Cyberscope

Audit Report

Fin AI

May 2025

Network	BSC
Address	0x5ac50117dde79bd9960505ccf18f46080f51bdc0
Audited by	© cyberscope

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

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

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	FRV	Fee Restoration Vulnerability	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	CO	Code Optimization	Unresolved
●	CCR	Contract Centralization Risk	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	MC	Missing Check	Unresolved
●	PUT	Permanently Unused Tokens	Unresolved
●	RF	Redundant Functionality	Unresolved
●	UDO	Unnecessary Decimals Override	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L19	Stable Compiler Version	Unresolved
●	L22	Potential Locked Ether	Unresolved

Table of Contents

Risk Classification

The criticality of findings in Cyberscope's smart contract audits is determined by evaluating multiple variables. The two primary variables are:

1. **Likelihood of Exploitation**: This considers how easily an attack can be executed, including the economic feasibility for an attacker.
2. **Impact of Exploitation**: This assesses the potential consequences of an attack, particularly in terms of the loss of funds or disruption to the contract's functionality.

Based on these variables, findings are categorized into the following severity levels:

1. **Critical**: Indicates a vulnerability that is both highly likely to be exploited and can result in significant fund loss or severe disruption. Immediate action is required to address these issues.
2. **Medium**: Refers to vulnerabilities that are either less likely to be exploited or would have a moderate impact if exploited. These issues should be addressed in due course to ensure overall contract security.
3. **Minor**: Involves vulnerabilities that are unlikely to be exploited and would have a minor impact. These findings should still be considered for resolution to maintain best practices in security.
4. **Informative**: Points out potential improvements or informational notes that do not pose an immediate risk. Addressing these can enhance the overall quality and robustness of the contract.

Severity	Likelihood / Impact of Exploitation
● Critical	Highly Likely / High Impact
● Medium	Less Likely / High Impact or Highly Likely/ Lower Impact
● Minor / Informative	Unlikely / Low to no Impact

Review

Contract Name	FinAI
Compiler Version	v0.8.28+commit.7893614a
Optimization	200 runs
Explorer	https://bscscan.com/address/0x5ac50117dde79bd9960505ccf18f46080f51bdc0
Address	0x5ac50117dde79bd9960505ccf18f46080f51bdc0
Network	BSC
Symbol	FIN
Decimals	18
Total Supply	1.000.000.000
Badge Eligibility	Must Fix Criticals

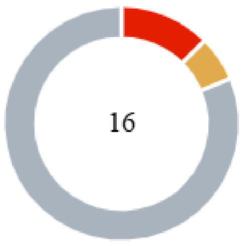
Audit Updates

Initial Audit	13 May 2025

Source Files

Filename	SHA256
FinAI.sol	693eda492cc08109732d56e57352f57ef55654815851ccbe93bef95cb6e651c7

Findings Breakdown



●	Critical	2
●	Medium	1
●	Minor / Informative	13

Severity		Unresolved	Acknowledged	Resolved	Other
●	Critical	2	0	0	0
●	Medium	1	0	0	0
●	Minor / Informative	13	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	FinAI.sol#L766,784,790
Status	Unresolved

Description

The contract owner has the authority to stop the sales for all users excluding the owner. The owner may take advantage of it by setting the total amount of the fees above `100`. As a result, the contract may operate as a honeypot.

```
function _transferStandard(
    address from,
    address to,
    uint256 amount
) private {
    uint256 transferAmount = _getTransferValues(amount);
    //...
}

function _getTransferValues(uint256 amount) private view
returns (uint256) {
    uint256 taxValue = _getCompleteTaxValue(amount);
    uint256 transferAmount = amount - taxValue;
    return transferAmount;
}

function _getCompleteTaxValue(
    uint256 amount
) private view returns (uint256) {
    uint256 allTaxes = _burnFee +
        _marketingFee +
        _developerFee +
        _charityFee;
    uint256 taxValue = (amount * allTaxes) / 100;
    return taxValue;
}
```

Recommendation

Considering the `ELFM` finding, the contract could embody a check for not allowing the total fees to be greater than 25%. The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

ELFM - Exceeds Fees Limit

Criticality	Critical
Location	FinAI.sol#L659,663,667,671
Status	Unresolved

Description

The contract owner has the authority to increase fees over the allowed limit of 25%. The owner may take advantage of it by calling `setMarketingFee` , `setDeveloperFee` , `setCharityFee` or `setBurnFee` functions with a high percentage value.

```
function setBurnFee(uint256 burnFee_) external onlyOwner {
    _burnFee = burnFee_;
}
function setMarketingFee(uint256 marketingFee_) external
onlyOwner {
    _marketingFee = marketingFee_;
}
function setDeveloperFee(uint256 developerFee_) external
onlyOwner {
    _developerFee = developerFee_;
}
function setCharityFee(uint256 charityFee_) external onlyOwner
{
    _charityFee = charityFee_;
}
```

Recommendation

The contract could embody a check for the maximum acceptable value. The team should carefully manage the private keys of the owner’s account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

FRV - Fee Restoration Vulnerability

Criticality	Medium
Location	FinAI.sol#L730,736,740,759
Status	Unresolved

Description

The contract demonstrates a potential vulnerability upon removing and restoring the fees. This vulnerability can occur when the fees have been set to zero. During a transaction, if the fees have been set to zero, then both remove fees and restore fees functions will be executed. The remove fees function is executed to temporarily remove the fees, ensuring the sender is not taxed during the transfer. However, the function prematurely returns without setting the variables that hold the previous fee values.

As a result, when the subsequent restore fees function is called after the transfer, it restores the fees to their previous values. However, since the previous fee values were not properly set to zero, there is a risk that the fees will retain their non-zero values from before the fees were removed. This can lead to unintended consequences, potentially causing incorrect fee calculations or unexpected behavior within the contract.

```
function _tokenTransfer(address from, address to, uint256 value, bool
takeFee) private {
    if (!takeFee) {
        removeAllFee();
    }
    _transferStandard(from, to, value);
    if (!takeFee) {
        restoreAllFee();
    }
}
function removeAllFee() private {
    if (_burnFee == 0 && _marketingFee == 0 && _developerFee == 0 &&
_charityFee == 0) return;
    _previousBurnFee = _burnFee;
    _previousMarketingFee = _marketingFee;
    _previousDeveloperFee = _developerFee;
    _previousCharityFee = _charityFee;
    _burnFee = 0;
    _marketingFee = 0;
    _developerFee = 0;
    _charityFee = 0;
}
function restoreAllFee() private {
    _burnFee = _previousBurnFee;
    _marketingFee = _previousMarketingFee;
    _developerFee = _previousDeveloperFee;
    _charityFee = _previousCharityFee;
}
```

Recommendation

The team is advised to modify the remove fees function to ensure that the previous fee values are correctly set to zero, regardless of their initial values. A recommended approach would be to remove the early return when both fees are zero.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	FinAI.sol#L659,663,667,671,675,679,683,691,695
Status	Unresolved

Description

The contract performs actions and state mutations from external methods that do not result in the emission of events. Emitting events for significant actions is important as it allows external parties, such as wallets or dApps, to track and monitor the activity on the contract. Without these events, it may be difficult for external parties to accurately determine the current state of the contract.

```
function setBurnFee(uint256 burnFee_) external onlyOwner
function setMarketingFee(uint256 marketingFee_) external
onlyOwner
function setDeveloperFee(uint256 developerFee_) external
onlyOwner
function setCharityFee(uint256 charityFee_) external onlyOwner
function setMarketingAddress(address _marketingAddress)
external onlyOwner
function setDeveloperAddress(address _developerAddress)
external onlyOwner
function setCharityAddress(address _charityAddress) external
onlyOwner
function excludeFromFee(address account) public onlyOwner
function includeInFee(address account) public onlyOwner
```

Recommendation

It is recommended to include events in the code that are triggered each time a significant action is taking place within the contract. These events should include relevant details such as the user's address and the nature of the action taken. By doing so, the contract will be more transparent and easily auditable by external parties. It will also help prevent potential issues or disputes that may arise in the future.

CO - Code Optimization

Criticality	Minor / Informative
Location	FinAI.sol#L775,781,782,783
Status	Unresolved

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

During `_transfer` , `transferAmount` is calculated by the `_getTransferValues` and then each tax is calculated separately again during `burnFeeTransfer` and `feeTransfer` .

```
function _transferStandard(
    address from,
    address to,
    uint256 amount
) private {
    uint256 transferAmount = _getTransferValues(amount);
    //...
    burnFeeTransfer(from, amount);
    feeTransfer(from, amount, _marketingFee, marketingAddress);
    feeTransfer(from, amount, _developerFee, developerAddress);
    feeTransfer(from, amount, _charityFee, charityAddress);
    //...
}
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	FinAI.sol#L659,663,667,671,675,679,683,691,695
Status	Unresolved

Description

The contract's functionality and behavior are heavily dependent on external parameters or configurations. While external configuration can offer flexibility, it also poses several centralization risks that warrant attention. Centralization risks arising from the dependence on external configuration include Single Point of Control, Vulnerability to Attacks, Operational Delays, Trust Dependencies, and Decentralization Erosion.

```
function setBurnFee(uint256 burnFee_) external onlyOwner
function setMarketingFee(uint256 marketingFee_) external
onlyOwner
function setDeveloperFee(uint256 developerFee_) external
onlyOwner
function setCharityFee(uint256 charityFee_) external onlyOwner
function setMarketingAddress(address _marketingAddress)
external onlyOwner
function setDeveloperAddress(address _developerAddress)
external onlyOwner
function setCharityAddress(address _charityAddress) external
onlyOwner
function excludeFromFee(address account) public onlyOwner
function includeInFee(address account) public onlyOwner
```

Recommendation

To address this finding and mitigate centralization risks, it is recommended to evaluate the feasibility of migrating critical configurations and functionality into the contract's codebase itself. This approach would reduce external dependencies and enhance the contract's self-sufficiency. It is essential to carefully weigh the trade-offs between external configuration flexibility and the risks associated with centralization.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	FinAI.sol#L618
Status	Unresolved

Description

The contract declares state variables that their value is initialized once in the constructor and are not modified afterwards. The `immutable` is a special declaration for this kind of state variables that saves gas when it is defined.

```
_decimals
```

Recommendation

By declaring a variable as immutable, the Solidity compiler is able to make certain optimizations. This can reduce the amount of storage and computation required by the contract, and make it more gas-efficient.

MC - Missing Check

Criticality	Minor / Informative
Location	FinAI.sol#L659,663,667,671,675,679,683,691,695
Status	Unresolved

Description

The contract is processing variables that have not been properly sanitized and checked that they form the proper shape. These variables may produce vulnerability issues.

Functions that set fees are missing checks to ensure that the total fees cannot be set to values greater than 25%. Moreover there is no check to ensure that they cannot be greater than 100%

```
function setBurnFee(uint256 burnFee_) external onlyOwner {
    _burnFee = burnFee_;
}
function setMarketingFee(uint256 marketingFee_) external
onlyOwner {
    _marketingFee = marketingFee_;
}
function setDeveloperFee(uint256 developerFee_) external
onlyOwner {
    _developerFee = developerFee_;
}
function setCharityFee(uint256 charityFee_) external onlyOwner
{
    _charityFee = charityFee_;
}
```

Functions that set addresses are missing checks to ensure that the provided address is not address zero.

```
function setMarketingAddress(address _marketingAddress)
external onlyOwner {
    marketingAddress = _marketingAddress;
}
function setDeveloperAddress(address _developerAddress)
external onlyOwner {
    developerAddress = _developerAddress;
}
function setCharityAddress(address _charityAddress) external
onlyOwner {
    charityAddress = _charityAddress;
}
```

`excludeFromFee` and `includeInFee` are missing checks to ensure the account added as parameter is not already excluded or included respectively.

```
function excludeFromFee(address account) public onlyOwner {
    _isExcludedFromFee[account] = true;
}
function includeInFee(address account) public onlyOwner {
    _isExcludedFromFee[account] = false;
}
```

Recommendation

The team is advised to properly check the variables according to the required specifications.

PUT - Permanently Unused Tokens

Criticality	Minor / Informative
Location	FinAI.sol#L675,679,683,766
Status	Unresolved

Description

The `_transferStandard` function sends fees to `marketingAddress`, `developerAddress` and `charityAddress` by the `feeTransfer` function. These addresses can be equal to `address(0)`. Tokens send to that address cannot be used again. Depending on the implementation tokens could only be sent to `address(0)` if they are burned.

However the contract is using a different implementation for the burned tokens that instead subtracts them from the `totalSupply` via the `burnFeeTransfer` so sending tokens to address(0) should not be allowed.

```
function setMarketingAddress(address _marketingAddress)
external onlyOwner {
    marketingAddress = _marketingAddress;
}
function setDeveloperAddress(address _developerAddress)
external onlyOwner {
    developerAddress = _developerAddress;
}
function setCharityAddress(address _charityAddress) external
onlyOwner {
    charityAddress = _charityAddress;
}
function _transferStandard(address from, address to, uint256
amount) private {
    //...
    burnFeeTransfer(from, amount);
    feeTransfer(from, amount, _marketingFee, marketingAddress);
    feeTransfer(from, amount, _developerFee, developerAddress);
    feeTransfer(from, amount, _charityFee, charityAddress);
    //...
}
```

Recommendation

The team should implement checks in the configuration function and constructor to ensure that address(0) will not be able to receive tokens.

Additionally, the team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

RF - Redundant Functionality

Criticality	Minor / Informative
Location	FinAI.sol#L651,712
Status	Unresolved

Description

The contract has a function called getBalance that returns the contract's balance of the native currency. This function is redundant since balance is public method that can be called for any address.

```
function getBalance() private view returns (uint256) {
    return address(this).balance;
}
```

Additionally, in the `_transfer` function `_beforeTokenTransfer` is called but it has no functionality.

```
function _transfer(
    address sender,
    address recipient,
    uint256 amount
) internal virtual override {
    //...
    _beforeTokenTransfer(sender, recipient, amount);
    //...
}
```

Recommendation

It is recommended to remove redundant functionalities to enhance code optimization and readability.

UDO - Unnecessary Decimals Override

Criticality	Minor / Informative
Location	FinAI.sol#L655
Status	Unresolved

Description

The contract is currently implementing an override of the decimals function, which simply returns the value 18. This override is redundant since the extending token contract already specifies 18 decimals as its standard. In the context of ERC-20 tokens, 18 decimals is a common default, and overriding this function to return the same value adds unnecessary complexity to the contract. This redundancy does not contribute to the functionality of the contract and could potentially lead to confusion about the necessity of this override.

```
function decimals() public view virtual override returns
(uint8) {
    return _decimals;
}
```

Recommendation

Since the inherited ERC-20 contract already defines the decimals number, maintaining an overriding function that merely repeats this value does not contribute to the contract's effectiveness. As a result, it is recommended to remove the redundant `decimals` function from the contract. Removing this function will simplify the contract, making it more straightforward to maintain without impacting its operational capabilities.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	FinAI.sol#L598,601,604,607,675,679,683
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.

```
uint256 public _burnFee
uint256 public _marketingFee
uint256 public _developerFee
uint256 public _charityFee
address _marketingAddress
address _developerAddress
address _charityAddress
```

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/stable/style-guide.html#naming-conventions.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	FinAI.sol#L449,506
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 _transfer(address sender, address recipient, uint256
amount) internal virtual {
        require(sender != address(0), "ERC20: transfer from the
zero address");
...
        }
        _balances[recipient] += amount;
        emit Transfer(sender, recipient, amount);
        _afterTokenTransfer(sender, recipient, amount);
    }
...
```

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	FinAI.sol#L676,680,684
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.

```
marketingAddress = _marketingAddress
developerAddress = _developerAddress
charityAddress = _charityAddress
```

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.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	FinAI.sol#L3
Status	Unresolved

Description

The `^` symbol indicates that any version of Solidity that is compatible with the specified version (i.e., any version that is a higher minor or patch version) can be used to compile the contract. The version lock is a mechanism that allows the author to specify a minimum version of the Solidity compiler that must be used to compile the contract code. This is useful because it ensures that the contract will be compiled using a version of the compiler that is known to be compatible with the code.

```
pragma solidity ^0.8.0;
```

Recommendation

The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest version that provides all the required functionality for the codebase. As a result, the project will be compiled in a well-tested LTS (Long Term Support) environment.

L22 - Potential Locked Ether

Criticality	Minor / Informative
Location	FinAI.sol#L649
Status	Unresolved

Description

The contract is able to receive Ether via the `receive` function. This Ether cannot be transferred. Thus, it is impossible to access the locked Ether. This may produce a financial loss for the users that have called the `receive` method.

```
receive() external payable {}
```

Recommendation

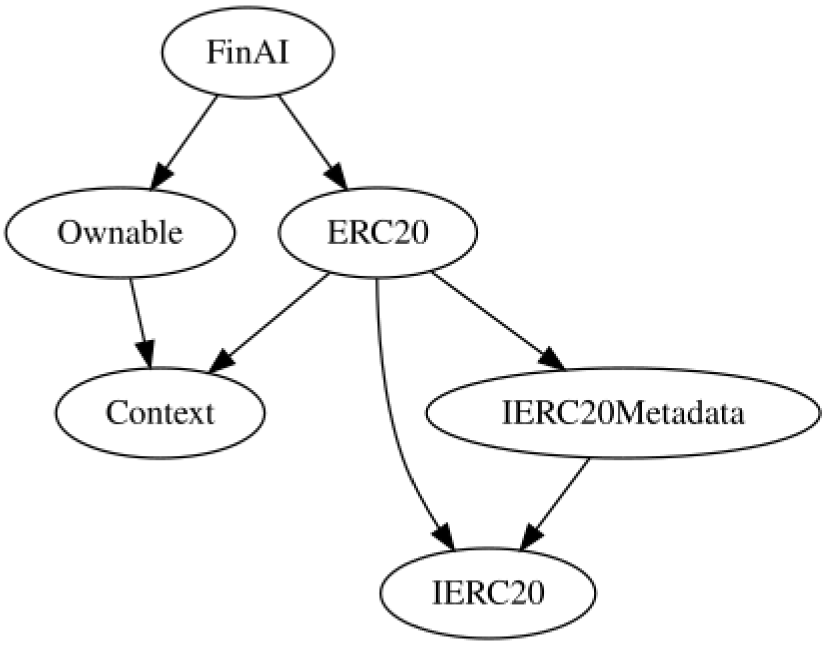
The team is advised to either remove the payable method or add a withdraw functionality. it is important to carefully consider the risks and potential issues associated with locked Ether.

Functions Analysis

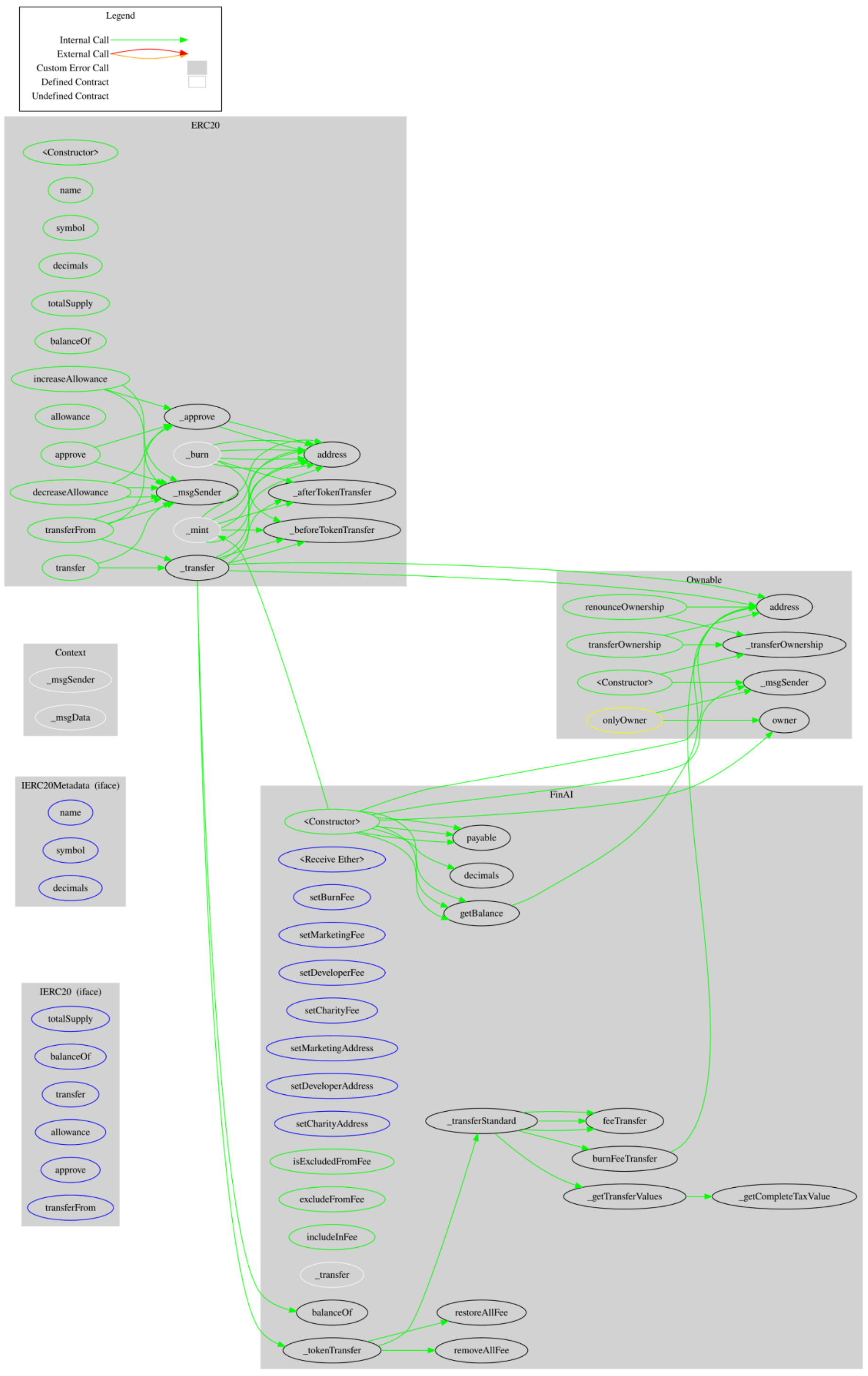
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
FinAI	Implementation	ERC20, Ownable		
		Public	Payable	ERC20
		External	Payable	-
	getBalance	Private		
	decimals	Public		-
	setBurnFee	External	✓	onlyOwner
	setMarketingFee	External	✓	onlyOwner
	setDeveloperFee	External	✓	onlyOwner
	setCharityFee	External	✓	onlyOwner
	setMarketingAddress	External	✓	onlyOwner
	setDeveloperAddress	External	✓	onlyOwner
	setCharityAddress	External	✓	onlyOwner
	isExcludedFromFee	Public		-
	excludeFromFee	Public	✓	onlyOwner
	includeInFee	Public	✓	onlyOwner
	_transfer	Internal	✓	
	_tokenTransfer	Private	✓	
	removeAllFee	Private	✓	
	restoreAllFee	Private	✓	
	_transferStandard	Private	✓	

	_getTransferValues	Private		
	_getCompleteTaxValue	Private		
	burnFeeTransfer	Private	✓	
	feeTransfer	Private	✓	

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Custom Error Call
Defined Contract
Undefined Contract
ERC20
<Constructor>
name
symbol
decimals
totalSupply
balanceOf
increaseAllowance
allowance
approve
decreaseAllowance
transferFrom
transfer
_approve
_burn
_msgSender
_mint
_transfer
address
_afterTokenTransfer
_beforeTokenTransfer
Context
_msgSender
_msgData
IERC20Metadata (iface)
name
symbol
decimals
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
Ownable
renounceOwnership
transferOwnership
<Constructor>
onlyOwner
address
_transferOwnership
_msgSender
owner
FinAI
<Constructor>
<Receive Ether>
setBurnFee
setMarketingFee
setDeveloperFee
setCharityFee
setMarketingAddress
setDeveloperAddress
setCharityAddress
isExcludedFromFee
excludeFromFee
includeInFee
_transfer
balanceOf
_tokenTransfer
payable
decimals
getBalance
_transferStandard
feeTransfer
burnFeeTransfer
_getTransferValues
_getCompleteTaxValue
restoreAllFee
removeAllFee

Summary

Fin AI contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. There are some functions that can be abused by the owner like stop transactions and manipulate the fees. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats.

Disclaimer

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

cyberscope.io