auditace.tech

AuditAce
FROM INCEPTION TO SUCCESS

Smart Contract Audit

FOR

FSEC

DATED : 12 June 23'

HIGH RISK FINDING

Centralization – Trades must be enabled

Severity: High

function: enableTrading

Status: Not Resolved

Overview:

The smart contract owner must enable trades for holders. If trading remain disabled, no one would be able to buy/sell/transfer tokens.

```
function enableTrading() external onlyOwner {
    require(!tradingEnabled, "Trading already enabled.");
    tradingEnabled = true;
    swapEnabled = true;
}
```

Suggestion

To mitigate this centralization issue, we propose the following options:

1. Renounce Ownership: Consider relinquishing control of the smart contract by renouncing ownership. This would remove the ability for a single entity to manipulate the router, reducing centralization risks.
2. Multi-signature Wallet: Transfer ownership to a multi-signature wallet. This would require multiple approvals for any changes to the mainRouter, adding an additional layer of security and reducing the centralization risk.

Transfer ownership to a trusted and valid 3rd party in order to guarantee enabling of the trades

AUDIT SUMMARY

Project name – FSEC

Date: 12 June, 2023

Scope of Audit- Audit Ace was consulted to conduct the smart contract audit of the solidity source codes.

Audit Status: Passed

Issues Found

Status	Critical	High	Medium	Low	Suggestion
Open	0	1	1	1	0
Acknowledged	0	0	0	0	0
Resolved	0	0	0	0	0

USED TOOLS

Tools:

1- Manual Review:
A line by line code review has been performed by audit ace team.

2- BSC Test Network: All tests were conducted on the BSC Test network, and each test has a corresponding transaction attached to it. These tests can be found in the "Functional Tests" section of the report.

3- Slither :
The code has undergone static analysis using Slither.

Testnet version:
Contract has been tested on binance smart chain testnet which can be found in below link:
https://testnet.bscscan.com/address/0x06f4c42a5e8ca68a0dcd869817b8013223f9a3d8

Token Information

Token Name : Fuck the SEC _|_

Token Symbol: FSEC

Decimals: 18

Token Supply: 1, 000, 000

Token Address:
0x1f0369B97f4d3198458Dcd21720F2E32C832F8F9

Checksum:
c78180bde5428270947152dae3633e24a9938b67

Owner:
0xD80Cd37587f1B95681C9B24ab535E7C7eb2105Bb

Deployer:
0xD80Cd37587f1B95681C9B24ab535E7C7eb2105Bb

TOKEN OVERVIEW

Fees:

Buy Fees: 5%

Sell Fees: 5%

Transfer Fees: 5%

Fees Privilege: Static fees

Ownership: Owned

Minting: None

Max Tx Amount/ Max Wallet Amount: No

Blacklist: No

Other Privileges: - initial distribution of the token

- enabling trades manually

AUDIT METHODOLOGY

The auditing process will follow a routine as special considerations by Auditace:

- Review of the specifications, sources, and instructions provided to Auditace to make sure the contract logic meets the intentions of the client without exposing the user's funds to risk.
- Manual review of the entire codebase by our experts, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
- Specification comparison is the process of checking whether the code does what the specifications, sources, and instructions provided to Auditace describe.
- Test coverage analysis determines whether the test cases are covering the code and how much code isexercised when we run the test cases.
- Symbolic execution is analysing a program to determine what inputs cause each part of a program to execute.
- Reviewing the codebase to improve maintainability, security, and control based on the established industry and academic practices.

VULNERABILITY CHECKLIST

- [x] Return values of low-level calls
- [x] Private modifier
- [x] Multiple Sends
- [x] Using Suicide
- [x] Gas Limitand Loops
- [x] Address hardcoded
- [x] Exception Disorder
- [x] Using inline assembly
- [x] Divide before multiply
- [x] Missing Zero Address Validation
- [x] Compiler version not fixed
- [x] **Gasless Send**
- [x] Using block.timestamp
- [x] Re-entrancy
- [x] Tautology or contradiction
- [x] Timestamp Dependence
- [x] Revert/require functions
- [x] Use of tx.origin
- [x] Integer overflow/underflow
- [x] Dangerous strict equalities
- [x] Using SHA3
- [x] Using throw

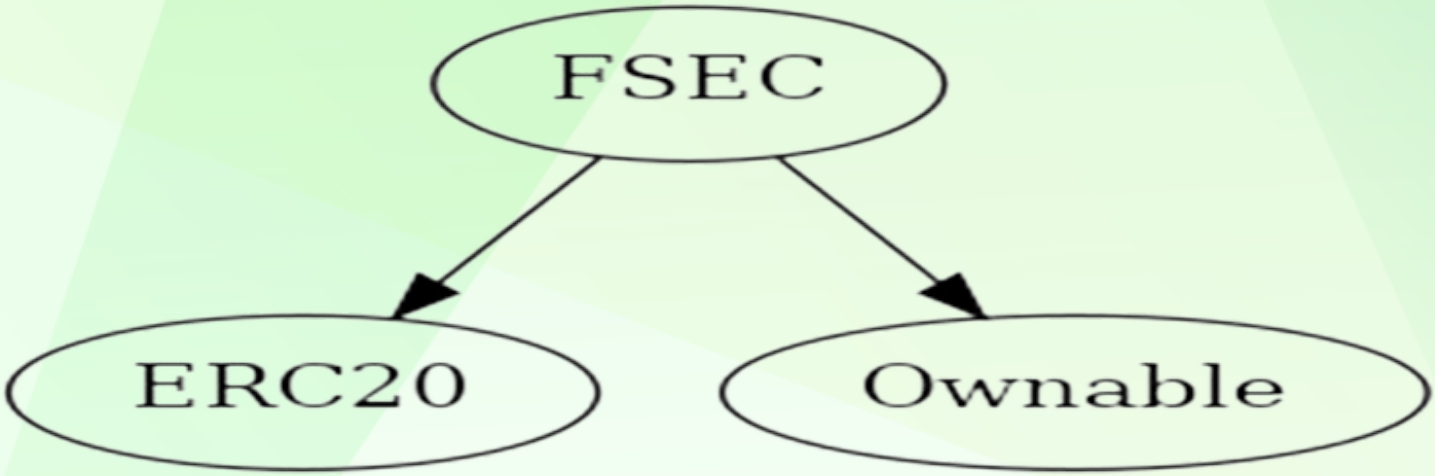
CLASSIFICATION OF RISK

Severity	Description
Critical	These vulnerabilities could be exploited easily and can lead to asset loss, data loss, asset, or data manipulation. They should be fixed right away.
High-Risk	A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.
Medium-Risk	A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.
Low-Risk	A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.
Gas Optimization /Suggestion	A vulnerability that has an informational character but is not affecting any of the code.

Findings

Severity	Found
Critical	0
High-Risk	1
Medium-Risk	1
Low-Risk	1
Gas Optimization / Suggestions	0

INHERITANCE TREE



POINTS TO NOTE

- Owner is not able to change buy/sell fees (5% each)
- Owner is not able to set fee on transfers (0%)
- Owner is not able to set max buy/sell/transfer/hold amount
- Owner is not able to blacklist an arbitrary wallet
- Owner is not able to mint new tokens
- Owner is not able to disable trades
- Owner has 100% of total supply after deployment
- Owner must enable trades manually

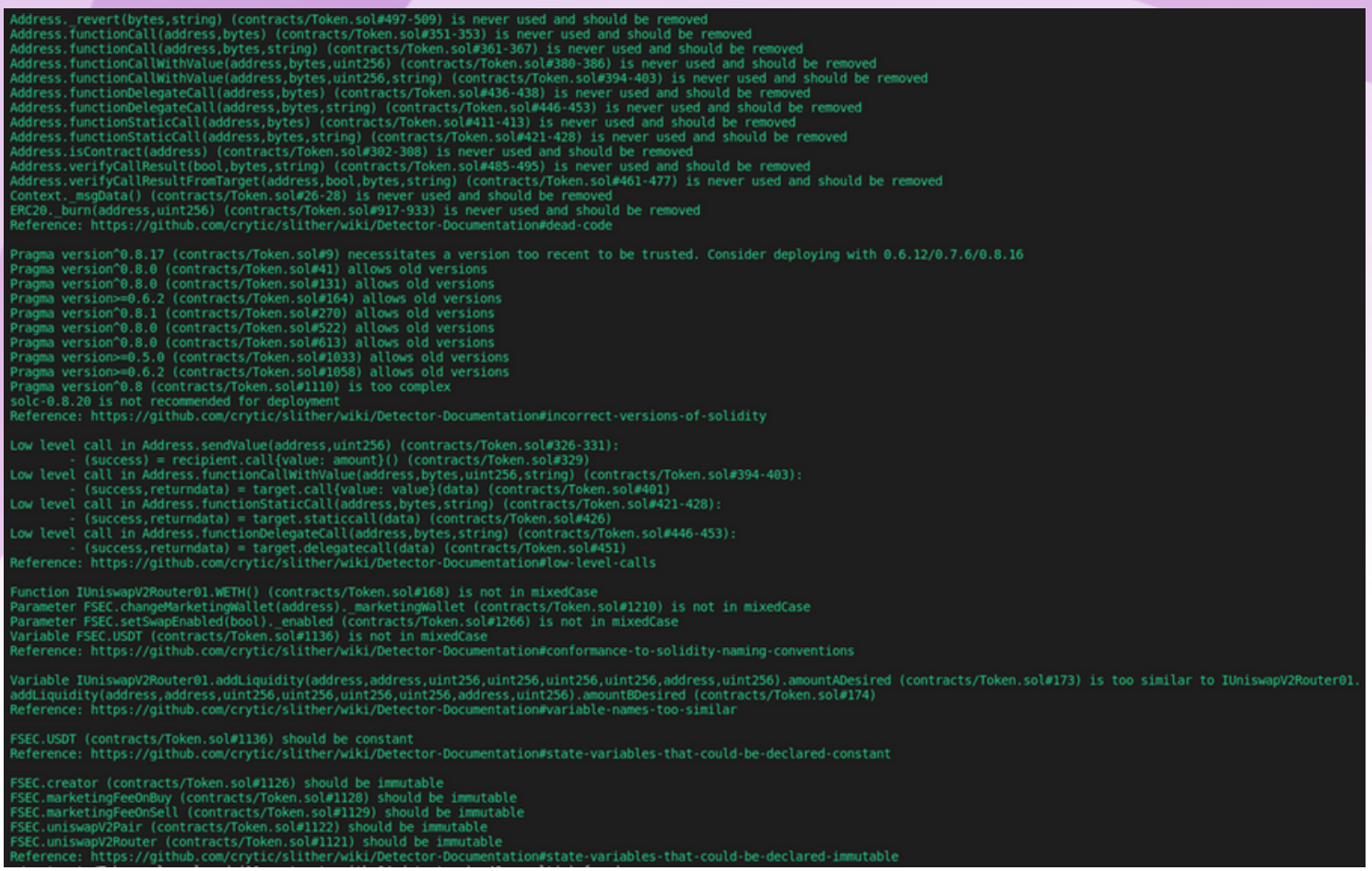
CONTRACT ASSESMENT

Contract	Type	Bases		
└	**Function Name**	**Visibility**	**Mutability**	**Modifiers**
FSEC	Implementation	ERC20, Ownable		
└	<Constructor>	Public ❗	🛑	ERC20
└	<Receive Ether>	External ❗	💵	NO ❗
└	claimStuckTokens	External ❗	🛑	onlyOwner
└	excludeFromFees	External ❗	🛑	onlyOwner
└	isExcludedFromFees	Public ❗		NO ❗
└	changeMarketingWallet	External ❗	🛑	onlyOwner
└	enableTrading	External ❗	🛑	onlyOwner
└	_transfer	Internal 🔒	🛑	
└	setSwapEnabled	External ❗	🛑	onlyOwner
└	setSwapTokensAtAmount	External ❗	🛑	onlyOwner
└	swapAndSendMarketing	Private 🔐	🛑	

Legend

Symbol	Meaning
🛑	Function can modify state
💵	Function is payable

STATIC ANALYSIS



```
Address._revert(bytes,string) (contracts/Token.sol#497-509) is never used and should be removed
Address.functionCall(address,bytes) (contracts/Token.sol#351-353) is never used and should be removed
Address.functionCall(address,bytes,string) (contracts/Token.sol#361-367) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (contracts/Token.sol#380-386) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256,string) (contracts/Token.sol#394-403) is never used and should be removed
Address.functionDelegateCall(address,bytes) (contracts/Token.sol#436-438) is never used and should be removed
Address.functionDelegateCall(address,bytes,string) (contracts/Token.sol#446-453) is never used and should be removed
Address.functionStaticCall(address,bytes) (contracts/Token.sol#411-413) is never used and should be removed
Address.functionStaticCall(address,bytes,string) (contracts/Token.sol#421-428) is never used and should be removed
Address.isContract(address) (contracts/Token.sol#302-308) is never used and should be removed
Address.verifyCallResult(bool,bytes,string) (contracts/Token.sol#485-495) is never used and should be removed
Address.verifyCallResultFromTarget(address,bool,bytes,string) (contracts/Token.sol#461-477) is never used and should be removed
Context._msgData() (contracts/Token.sol#26-28) is never used and should be removed
ERC20._burn(address,uint256) (contracts/Token.sol#917-933) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.8.17 (contracts/Token.sol#9) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.16
Pragma version^0.8.0 (contracts/Token.sol#41) allows old versions
Pragma version^0.8.0 (contracts/Token.sol#131) allows old versions
Pragma version>=0.6.2 (contracts/Token.sol#164) allows old versions
Pragma version^0.8.1 (contracts/Token.sol#270) allows old versions
Pragma version^0.8.0 (contracts/Token.sol#522) allows old versions
Pragma version^0.8.0 (contracts/Token.sol#613) allows old versions
Pragma version>=0.5.0 (contracts/Token.sol#1033) allows old versions
Pragma version>=0.6.2 (contracts/Token.sol#1058) allows old versions
Pragma version^0.8 (contracts/Token.sol#1110) is too complex
solc-0.8.20 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in Address.sendValue(address,uint256) (contracts/Token.sol#326-331):
        - (success) = recipient.call{value: amount}() (contracts/Token.sol#329)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (contracts/Token.sol#394-403):
        - (success,returndata) = target.call{value: value}(data) (contracts/Token.sol#401)
Low level call in Address.functionStaticCall(address,bytes,string) (contracts/Token.sol#421-428):
        - (success,returndata) = target.staticcall(data) (contracts/Token.sol#426)
Low level call in Address.functionDelegateCall(address,bytes,string) (contracts/Token.sol#446-453):
        - (success,returndata) = target.delegatecall(data) (contracts/Token.sol#451)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

Function IUniswapV2Router01.WETH() (contracts/Token.sol#168) is not in mixedCase
Parameter FSEC.changeMarketingWallet(address)._marketingWallet (contracts/Token.sol#1210) is not in mixedCase
Parameter FSEC.setSwapEnabled(bool)._enabled (contracts/Token.sol#1266) is not in mixedCase
Variable FSEC.USDT (contracts/Token.sol#1136) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Variable IUniswapV2Router01.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amountADesired (contracts/Token.sol#173) is too similar to IUniswapV2Router01.
addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amountBDesired (contracts/Token.sol#174)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-too-similar

FSEC.USDT (contracts/Token.sol#1136) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant

FSEC.creator (contracts/Token.sol#1126) should be immutable
FSEC.marketingFeeOnBuy (contracts/Token.sol#1128) should be immutable
FSEC.marketingFeeOnSell (contracts/Token.sol#1129) should be immutable
FSEC.uniswapV2Pair (contracts/Token.sol#1122) should be immutable
FSEC.uniswapV2Router (contracts/Token.sol#1121) should be immutable
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-immutable
```

Result => A static analysis of contract's source code has been performed using slither,

No major issues were found in the output

FUNCTIONAL TESTING

1- Adding liquidity (passed):

https://testnet.bscscan.com/tx/0xdacc02589ada1cb2e364e46431c9d522afa3b8282e6eb62e33cb0135fe097f7d

2- Buying when excluded from fees (0% tax) (passed):

https://testnet.bscscan.com/tx/0x29f60a69b9103ff9ff9eb2eafa2064bcb6715ff8cc7cfd28abb4cc0e5c0c29df

3- Selling when excluded from fees (0% tax) (passed):

https://testnet.bscscan.com/tx/0xd2c9974f920ee23c650c3705e6ddda1a9ca28fbaebc214abf06cdb25a54cc943

4- Transferring when excluded from fees (0% tax) (passed):

https://testnet.bscscan.com/tx/0x748e274408d4cfc306ca28483cfa32ef257e7405cdaf9214650017b1956911c6

5- Buying when not excluded from fees (5% tax) (passed):

https://testnet.bscscan.com/tx/0x039368f0a60ac0b4de93f9fc2fae94a88952738907c0e170cf5954c91eaa7874

6- Selling when not excluded from fees (5% tax) (passed):

https://testnet.bscscan.com/tx/0xd99852d8045dec96558001013abdab171438a1d2b3d0feb23c0f1f6a6813ea64

FUNCTIONAL TESTING

7- Transferring when not excluded from fees (0% tax) (passed):

https://testnet.bscscan.com/tx/0x4efda3b792b4ba3ba80482ba89f51fa26d30e4f184aaab6b1fe50b9177c280ce

8- Internal swap (Fees => USDT => Marketing wallet) (passed):

https://testnet.bscscan.com/tx/0xd99852d8045dec96558001013abdab171438a1d2b3d0feb23c0f1f6a6813ea64

FUNCTIONAL TESTING

Centralization – Trades must be enabled

Severity: High

function: enableTrading

Status: Not Resolved

Overview:

The smart contract owner must enable trades for holders. If trading remain disabled, no one would be able to buy/sell/transfer tokens.

```
function enableTrading() external onlyOwner {
    require(!tradingEnabled, "Trading already enabled.");
    tradingEnabled = true;
    swapEnabled = true;
}
```

Suggestion

To mitigate this centralization issue, we propose the following options:

1. Renounce Ownership: Consider relinquishing control of the smart contract by renouncing ownership. This would remove the ability for a single entity to manipulate the router, reducing centralization risks.
2. Multi-signature Wallet: Transfer ownership to a multi-signature wallet. This would require multiple approvals for any changes to the mainRouter, adding an additional layer of security and reducing the centralization risk.

Transfer ownership to a trusted and valid 3rd party in order to guarantee enabling of the trades

FUNCTIONAL TESTING

Potential Swap Failure – Incompatibility with Fee-On-Transfer Tokens

Severity: Medium

Function: swapAndSendMarketing

Status: Not Resolved

Overview: The contract utilizes the swapExactTokensForTokens function of Uniswap Router for swapping tokens to USDT. This mechanism could potentially fail with tokens that implement transfer fees within their contracts, also known as Fee-On-Transfer tokens. When the swapExactTokensForTokens function is called, it may not account for the fees taken out during the transfer, resulting in either transaction failure or less output tokens than expected.

The swapExactTokensForTokens function is used in the swapAndSendMarketing function in the contract.

```
function swapAndSendMarketing(uint256 tokenAmount) private {
    address[] memory path = new address[](2);
    path[0] = address(this);
    path[1] = address(USDT);

    uniswapV2Router.swapExactTokensForTokens(tokenAmount, 0, path, marketingWallet, block.timestamp);

    emit SwapAndSendMarketing(tokenAmount, tokenAmount);
}
```

Suggestion: To mitigate this potential swap failure risk, we propose the following solutions:

1. **Use** swapExactTokensForTokensSupportingFeeOnTransferTokens**:** Consider using the swapExactTokensForTokensSupportingFeeOnTransferTokens function provided by the Uniswap Router instead. This function is designed to handle tokens with transfer fees and will ensure the swap will function as expected.
2. **Exclude Contract from Fees:** Another option would be to exclude the contract from fees when it's doing the swapping. This would prevent the fees from being deducted during the token transfer, allowing the swapExactTokensForTokens function to work as expected. Also ensure that contract can not be included in fees later

FUNCTIONAL TESTING

Missing logic – Static tax

Severity: Informational

Status: Not Resolved

Overview:

Current fees can not be changed later, owner of the token might need to change fees based on different market condition. (e.g. decrease buy tax to encourage new investors)

Suggestion

Its suggested to have a function that enables owner to change fees in a safe range:

0 <= buy tax <= 10

0 <= sell tax <= 10

0 <= transfer tax <= 10

DISCLAIMER

All the content provided in this document is for general information only and should not be used as financial advice or a reason to buy any investment. Team provides no guarantees against the sale of team tokens or the removal of liquidity by the project audited in this document. Always Do your own research and protect yourselves from being scammed. The Auditace team has audited this project for general information and only expresses their opinion based on similar projects and checks from popular diagnostic tools. Under no circumstances did Auditace receive a payment to manipulate those results or change the awarding badge that we will be adding in our website. Always Do your own research and protect yourselves from scams. This document should not be presented as a reason to buy or not buy any particular token. The Auditace team disclaims any liability for the resulting losses.

ABOUT AUDITACE

We specializes in providing thorough and reliable audits for Web3 projects. With a team of experienced professionals, we use cutting-edge technology and rigorous methodologies to evaluate the security and integrity of blockchain systems. We are committed to helping our clients ensure the safety and transparency of their digital assets and transactions.

https://auditace.tech/

https://t.me/Audit_Ace

https://twitter.com/auditace_

https://github.com/Audit-Ace