Cyberscope

# Audit Report

# Social.fans

March 2024

| | |
|---|---|
| Network | ETH |
| Address | 0x3cdf8020469abf01ce9b1b9cf1f1f695d0bb91c2 |
| Audited by | © cyberscope |

# Analysis

Critical Medium Minor / Informative Pass

| Severity | Code | Description | Status |
|---|---|---|---|
| ● | ST | Stops Transactions | Passed |
| ● | OTUT | Transfers User's Tokens | Passed |
| ● | ELFM | Exceeds Fees Limit | Passed |
| ● | MT | Mints Tokens | Passed |
| ● | BT | Burns Tokens | Passed |
| ● | BC | Blacklists Addresses | Passed |

# Diagnostics

Critical Medium Minor / Informative

| Severity | Code | Description | Status |
|---|---|---|---|
| ● | IEH | Inconsistent ETH Handling | Unresolved |
| ● | PLPI | Potential Liquidity Provision Inadequacy | Unresolved |
| ● | PVC | Price Volatility Concern | Unresolved |
| ● | RSW | Redundant Storage Writes | Unresolved |
| ● | RSFU | Redundant Struct Fee Usage | Unresolved |
| ● | L04 | Conformance to Solidity Naming Conventions | Unresolved |
| ● | L08 | Tautology or Contradiction | Unresolved |
| ● | L09 | Dead Code Elimination | Unresolved |
| ● | L16 | Validate Variable Setters | Unresolved |

# Table of Contents

# Review

| | |
|---|---|
| **Contract Name** | FANS |
| **Compiler Version** | v0.8.24+commit.e11b9ed9 |
| **Optimization** | 200 runs |
| **Explorer** | https://etherscan.io/address/0x3cdf8020469abf01ce9b1b9cf1f1f695d0bb91c2 |
| **Address** | 0x3cdf8020469abf01ce9b1b9cf1f1f695d0bb91c2 |
| **Network** | ETH |
| **Symbol** | FANS |
| **Decimals** | 18 |
| **Total Supply** | 21,000,000 |
| **Badge Eligibility** | Yes |

## Audit Updates

| | |
|---|---|
| **Initial Audit** | 22 Mar 2024 |

## Source Files

| | |
|---|---|
| **Filename** | SHA256 |
| **FANS.sol** | 5bcc1a389d20908c516180eb8245d5ac09eb57d9b07296933fe3aa2f60523133 |

# Findings Breakdown


| | |
|---|---|
| Critical | 0 |
| Medium | 0 |
| Minor / Informative | 9 |

| Severity | Unresolved | Acknowledged | Resolved | Other |
|---|---|---|---|---|
| Critical | 0 | 0 | 0 | 0 |
| Medium | 0 | 0 | 0 | 0 |
| Minor / Informative | 9 | 0 | 0 | 0 |

# IEH - Inconsistent ETH Handling

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | FANS.sol#L252,255,347 |
| **Status** | Unresolved |

## Description

The contract manages Ethereum balance transfers in two distinct segments with differing methodologies for sending ETH to external addresses. In the first code segment, the contract calculates the amounts for `marketing` and `development` based on the delta of the contract's balance before and after that is using the `sendValue` function for transferring ETH. This approach ensures safer error handling and gas management. Conversely, the second code segment directly calculates the ETH balance to be divided between marketing and development, employing a lower-level call method for transfers without reusing the `sendValue` function. This inconsistency not only increases the complexity and potential for errors within the contract but also overlooks the benefits of reusing `sendValue` for improved error handling and gas efficiency.

```
uint256 deltaBalance = address(this).balance - initialBalance;
uint256 marketingAmt = deltaBalance * 50 / 100;
uint256 developmentAmt = deltaBalance - marketingAmt;
if(marketingAmt > 0){
    payable(marketingWallet).sendValue(marketingAmt);
}
if(developmentAmt > 0){
    payable(developmentWallet).sendValue(developmentAmt);
}

...
bool success;
(success, ) = address(marketingWallet).call{value: ethMarketing}("");
(success, ) = address(developmentWallet).call{value:
ethDevelopment}("");
```

## Recommendation

It is recommended to standardize the method of handling ETH transfers within the contract to ensure consistency, safety, and gas efficiency. Specifically, reusing the `sendValue` function across all segments requiring ETH transfers can leverage its built-in error handling and gas optimizations. This approach simplifies the contract's logic. Refactoring the second code segment to utilize the `sendValue` function, as demonstrated in the first segment, will align the contract's ETH handling practices, enhancing its overall security and efficiency.

## PLPI - Potential Liquidity Provision Inadequacy

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | FANS.sol#L259 |
| **Status** | Unresolved |

### Description

The contract operates under the assumption that liquidity is consistently provided to the pair between the contract's token and the native currency. However, there is a possibility that liquidity is provided to a different pair. This inadequacy in liquidity provision in the main pair could expose the contract to risks. Specifically, during eligible transactions, where the contract attempts to swap tokens with the main pair, a failure may occur if liquidity has been added to a pair other than the primary one. Consequently, transactions triggering the swap functionality will result in a revert.

```
    function swapTokensForETH(uint256 tokenAmount) private {
        address[] memory path = new address[](2);
        path[0] = address(this);
        path[1] = router.WETH();
        _approve(address(this), address(router), tokenAmount);

router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,
0, path, address(this), block.timestamp);
    }
```

### Recommendation

The team is advised to implement a runtime mechanism to check if the pair has adequate liquidity provisions. This feature allows the contract to omit token swaps if the pair does not have adequate liquidity provisions, significantly minimizing the risk of potential failures.

Furthermore, the team could ensure the contract has the capability to switch its active pair in case liquidity is added to another pair.

Additionally, the contract could be designed to tolerate potential reverts from the swap functionality, especially when it is a part of the main transfer flow. This can be achieved by

executing the contract's token swaps in a non-reversible manner, thereby ensuring a more resilient and predictable operation.

# PVC - Price Volatility Concern

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | FANS.sol#L238 |
| **Status** | Unresolved |

## Description

The contract accumulates tokens from the taxes to swap them for ETH. If the variable `swapEnabled` is set to false then the contract will accumulate tokens. Subsequently when the `swapEnabled` is set to `true` then the contract will trigger the swap functionality. If contract holds a huge amount of tokens then the contract will swap a huge amount of tokens for ETH.

It is important to note that the price of the token representing it, can be highly volatile. This means that the value of a price volatility swap involving Ether could fluctuate significantly at the triggered point, potentially leading to significant price volatility for the parties involved.

```
if (swapEnabled && !swapping && sender != pair && fee > 0)
swapForFees();
    ...
    function swapForFees() private inSwap {
        uint256 contractBalance = balanceOf(address(this));
        if (contractBalance >= swapThreshold) {
            uint256 toSwap = contractBalance;
            uint256 initialBalance = address(this).balance;
            swapTokensForETH(toSwap);
    ...
    }
```

## Recommendation

The contract could ensure that it will not sell more than a reasonable amount of tokens in a single transaction. A suggested implementation could check that the maximum amount should be less than a fixed percentage of the exchange reserves. Hence, the contract will guarantee that it cannot accumulate a huge amount of tokens in order to sell them.

## RSW - Redundant Storage Writes

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | FANS.sol#L325 |
| **Status** | Unresolved |

### Description

The contract modifies the state of the following variables without checking if their current value is the same as the one given as an argument. As a result, the contract performs redundant storage writes, when the provided parameter matches the current state of the variables, leading to unnecessary gas consumption and inefficiencies in contract execution.

```
    function setExcludedFromFees(address _address, bool state)
external onlyOwner {
        excludedFromFees[_address] = state;
        emit ExcludedFromFeesUpdated();
    }
```

### Recommendation

The team is advised to implement additional checks within to prevent redundant storage writes when the provided argument matches the current state of the variables. By incorporating statements to compare the new values with the existing values before proceeding with any state modification, the contract can avoid unnecessary storage operations, thereby optimizing gas usage.

## RSFU - Redundant Struct Fee Usage

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | FANS.sol#L199,248 |
| **Status** | Unresolved |

## Description

The contract declares the `Taxes` struct, which includes fields for `marketing` and `development` fees. This struct suggests an intention to organize and manage tax allocations in a structured manner. However, the `Taxes` struct is not utilized in any of the tax allocation calculations or operations. Instead, the contract directly calculates `marketing` and `development` amounts using separate variables and arithmetic operations. This discrepancy indicates that the `Taxes` struct is redundant, as its declared purpose is not reflected in the contract's functional logic. The presence of this unused struct not only adds unnecessary complexity to the contract but also misleads readers about the contract's design and intended mechanisms for handling taxes.

```
struct Taxes {
    uint256 marketing;
    uint256 development;
}
Taxes public buyTaxes = Taxes(0,0);
Taxes public sellTaxes = Taxes(0,0);
...
uint256 deltaBalance = address(this).balance - initialBalance;
uint256 marketingAmt = deltaBalance * 50 / 100;
uint256 developmentAmt = deltaBalance - marketingAmt;
```

## Recommendation

It is recommended to reassess the necessity of the `Taxes` struct within the contract's architecture. If the struct's fields are not intended to be used in future development or for other specific reasons, removing the `Taxes` struct could streamline the contract's codebase, reducing confusion and potential misunderstandings about the contract's functionality. Simplifying the contract by eliminating unused declarations will enhance readability, maintainability, and clarity in the contract's design and implementation strategy.

Should the contract's tax handling logic evolve to require structured data, reconsidering the `Taxes` struct's integration in a manner that aligns with its actual usage would be advisable.

## L04 - Conformance to Solidity Naming Conventions

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | FANS.sol#L43,44,158,281,293,296,299,306,325,329,332 |
| **Status** | Unresolved |

### Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```
mapping (address => uint256) internal _balances
mapping (address => mapping (address => uint256)) internal
_allowances
function WETH() external pure returns (address);
uint256 new_amount
address _pair
address _router
uint256 _development
uint256 _marketing
address _address
address _to
address _token
```

## Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.
Find more information on the Solidity documentation
https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention.

## L08 - Tautology or Contradiction

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | FANS.sol#L303,310 |
| **Status** | Unresolved |

### Description

A tautology is a logical statement that is always true, regardless of the values of its variables. A contradiction is a logical statement that is always false, regardless of the values of its variables.

Using tautologies or contradictions can lead to unintended behavior and can make the code harder to understand and maintain. It is generally considered good practice to avoid tautologies and contradictions in the code.

```
require(totBuyTax >= 0,"Total buy fees cannot be less than 0%")
require(totSellTax >= 0,"Total sell fees cannot be less than
0%")
```

### Recommendation

The team is advised to carefully consider the logical conditions is using in the code and ensure that it is well-defined and make sense in the context of the smart contract.

# L09 - Dead Code Elimination

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | FANS.sol#L266 |
| **Status** | Unresolved |

## Description

In Solidity, dead code is code that is written in the contract, but is never executed or reached during normal contract execution. Dead code can occur for a variety of reasons, such as:

- Conditional statements that are always false.
- Functions that are never called.
- Unreachable code (e.g., code that follows a return statement).

Dead code can make a contract more difficult to understand and maintain, and can also increase the size of the contract and the cost of deploying and interacting with it.

```
function addLiquidity(uint256 tokenAmount, uint256 bnbAmount)
private {
        _approve(address(this), address(router), tokenAmount);
        router.addLiquidityETH{value: bnbAmount}(
            address(this),
            tokenAmount,
            0, // slippage is unavoidable
            0, // slippage is unavoidable
            address(0xdead),
            block.timestamp
        );
    }
```

## Recommendation

To avoid creating dead code, it's important to carefully consider the logic and flow of the contract and to remove any code that is not needed or that is never executed. This can help improve the clarity and efficiency of the contract.

## L16 - Validate Variable Setters

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | FANS.sol#L294 |
| **Status** | Unresolved |

### Description

The contract performs operations on variables that have been configured on user-supplied input. These variables are missing of proper check for the case where a value is zero. This can lead to problems when the contract is executed, as certain actions may not be properly handled when the value is zero.

```
pair = _pair
```

### Recommendation

By adding the proper check, the contract will not allow the variables to be configured with zero value. This will ensure that the contract can handle all possible input values and avoid unexpected behavior or errors. Hence, it can help to prevent the contract from being exploited or operating unexpectedly.

# Functions Analysis

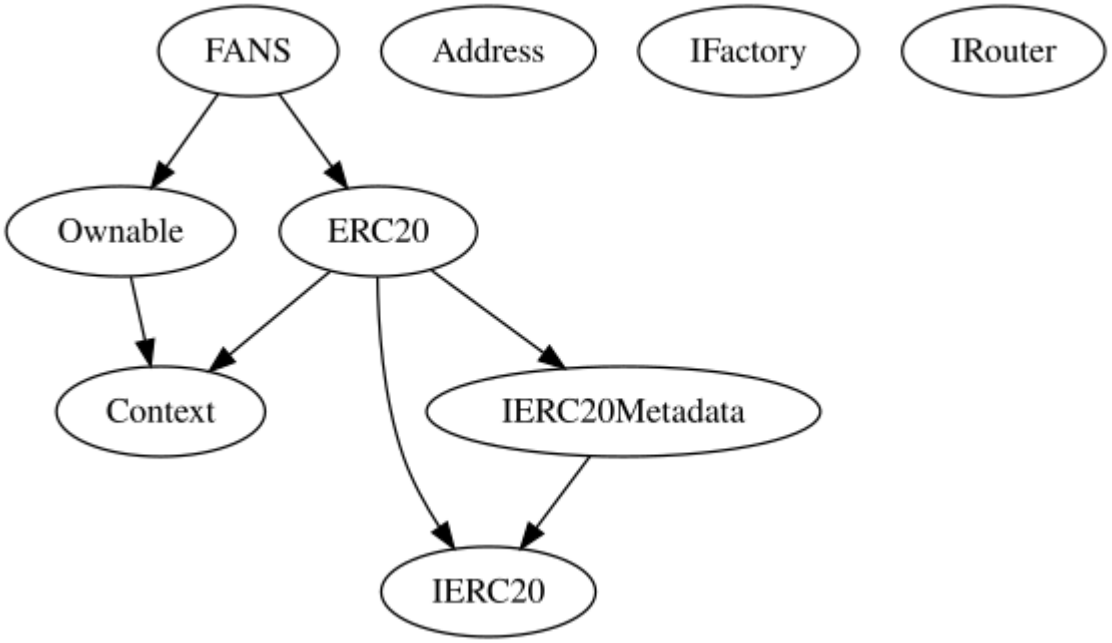
| Contract | Type | Bases | | |
|---|---|---|---|---|
| | **Function Name** | **Visibility** | **Mutability** | **Modifiers** |
| | | | | |
| **Context** | Implementation | | | |
| | _msgSender | Internal | | |
| | _msgData | Internal | | |
| | | | | |
| **IERC20** | Interface | | | |
| | totalSupply | External | | - |
| | balanceOf | External | | - |
| | transfer | External | ✓ | - |
| | allowance | External | | - |
| | approve | External | ✓ | - |
| | transferFrom | External | ✓ | - |
| | | | | |
| **IERC20Metadata** | Interface | IERC20 | | |
| | name | External | | - |
| | symbol | External | | - |
| | decimals | External | | - |
| | | | | |
| **ERC20** | Implementation | Context, IERC20, IERC20Metadata | | |

| | | | | |
|---|---|---|---|---|
| | | Public | ✓ | - |
| | name | Public | | - |
| | symbol | Public | | - |
| | decimals | Public | | - |
| | totalSupply | Public | | - |
| | balanceOf | Public | | - |
| | transfer | Public | ✓ | - |
| | allowance | Public | | - |
| | approve | Public | ✓ | - |
| | transferFrom | Public | ✓ | - |
| | increaseAllowance | Public | ✓ | - |
| | decreaseAllowance | Public | ✓ | - |
| | _transfer | Internal | ✓ | |
| | _mint | Internal | ✓ | |
| | _approve | Internal | ✓ | |
| | _beforeTokenTransfer | Internal | ✓ | |
| | | | | |
| **Address** | Library | | | |
| | sendValue | Internal | ✓ | |
| | | | | |
| **Ownable** | Implementation | Context | | |
| | | Public | ✓ | - |
| | owner | Public | | - |

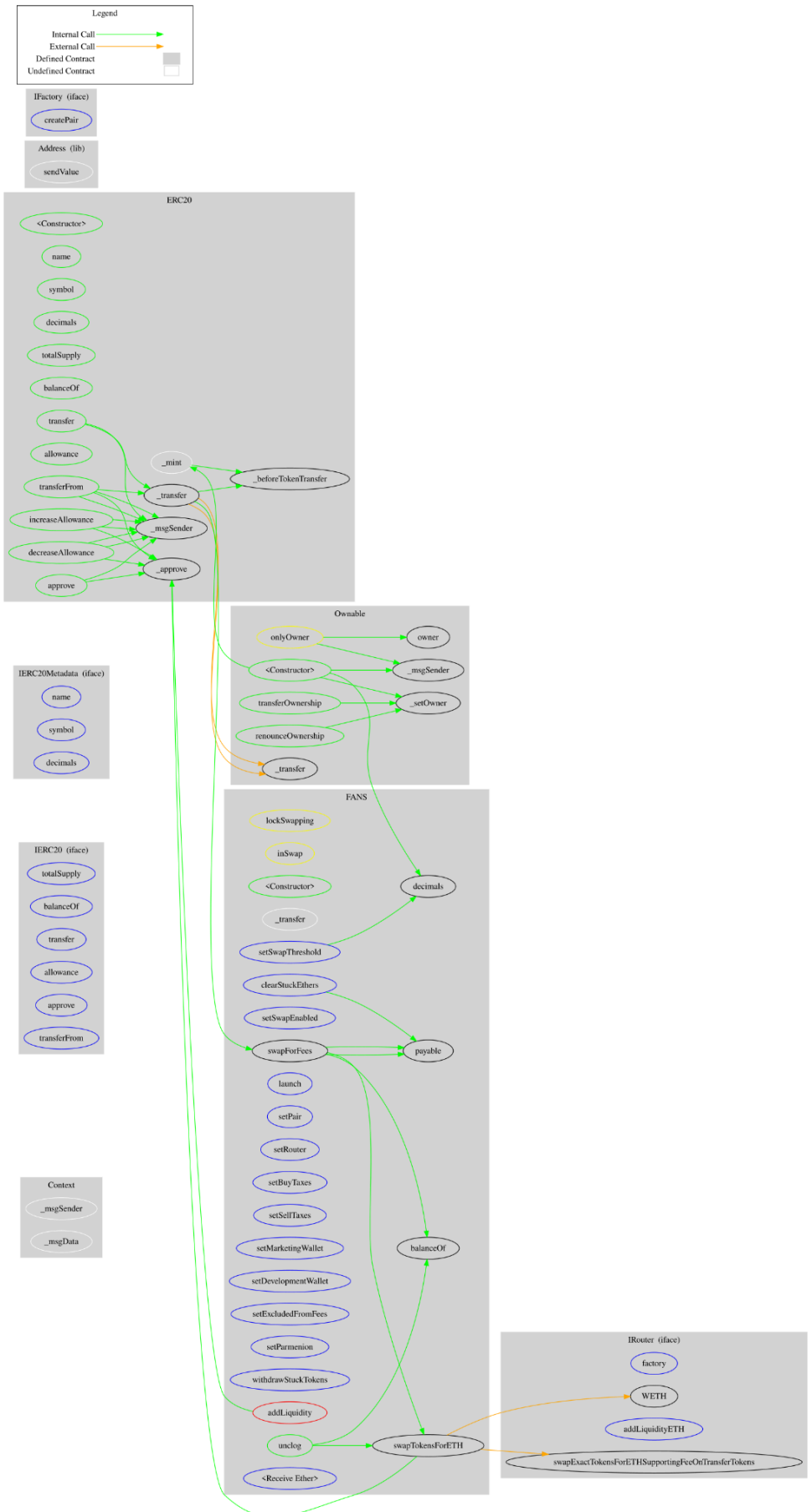
| | | | | |
|---|---|---|---|---|
| | renounceOwnership | Public | ✓ | onlyOwner |
| | transferOwnership | Public | ✓ | onlyOwner |
| | _setOwner | Private | ✓ | |
| | | | | |
| **IFactory** | Interface | | | |
| | createPair | External | ✓ | - |
| | | | | |
| **IRouter** | Interface | | | |
| | factory | External | | - |
| | WETH | External | | - |
| | addLiquidityETH | External | Payable | - |
| | swapExactTokensForETHSupportingFeeOnTransferTokens | External | ✓ | - |
| | | | | |
| **FANS** | Implementation | ERC20, Ownable | | |
| | | Public | ✓ | ERC20 |
| | _transfer | Internal | ✓ | |
| | swapForFees | Private | ✓ | inSwap |
| | swapTokensForETH | Private | ✓ | |
| | addLiquidity | Private | ✓ | |
| | setSwapEnabled | External | ✓ | onlyOwner |
| | setSwapThreshold | External | ✓ | onlyOwner |
| | launch | External | ✓ | onlyOwner |
| | setPair | External | ✓ | onlyOwner |

| | | | | |
|---|---|---|---|---|
| | setRouter | External | ✓ | onlyOwner |
| | setBuyTaxes | External | ✓ | onlyOwner |
| | setSellTaxes | External | ✓ | onlyOwner |
| | setMarketingWallet | External | ✓ | onlyOwner |
| | setDevelopmentWallet | External | ✓ | onlyOwner |
| | setExcludedFromFees | External | ✓ | onlyOwner |
| | setParmenion | External | ✓ | onlyOwner |
| | withdrawStuckTokens | External | ✓ | onlyOwner |
| | clearStuckEthers | External | ✓ | onlyOwner |
| | unclog | Public | ✓ | onlyOwner lockSwapping |
| | | External | Payable | - |

# Inheritance Graph



# Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
IFactory (iface)
createPair
Address (lib)
sendValue
ERC20
<Constructor>
name
symbol
decimals
totalSupply
balanceOf
transfer
allowance
transferFrom
increaseAllowance
decreaseAllowance
approve
_mint
_transfer
_msgSender
_approve
_beforeTokenTransfer
Ownable
onlyOwner
owner
<Constructor>
_msgSender
transferOwnership
_setOwner
renounceOwnership
_transfer
IERC20Metadata (iface)
name
symbol
decimals
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
Context
_msgSender
_msgData
FANS
lockSwapping
inSwap
<Constructor>
_transfer
setSwapThreshold
clearStuckEthers
setSwapEnabled
swapForFees
launch
setPair
setRouter
setBuyTaxes
setSellTaxes
setMarketingWallet
setDevelopmentWallet
setExcludedFromFees
setParmenion
withdrawStuckTokens
addLiquidity
unclog
<Receive Ether>
decimals
payable
balanceOf
swapTokensForETH
IRouter (iface)
factory
WETH
addLiquidityETH
swapExactTokensForETHSupportingFeeOnTransferTokens

# Summary

Social.fans contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. social.fans is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler error or critical issues. The contract Owner can access some admin functions that can not be used in a malicious way to disturb the users' transactions. The fees are locked at 2.5% on buy and sell transactions.

The contract's ownership has been renounced. The information regarding the transaction can be accessed through the following link:

https://etherscan.io/tx/0x328a3c7537c87473aa9ab7062dbe45afbba761f84953d34fd6a44e3d4a193c24

# Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope's prior written consent. This report is not nor should be considered an "endorsement" or "disapproval" of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

# About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

**The Cyberscope team**

https://www.cyberscope.io