

SKELETON ECOSYSTEM

SMART CONTRACT AUDIT



OneSquare

OSE

ERC20

0x9EA8bc1e8b65b0d175Fb7D29130dee669dC7f921

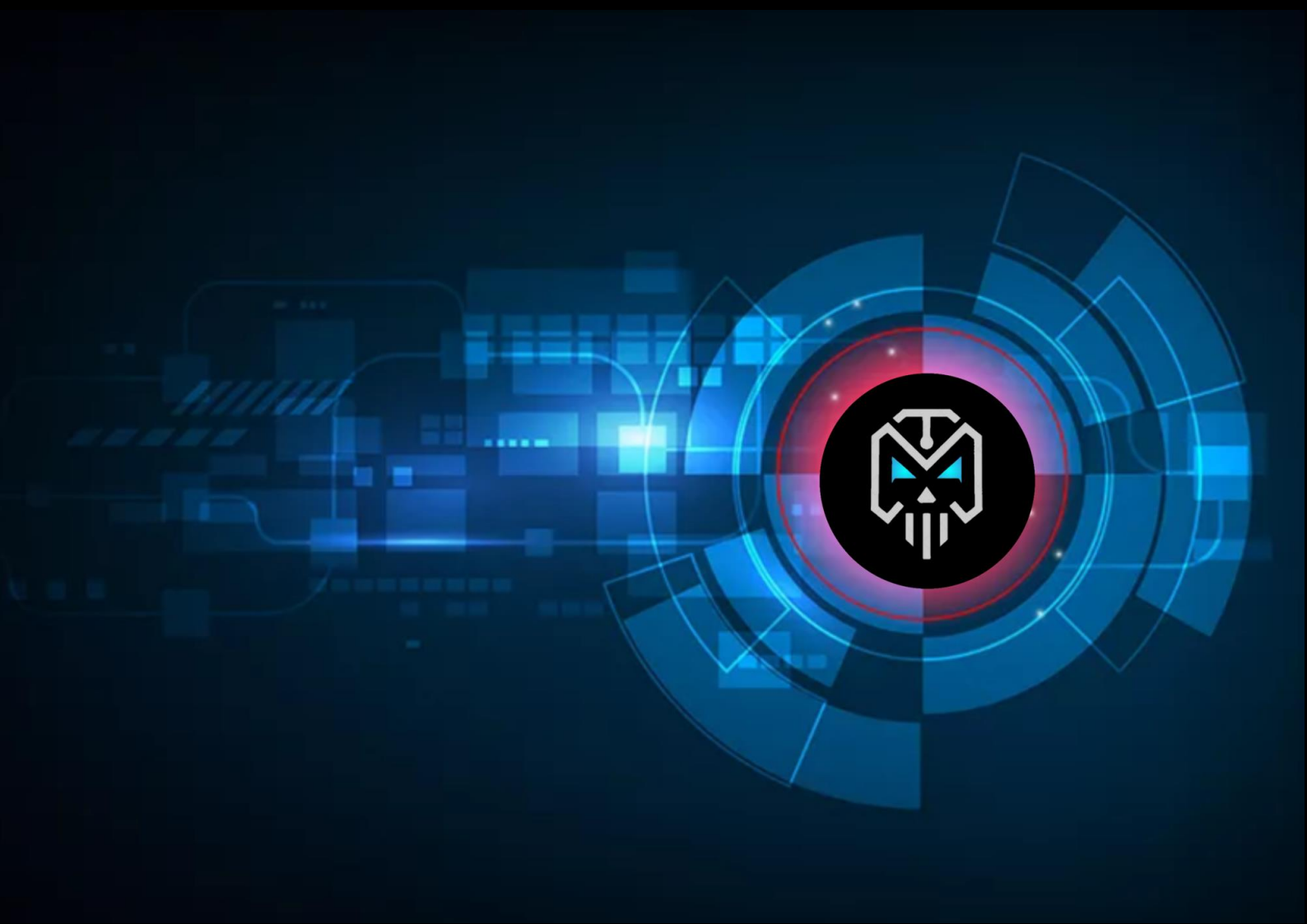


Table of Contents

Table of Contents	1
Disclaimer	2
Overview	3
Creation/Audit Date	3
Verified Socials	3
Contract Functions Analysis	4
Contract Safety and Weakness	8
Detected Vulnerability Description	12
Contract Flow Graph	17
Contract Interaction Graph	18
Inheritance Graph	19
Contract Descriptions	20
Audit Scope	26

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	One Square
Ticker/Symbol	OSE
Blockchain	Ethereum ERC20
Contract Address	0x9EA8bc1e8b65b0d175Fb7D29130dee669dC7f921
Creator Address	0x3Be4533bcEfFbAc5AA46eda38B55279D14C1c782
Current Owner Address	0x00
Contract Explorer	https://etherscan.io/token/0x9EA8bc1e8b65b0d175Fb7D29130dee669dC7f921#code
Compiler Version	v0.8.21+commit.d9974bed
License	MIT
Optimisation	No with 200 Runs
Total Supply	100,000,000 OSE
Decimals	18




Creation/Audit

Contract Deployed	30.07.2024
Audit Created	01.08.2024
Audit Update	V 1.0

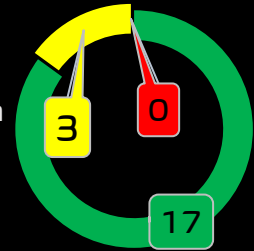
Verified Socials








Website	https://onesquare-erc.com
Telegram	https://t.me/onesquareerc
Twitter (X)	https://x.com/onesquareerc










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		0x00 Sometimes referred to as the "zero address" or "dead address" and is not owned by anyone.
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	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. 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 01.08.2024 Lp Locked: 99.00% UNCX for 183 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. . Contract renounced, function can not be triggered by owner. 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>Whitelist Setting function found.</p> <p>Contract renounced, function can not be triggered by owner.</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. Toggle remove limits. Max wallet and max transfer = Totalsupply</p> <p>Contract renounced, function can not be triggered by owner.</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

Removing Risk of contract function based on renounced ownership

Transaction Receipt Event Logs

235

Address 0x9ea8bc1e8b65b0d175fb7d29130dee669dc7f921   

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

Topics

0

0x8be0079c531659141344cd1fd0a4f28419497f9722a3daafe3b4186f6b6457e0

1: previousOwner

Dec ▾

→ 0x38e4533bcEffBac5AA46eda38B55279D14C1c782

2: newOwner

Dec ▾

→ 0x00

Data 0x

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

Set Fee

Contract renounced, function can not be triggered by owner.

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
565     function updateFees(uint256 _buy!, uint256 _sell!) external onlyOwner {
566         require(_buy! <= 5, "Exceed the limit");
567         require(_sell! <= 5, "Exceed the limit");
568         buyTotalFees = _buy!;
569         sellTotalFees = _sell!;
570     }
571

```

⚠️ Max Transaction and Holding Modify function

Contract renounced, function can not be triggered by owner.

Toggle remove limits. Max wallet and max transfer = Totalsupply

```
fttrace | funcSig
504 function updateMaxTxnAmount(uint256 newNum!) external onlyOwner {
505     require(newNum! >= ((totalSupply() * 1) / 1000) / 1e18, "Cannot set maxTransactionAmount lower than 0.1%");
506     maxTransactionAmount = newNum! * (10 ** 18);
507 }
508
fttrace | funcSig
509 function updateMaxWalletAmount(uint256 newNum!) external onlyOwner {
510     require(newNum! >= ((totalSupply() * 5) / 1000) / 1e18, "Cannot set maxWallet lower than 0.5%");
511     maxWallet = newNum! * (10 ** 18);
512 }
```

⚠️ Whitelist Function

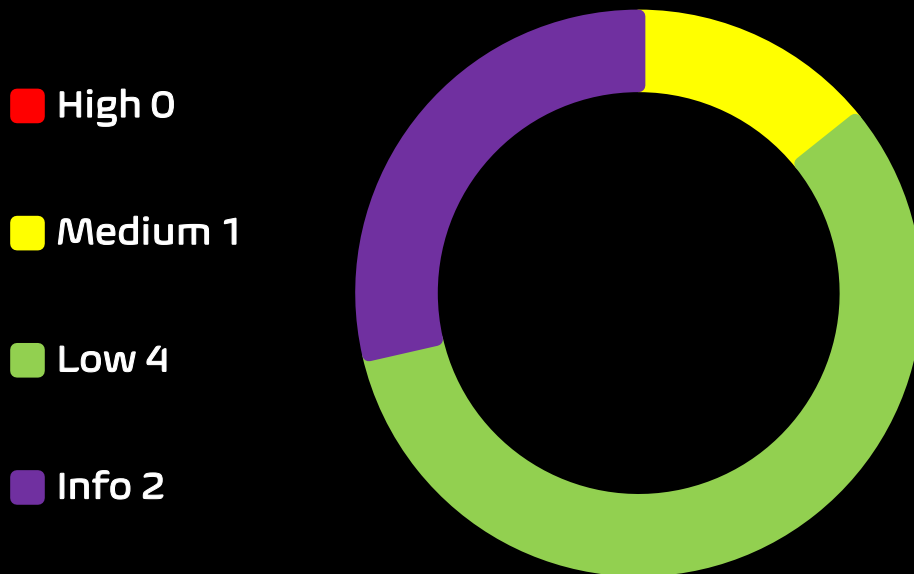
Contract renounced, function can not be triggered by owner.

If there is a function for this Developer can set zero fee or no max wallet size for addresses (for example team wallets can trade without fee. Can cause farming)

```
fttrace | funcSig
529 function excludeFromFees(address account!, bool excluded!) public onlyOwner {
530     _isExcludedFromFees[account!] = excluded!;
531     emit ExcludeFromFees(account!, excluded!);
532 }
533
```


Contract Security

Total Findings: 7



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)

- Incorrect Acces Control

Low severity issues: (4)

- Long number literals
- Outdated Compiler Version
- Missing Events
- Approve Front running attacks

Informational severity issues: (2)

- Public Functions Should be Declared External
- Uninitialized Local Variables

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	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	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.

⚠️ Incorrect Acces Control [2 Item]

Item: 1	Location:	Line 534-538	Severity:	Medium
---------	-----------	--------------	-----------	--------

Function	<p>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.</p> <p>The contract OSE is importing an access control library @openzeppelin/contracts/access/Ownable.sol but the function manualswap is missing the modifier onlyOwner.</p>
Remediation	<ol style="list-style-type: none"> 1. Ensure that initialization functions can only be called once and only by authorized entities. 2. Implement least-privilege roles using libraries like OpenZeppelin's Access Control. 3. Add proper access control modifiers to sensitive functions, such as onlyOwner or custom roles.

```

534      ftrace | funcSig
535      function manualswap(uint256 amount) external {
536          require(_msgSender() == marketingWallet);
537          require(amount <= balanceOf(address(this)) && amount > 0, "Wrong amount");
538          swapTokensForEth(amount);
539      }
  
```

Item: 2	Location:	Line 540-543	Severity: ■ Medium
---------	-----------	--------------	---

Function	<p>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.</p> <p>The contract OSE is importing an access control library @openzeppelin/contracts/access/Ownable.sol but the function manualsend is missing the modifier onlyOwner.</p>
Remediation	<ol style="list-style-type: none"> 1. Ensure that initialization functions can only be called once and only by authorized entities. 2. Implement least-privilege roles using libraries like OpenZeppelin's Access Control. 3. Add proper access control modifiers to sensitive functions, such as onlyOwner or custom roles.

```

ftrace | funcSig
540  function manualsend() external {
541      bool success;
542      (success,) = address(marketingWallet).call(value: address(this).balance)("");
543  }
544

```

Item: 3	Location:	Line 577-584	Severity: ■ Medium
---------	-----------	--------------	---

Function	<p>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.</p> <p>The contract OSE is importing an access control library @openzeppelin/contracts/access/Ownable.sol but the function airdrop is missing the modifier onlyOwner.</p>
Remediation	<ol style="list-style-type: none"> 1. Ensure that initialization functions can only be called once and only by authorized entities. 2. Implement least-privilege roles using libraries like OpenZeppelin's Access Control. 3. Add proper access control modifiers to sensitive functions, such as onlyOwner or custom roles.

```

577   ftrace | funcSig
578   function airdrop(address[] calldata addresses!, uint256[] calldata amounts!) external {
579       require(addresses!.length > 0 && amounts!.length == addresses!.length);
580       address from = msg.sender;
581
582       for (uint256 i = 0; i < addresses!.length; i++) {
583           _transfer(from, addresses![i], amounts![i] * (10 ** 18));
584       }
585   }
  
```

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

Item: 1	Location:	299-302	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.</p> <p>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.</p>
Remediation	<p>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.</p> <p>2.Use transaction taxes to prevent against front-run attack</p>

```

ttrace | funcsig
299     function approve(address spender, uint256 amount) public virtual override returns (bool) {
300         _approve(_msgSender(), spender, amount);
301         return true;
302     }
303
  
```


Item: 2	Location:	304-314	Severity: ■ Low
---------	-----------	---------	--

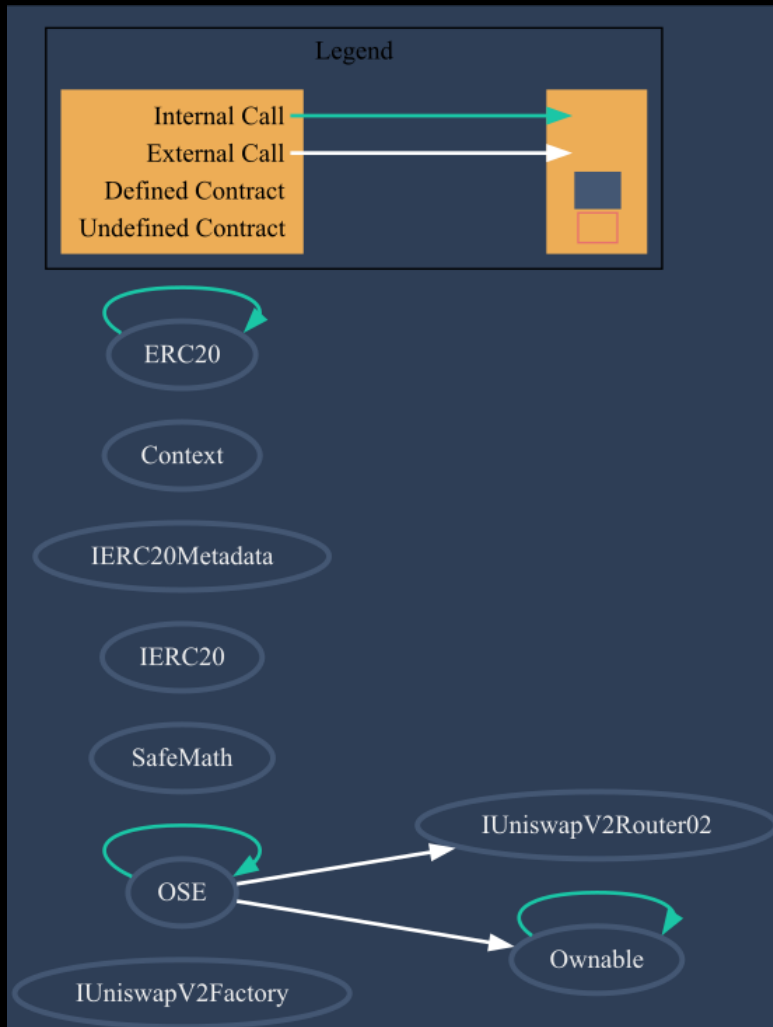
Function	<p>The transferFrom() method overrides current allowance regardless of whether the spender already used it or not, so there is no way to increase or decrease allowance by a certain value atomically unless the token owner is a smart contract, not an account.</p> <p>This can be abused by a token 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 transferFrom 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

```

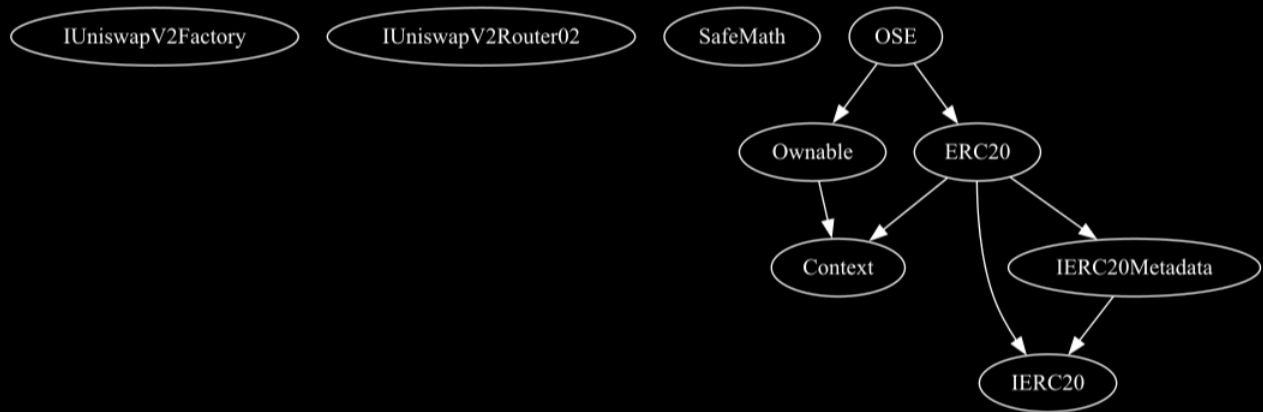
304  |  ftrace | funcSig
305  |  function transferFrom(address sender!, address recipient!, uint256 amount!) public virtual override returns (bool) {
306  |      _transfer(sender!, recipient!, amount!);
307  |
308  |      uint256 currentAllowance = _allowances[sender!][_msgSender()];
309  |      require(currentAllowance >= amount!, "ERC20: transfer amount exceeds allowance");
310  |      unchecked {
311  |          _approve(sender!, _msgSender(), currentAllowance - amount!);
312  |      }
313  |
314  |      return true;
315  |  }
  
```

[illegible]

































Contract Interaction Graph



Inheritance Graph





















Contract Functions














Contract	Type	Bases		
	Function Name	Visibility	Mutability	Modifiers
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 
IUniswapV2Router02	Interface			
	factory	External 		NO 
	WETH	External 		NO 
	addLiquidity	External 		NO 
	addLiquidityETH	External 		NO 
	swapExactTokensForTokensSupportingFees	External 		NO 

	eeOnTransferTokens			
L	swapExactETHForTokensSupportingFeeOnTransferTokens	External !		NO!
L	swapExactTokensForETHSupportingFeeOnTransferTokens	External !		NO!
SafeMath	Library			
L	tryAdd	Internal 		
L	trySub	Internal 		
L	tryMul	Internal 		
L	tryDiv	Internal 		
L	tryMod	Internal 		
L	add	Internal 		
L	sub	Internal 		
L	mul	Internal 		
L	div	Internal 		
L	mod	Internal 		
L	sub	Internal 		
L	div	Internal 		
L	mod	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 ?
Context	Implementation			
└	_msgSender	Internal 🔒		
└	_msgData	Internal 🔒		
Ownable	Implementation	Context		
└		Public ?	⬢	NO ?
└	owner	Public ?		NO ?
└	renounceOwnership	Public ?	⬢	onlyOwner
└	transferOwnership	Public ?	⬢	onlyOwner
└	_transferOwnership	Internal 🔒	⬢	

ERC20	Implementati on	Context, IERC20, IERC20Metad ata		
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	increaseAllow ance	Public 		NO 
L	decreaseAllo wance	Public 		NO 
L	_transfer	Internal 		
L	_mint	Internal 		
L	_burn	Internal 		
L	_approve	Internal 		
L	_beforeToken Transfer	Internal 		
L	_afterTokenTr ansfer	Internal 		

OSE	Implementati on	ERC20, Ownable		
L		Public 		ERC20
L		External 		NO 
L	AddLP	External 		onlyOwner
L	OpenTrading	External 		onlyOwner
L	removeLimits	External 		onlyOwner
L	updateSwapT okensAtAmou nt	External 		onlyOwner
L	updateMaxS wap	External 		onlyOwner
L	updateMaxTx nAmount	External 		onlyOwner
L	updateMaxW alletAmount	External 		onlyOwner
L	whitelistCont ract	Public 		onlyOwner
L	excludeFrom MaxTransacti on	Public 		onlyOwner
L	updateSwapE nabled	External 		onlyOwner
L	excludeFromF ees	Public 		onlyOwner
L	manualswap	External 		NO 
L	manualsend	External 		NO 
L	setAutomate dMarketMake rPair	Public 		onlyOwner

L	_setAutomatedMarketMakerPair	Private 		
L	updateFees	External 		onlyOwner
L	updateMarketingWallet	External 		onlyOwner
L	airdrop	External 		NO 
L	_transfer	Internal 		
L	swapTokensForEth	Private 		
L	swapBack	Private 		



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>

