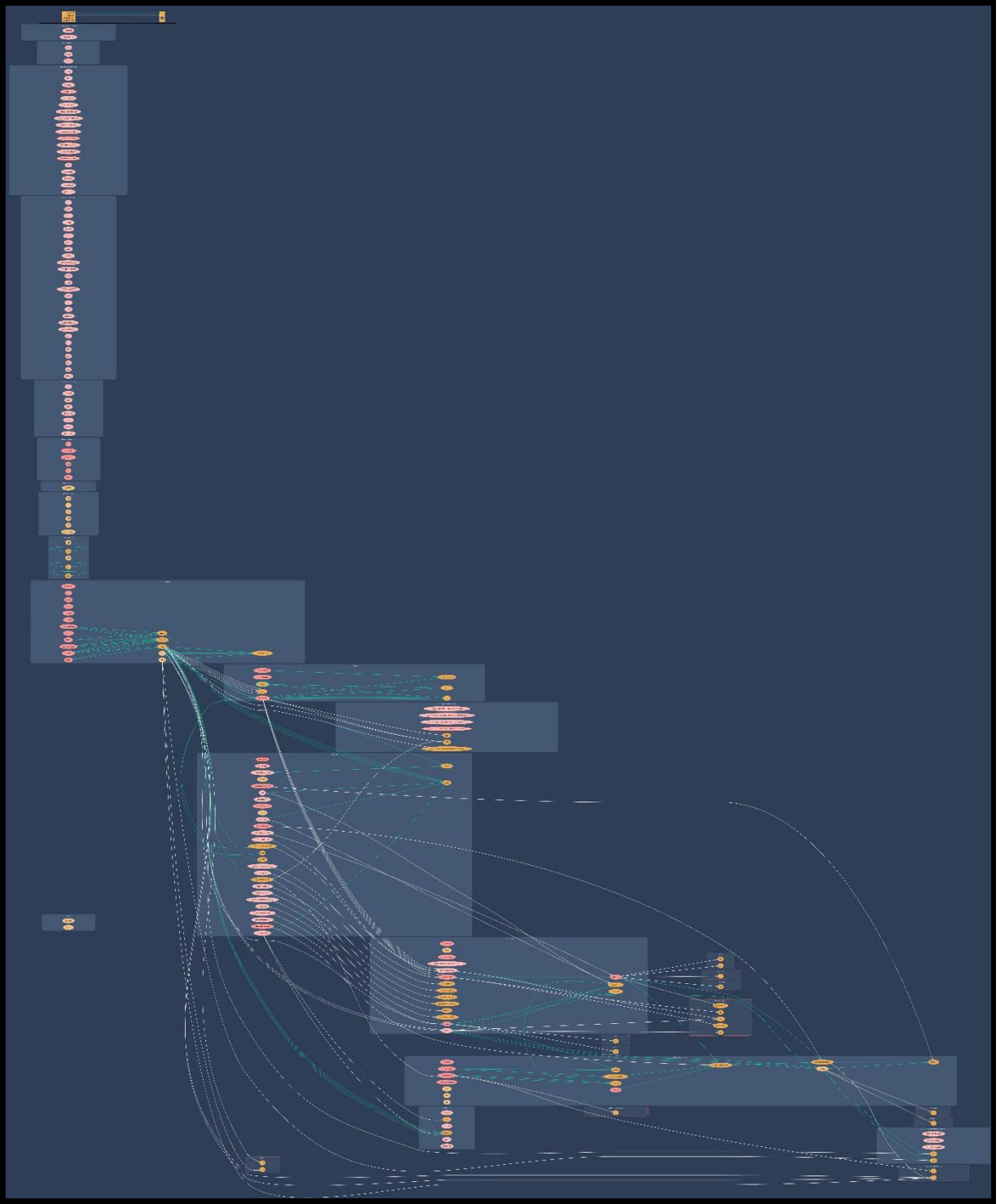
```
479
        ftrace | funcSig
480     function transferFrom(
481         address sender,
482         address recipient,
483         uint256 amount
484     ) public virtual override returns (bool) {
485         _transfer(sender, recipient, amount);
486         _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, "ERC20: transfer amount exceeds allowance"));
487         return true;
488     }
489
```

⚠ Outdated Compiler Version

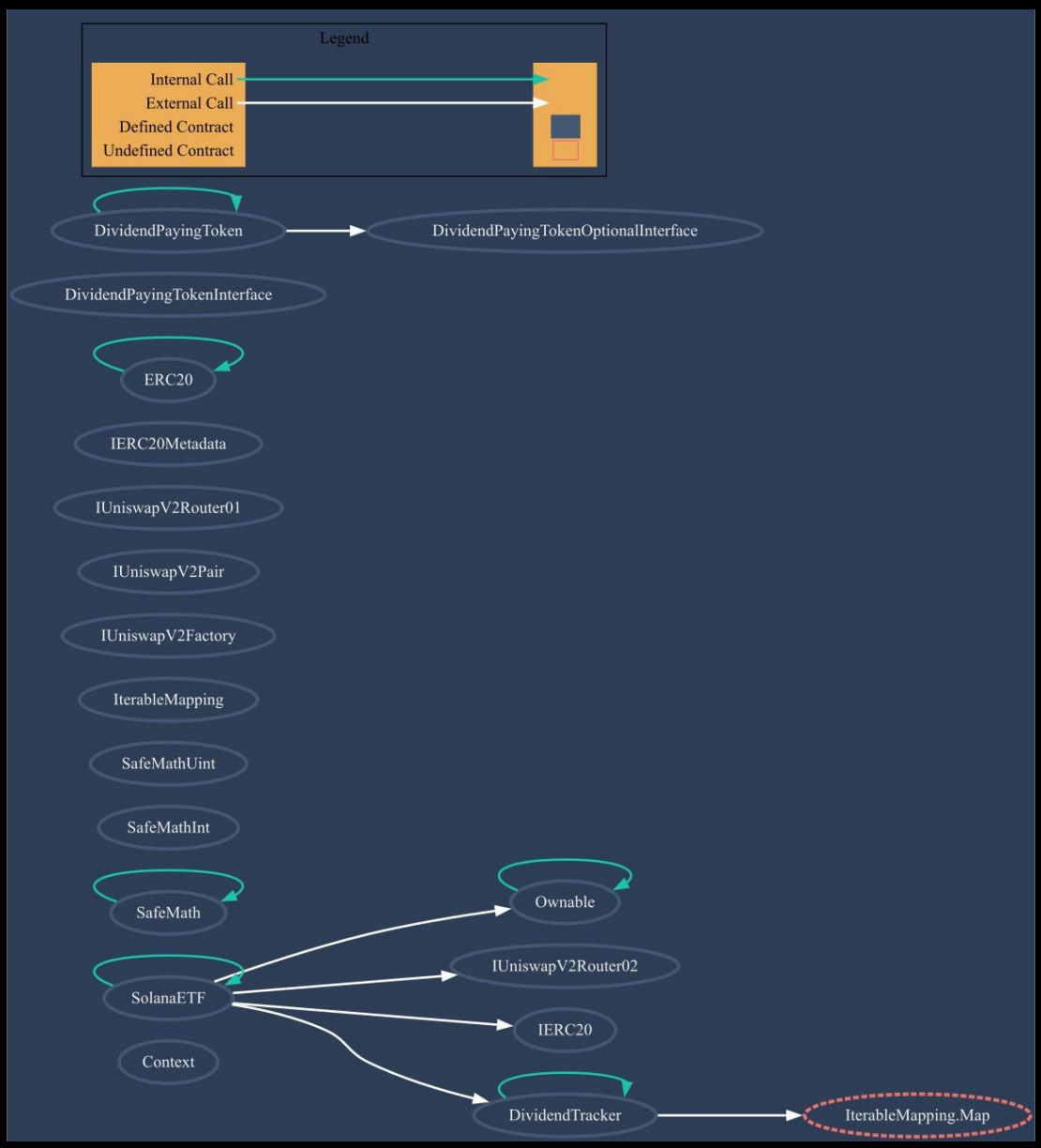
Item: 1	Location:	Line 10	Severity:	Low

Function	Using an outdated compiler version can be problematic especially if there are publicly disclosed bugs and issues that affect the current compiler version. The following outdated versions were detected: /soletf.sol - 0.8.19
Remedation	It is recommended to use a recent version of the Solidity compiler that should not be the most recent version, and it should not be an outdated version as well. Using very old versions of Solidity prevents the benefits of bug fixes and newer security checks. Consider using the solidity version v0.8.25, which patches most solidity vulnerabilities.

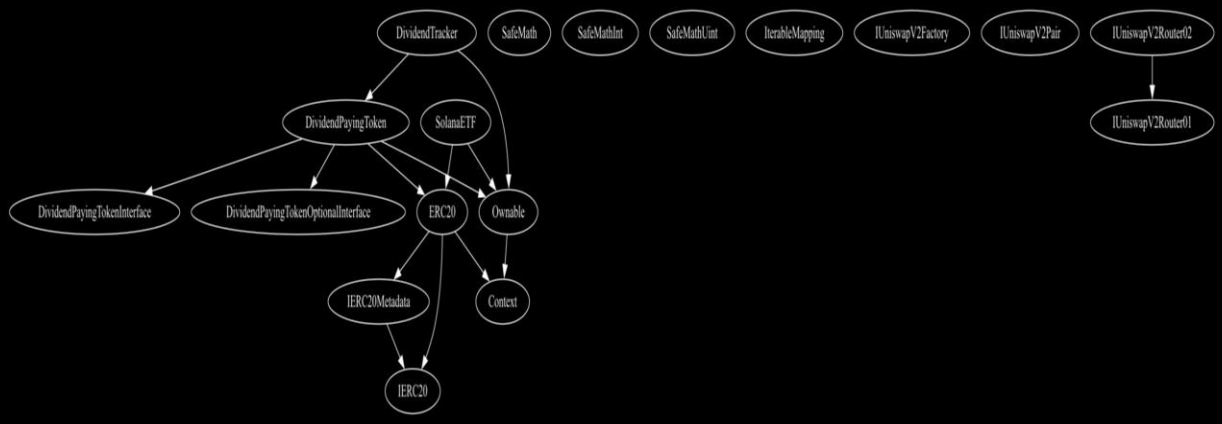
Contract Flow Graph



Contract Interaction Graph



Inheritance Graph



Contract Functions

Contract	Type	Bases		
L	Function Name	Visibility	Mutability	Modifiers
Context	Implementation			
L	_msgSender	Internal 🔒		
L	_msgData	Internal 🔒		
Ownable	Implementation	Context		
L		Public ❗	🛑	NO❗
L	owner	Public ❗		NO❗
L	renounceOwnership	Public ❗	🛑	onlyOwner
L	transferOwnership	Public ❗	🛑	onlyOwner
L	_transferOwnership	Internal 🔒	🛑	
SafeMath	Library			
L	add	Internal 🔒		
L	sub	Internal 🔒		
L	sub	Internal 🔒		
L	mul	Internal 🔒		
L	div	Internal 🔒		
L	div	Internal 🔒		
L	mod	Internal 🔒		
L	mod	Internal 🔒		
SafeMathInt	Library			
L	mul	Internal 🔒		

Contract	Type	Bases		
L	div	Internal 🔒		
L	sub	Internal 🔒		
L	add	Internal 🔒		
L	abs	Internal 🔒		
L	toUint256Safe	Internal 🔒		
SafeMathUint	Library			
L	toInt256Safe	Internal 🔒		
IterableMapping	Library			
L	get	Public ❗		NO❗
L	getIndexOfKey	Public ❗		NO❗
L	getKeyAtIndex	Public ❗		NO❗
L	size	Public ❗		NO❗
L	set	Public ❗	🛑	NO❗
L	remove	Public ❗	🛑	NO❗
IUniswapV2Factory	Interface			
L	feeTo	External ❗		NO❗
L	feeToSetter	External ❗		NO❗
L	getPair	External ❗		NO❗
L	allPairs	External ❗		NO❗
L	allPairsLength	External ❗		NO❗
L	createPair	External ❗	🛑	NO❗
L	setFeeTo	External ❗	🛑	NO❗
L	setFeeToSetter	External ❗	🛑	NO❗

Contract	Type	Bases		
IUniswapV2Pair	Interface			
L	name	External		NO
L	symbol	External		NO
L	decimals	External		NO
L	totalSupply	External		NO
L	balanceOf	External		NO
L	allowance	External		NO
L	approve	External		NO
L	transfer	External		NO
L	transferFrom	External		NO
L	DOMAIN_SEPARATOR	External		NO
L	PERMIT_TYPEHASH	External		NO
L	nonces	External		NO
L	permit	External		NO
L	MINIMUM_LIQUIDITY	External		NO
L	factory	External		NO
L	token0	External		NO
L	token1	External		NO
L	getReserves	External		NO
L	price0CumulativeLast	External		NO
L	price1CumulativeLast	External		NO
L	kLast	External		NO
L	mint	External		NO

Contract	Type	Bases		
L	burn	External		NO
L	swap	External		NO
L	skim	External		NO
L	sync	External		NO
L	initialize	External		NO
IUniswapV2Router01	Interface			
L	factory	External		NO
L	WETH	External		NO
L	addLiquidity	External		NO
L	addLiquidityETH	External		NO
L	removeLiquidity	External		NO
L	removeLiquidityETH	External		NO
L	removeLiquidityWithPermit	External		NO
L	removeLiquidityETHWithPermit	External		NO
L	swapExactTokensForTokens	External		NO
L	swapTokensForExactTokens	External		NO
L	swapExactETHForTokens	External		NO
L	swapTokensForExactETH	External		NO
L	swapExactTokensForETH	External		NO
L	swapETHForExactTokens	External		NO
L	quote	External		NO

Contract	Type	Bases		
L	getAmountOut	External !		NO!
L	getAmountIn	External !		NO!
L	getAmountsOut	External !		NO!
L	getAmountsIn	External !		NO!
IUniswapV2Router02	Interface	IUniswapV2Router01		
L	removeLiquidityETHSupportingFeeOnTransferTokens	External !		NO!
L	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External !		NO!
L	swapExactTokensForTokensSupportingFeeOnTransferTokens	External !		NO!
L	swapExactETHForTokensSupportingFeeOnTransferTokens	External !		NO!
L	swapExactTokensForETHSupportingFeeOnTransferTokens	External !		NO!
IERC20	Interface			
L	totalSupply	External !		NO!
L	balanceOf	External !		NO!
L	allowance	External !		NO!
L	transfer	External !		NO!
L	approve	External !		NO!
L	transferFrom	External !		NO!
IERC20Metadata	Interface	IERC20		

Contract	Type	Bases		
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!
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	_beforeTokenTransfer	Internal		
DividendPayingTokenInterface	Interface			
L	dividendOf	External !		NO!

Contract	Type	Bases		
L	withdrawDividend	External		NO!
DividendPayingTokenOptionalInterface	Interface			
L	withdrawableDividendOf	External		NO!
L	withdrawnDividendOf	External		NO!
L	accumulativeDividendOf	External		NO!
DividendPayingToken	Implementation	ERC20, Ownable, DividendPayingTokenInterface, DividendPayingTokenOptionalInterface		
L		Public		ERC20
L	distributeDividends	Public		onlyOwner
L	withdrawDividend	Public		NO!
L	_withdrawDividendOfUser	Internal		
L	dividendOf	Public		NO!
L	withdrawableDividendOf	Public		NO!
L	withdrawnDividendOf	Public		NO!
L	accumulativeDividendOf	Public		NO!
L	_transfer	Internal		
L	_mint	Internal		
L	_burn	Internal		
L	_setBalance	Internal		

Contract	Type	Bases		
DividendTracker	Implementation	Ownable, DividendPayingToken		
L		Public		DividendPayingToken
L	_transfer	Internal		
L	withdrawDividend	Public		NO!
L	updateMinimumTokenBalanceForDividends	External		onlyOwner
L	excludeFromDividends	External		onlyOwner
L	updateClaimWait	External		onlyOwner
L	setLastProcessedIndex	External		onlyOwner
L	getLastProcessedIndex	External		NO!
L	getNumberOfTokenHolders	External		NO!
L	getAccount	Public		NO!
L	getAccountAtIndex	Public		NO!
L	canAutoClaim	Private		
L	setBalance	External		onlyOwner
L	process	Public		NO!
L	processAccount	Public		onlyOwner
SolanaETF	Implementation	ERC20, Ownable		
L		Public		ERC20
L		External		NO!
L	claimStuckTokens	External		onlyOwner
L	isContract	Internal		

Contract	Type	Bases		
L	sendBNB	Internal		
L	_setAutomatedMarketMakerPair	Private		
L	excludeFromFees	External		onlyOwner
L	isExcludedFromFees	Public		NO!
L	_transfer	Internal		
L	swapAndSendDividends	Private		
L	setSwapTokensAtAmount	External		onlyOwner
L	updateClaimWait	External		onlyOwner
L	getClaimWait	External		NO!
L	getTotalDividendsDistributed	External		NO!
L	withdrawableDividendOf	Public		NO!
L	dividendTokenBalanceOf	Public		NO!
L	totalRewardsEarned	Public		NO!
L	excludeFromDividends	External		onlyOwner
L	getAccountDividendsInfo	External		NO!
L	getAccountDividendsInfoAtIndex	External		NO!
L	processDividendTracker	External		NO!
L	claim	External		NO!
L	claimAddress	External		onlyOwner
L	getLastProcessedIndex	External		NO!

Contract	Type	Bases		
L	setLastProcessedIndex	External !		onlyOwner
L	getNumberOfDividendTokenHolders	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 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