Cyberscope

Audit Report

blasthoge

April 2024

Network	Blast Testnet
Address	0x92ec6bC331c2ff9cF3F37f863f6600E9CeaFF5B5
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	Unresolved
●	MT	Mints Tokens	Unresolved
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	UPV	Uninitialized Protocol Variable	Unresolved
●	CCS	Clean Code Standard	Unresolved
●	CO	Code Optimization	Unresolved
●	CCR	Contract Centralization Risk	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	IRC	Initial Reflection Calculation	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	RCS	Redundant Code Segments	Unresolved
●	URR	Unnecessary Reflection Recalculation	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L18	Multiple Pragma Directives	Unresolved

●	L19	Stable Compiler Version	Unresolved

Table of Contents

Review

Contract Name	TokenFiERC20
Compiler Version	v0.8.18+commit.87f61d96
Optimization	200 runs
Explorer	https://testnet.blastscan.io/address/0x92ec6bC331c2ff9cF3F37f863f6600E9CeaFF5B5
Address	0x92ec6bC331c2ff9cF3F37f863f6600E9CeaFF5B5
Network	Blast Testnet
Decimals	18
Badge Eligibility	Must Fix Criticals

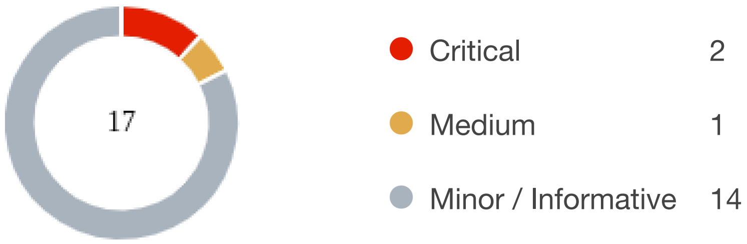
Audit Updates

Initial Audit	21 Apr 2024

Source Files

Filename	SHA256
interfaces/ITokenLauncherLiquidityPoolFactory.sol	a0bbc66c554cc4d67490d255c623121a352d7f7316b1370198764efb6c93ac67
interfaces/ITokenLauncherERC20.sol	74047892dd77691bafb4f80a08eae96f52c43921e3eac4535fc046861e1921c7
interfaces/ITokenLauncherCommon.sol	24c4999a0341aa3c7de1227d8d0510a39a08ca00861f51d1769e81931feeb532
interfaces/IBuyBackHandler.sol	064f5188d0eff363ed02a052a75d7a857c9478ab96d2d30bd09b8f987306b6cc
interfaces/IBlastPoints.sol	8b4f699b4589335a1b1f8fa579085124b6658fd0e45996e267a2f2e8c8807105
interfaces/IBlast.sol	0799032ecb04859c62e4b0ce69ee1a120fb1369831b44462e6711fb9410c46d9
interfaces/uniswap/IUniswapV2Router02.sol	e2bb7a517fbcf88346af61dc15f08d34097ba663194f3867f38e75b27d4fbae5
interfaces/uniswap/IUniswapV2Router01.sol	48b4e89370e99d00a8227841f450ae5d8004aa2090b12517af197cd1806c8f64
contracts/TokenFiERC20.sol	bde5206ae9a31381e5c152a0435cdfff6ee42c4830b4dbbc9488b03d027089d1

Findings Breakdown



	Severity	Unresolved	Acknowledged	Resolved	Other
●	Critical	2	0	0	0
●	Medium	1	0	0	0
●	Minor / Informative	14	0	0	0

ELFM - Exceeds Fees Limit

Criticality	Critical
Location	ontracts/TokenFiERC20.sol#L147
Status	Unresolved

Description

The contract owner has the authority to increase over the allowed limit of 25%. The owner may take advantage of it by calling the `updateFees` function with a high percentage value.

```
function updateFees(ITokenLauncherERC20.Fees memory _fees) external
onlyRole(DEFAULT_ADMIN_ROLE) {
    if (isReflectionToken) {
        require(_fees.reflection.percentage > 0, "TokenFiERC20:
reflection percentage must be non-zero");
    } else {
        require(_fees.reflection.percentage == 0, "TokenFiERC20:
reflection percentage must be zero");
    }
    uint256 maxFee = _fees.transferFee.percentage +
_fees.burn.percentage + _fees.reflection.percentage +
_fees.buyback.percentage;
    require(maxFee <= MULTIPLIER_BASIS, "TokenFiERC20: fees sum must be
less than 100%");
    fees = _fees;
}
```

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.

MT - Mints Tokens

Criticality	Critical
Location	contracts/TokenFiERC20.sol#L479
Status	Unresolved

Description

The contract owner has the authority to mint tokens. The owner may take advantage of it by calling the `mint` function. As a result, the contract tokens will be highly inflated.

```
function mint(address to, uint256 amount) external
onlyRole(DEFAULT_ADMIN_ROLE) {
    require(totalSupply() + amount <= maxSupply, "TokenFiERC20: max
supply exceeded");
    if (isReflectionToken) {
        _mintReflection(to, amount);
    } else {
        super._mint(to, amount);
    }
}
```

Recommendation

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.

UPV - Uninitialized Protocol Variable

Criticality	Medium
Location	contracts/TokenFiERC20.sol#L403,405
Status	Unresolved

Description

It was observed that the contract contains a potential vulnerability related to the uninitialized `buybackHandler` variable.

In the `_transfer` function, the contract attempts to execute a buyback operation if certain conditions are met. However, before calling the `buyback` function from the `IBuyBackHandler` interface, the contract uses the `buybackHandler` variable. This variable is assumed to hold the address of the buyback handler contract. However, the variable is not initialized.

If the `buybackHandler` variable is not initialized prior to its usage, it will contain the default value, which is `address(0)`. Attempting to call a function on this uninitialized address will result in a revert error, causing the transaction to fail.

```
function _transfer(address sender, address recipient, uint256 amount)
internal virtual override {
    ...
    if (fees.buyback.percentage > 0 && (!fees.buyback.onlyOnSwaps ||
isSwap) && !exemptedFromTax) {
        uint256 buybackFee = (originalAmount * fees.buyback.percentage)
/ MULTIPLIER_BASIS;
        _transferInternal(sender, buybackHandler, buybackFee,
originalAmount, false);
        if (!_exchangePools.contains(sender) && buybackDetails.router !=
address(0)) {
            IBuyBackHandler(buybackHandler).buyback(treasury,
buybackDetails);
        }
        amount -= buybackFee;
        emit TransferTax(sender, buybackHandler, buybackFee,
"buybackFee");
    }
    _transferInternal(sender, recipient, amount, originalAmount, true);
}
```

Recommendation

It is crucial to promptly address the uninitialized `buybackHandler` variable to mitigate potential risks to the smart contract's functionality and user funds. We recommend initializing the `buybackHandler` variable with the correct address of the buyback handler contract before its usage in the `_transfer` function. It's imperative to verify the proper initialization of the variable to ensure seamless execution of the buyback functionality, thereby preventing any potential revert errors.

Failure to initialize the `buybackHandler` variable may lead to unexpected contract behavior, including transaction reverting and potential loss of user funds. It is crucial to address this issue promptly to ensure the security and reliability of the smart contract.

CCS - Clean Code Standard

Criticality	Minor / Informative
Location	contracts/TokenFiERC20.sol#L496,501
Status	Unresolved

Description

In the contract, there are modifiers that are interspersed among the function definitions, which can make it cumbersome for developers to quickly identify and comprehend the access control mechanisms enforced by these modifiers. Conventionally, modifiers are declared at the beginning of the contract to provide a clear and concise overview of the access restrictions applied throughout the contract.

```
modifier onlyFromAdminOrLauncher() {
    require(hasRole(DEFAULT_ADMIN_ROLE, msg.sender) || msg.sender ==
tokenLauncher, "TokenFiERC20: must be admin");
    _;
}

modifier onlyReflection() {
    require(isReflectionToken, "TokenFiERC20: reflection not enabled");
    _;
}
```

Recommendation

Adhering to clean code standards not only enhances the readability of the contract but also facilitates easier maintenance and auditability. By consolidating modifier declarations at the top of the contract, developers can swiftly locate and review the access control logic, contributing to the overall robustness and security of the smart contract implementation. Therefore, it is recommended to relocate the modifier declarations to the top of the contract for improved clarity and adherence to best practices in smart contract development.

CO - Code Optimization

Criticality	Minor / Informative
Location	contracts/TokenFiERC20.sol#L183,195,248,267,286,305
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.

```
return _isExcludedFromReflectionRewards[account] || account ==
address(this);
...
function reflectionFromToken(uint256 tAmount, bool deductTransferFee)
public view returns (uint256) {
    require(tAmount <= totalReflection.t, "Amount must be less than
supply");
    if (!deductTransferFee) {
        (uint256 rAmount, , , , ) = _getValues(tAmount, tAmount, false,
true);
        return rAmount;
    } else {
        (, uint256 rTransferAmount, , , ) = _getValues(tAmount, tAmount,
false, true);
        return rTransferAmount;
    }
}
...
function _transferStandard(
    address sender,
    address recipient,
    // solhint-disable-next-line
    uint256 tAmount,
    // solhint-disable-next-line
    uint256 tTransferAmount,
    uint256 rAmount,
    uint256 rTransferAmount,
    bool shouldReflectFee
) private {
    _rOwned[sender] = _rOwned[sender] - rAmount;
    if (shouldReflectFee) {
        _rOwned[recipient] = _rOwned[recipient] + rTransferAmount;
    } else {
        _rOwned[recipient] = _rOwned[recipient] + rAmount;
    }
}

function _transferToExcluded(
    address sender,
    address recipient,
    uint256 tAmount,
    uint256 tTransferAmount,
    uint256 rAmount,
    uint256 rTransferAmount,
    bool shouldReflectFee
) private {
    _rOwned[sender] = _rOwned[sender] - rAmount;
    if (shouldReflectFee) {
        _tOwned[recipient] = _tOwned[recipient] + tTransferAmount;
        _rOwned[recipient] = _rOwned[recipient] + rTransferAmount;
    } else {
        _tOwned[recipient] = _tOwned[recipient] + tAmount;
        _rOwned[recipient] = _rOwned[recipient] + rAmount;
    }
}

function _transferFromExcluded(
    address sender,
    address recipient,
    uint256 tAmount,
    // solhint-disable-next-line
    uint256 tTransferAmount,
    uint256 rAmount,
    uint256 rTransferAmount,
    bool shouldReflectFee
) private {
    _tOwned[sender] = _tOwned[sender] - tAmount;
    _rOwned[sender] = _rOwned[sender] - rAmount;
    if (shouldReflectFee) {
        _rOwned[recipient] = _rOwned[recipient] + rTransferAmount;
    } else {
        _rOwned[recipient] = _rOwned[recipient] + rAmount;
    }
}

function _transferBothExcluded(
    address sender,
    address recipient,
    uint256 tAmount,
    uint256 tTransferAmount,
    uint256 rAmount,
    uint256 rTransferAmount,
    bool shouldReflectFee
) private {
    _tOwned[sender] = _tOwned[sender] - tAmount;
    _rOwned[sender] = _rOwned[sender] - rAmount;
    if (shouldReflectFee) {
        _tOwned[recipient] = _tOwned[recipient] + tTransferAmount;
        _rOwned[recipient] = _rOwned[recipient] + rTransferAmount;
    } else {
        _tOwned[recipient] = _tOwned[recipient] + tAmount;
        _rOwned[recipient] = _rOwned[recipient] + rAmount;
    }
}
```

`address(this)` could be added to `_isExcludedFromReflectionRewards` at the contract's constructor.

`_getValues` function could be called outside the if statement.

The functions `_transferStandard`, `_transferToExcluded`, `_transferFromExcluded` and `_transferBothExcluded` could be replaced by a single function that could handle every case.

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	contracts/TokenFiERC20.sol#L119,136,142,147,375,380,464,479,488
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
setBuybackDetails(ITokenLauncherLiquidityPoolFactory.BuyBackDetails
memory _buybackDetails) external onlyRole(FEE_MANAGER_ROLE)
function addExchangePool(address pool) external
onlyRole(FEE_MANAGER_ROLE)
function addExemptAddress(address account) external
onlyRole(FEE_MANAGER_ROLE)
function updateFees(ITokenLauncherERC20.Fees memory _fees) external
onlyRole(DEFAULT_ADMIN_ROLE)
function removeExchangePool(address pool) external
onlyRole(FEE_MANAGER_ROLE)
function removeExemptAddress(address account) external
onlyRole(FEE_MANAGER_ROLE)
function _mintReflection(address account, uint256 amount) private
function mint(address to, uint256 amount) external
onlyRole(DEFAULT_ADMIN_ROLE)
function updateTokenLauncher(address _newTokenLauncher) external
onlyRole(DEFAULT_ADMIN_ROLE)
```

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	contracts/TokenFiERC20.sol#L73,74,76,97
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.

```
treasury
maxSupply
_decimals
isReflectionToken
```

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.

IRC - Initial Reflection Calculation

Criticality	Minor / Informative
Location	contracts/TokenFiERC20.sol#L46,90
Status	Unresolved

Description

The contract implements a reflection token mechanism. However, it was observed that in the initialization of the `MAX` variable, which is utilized to calculate the initial reflection, instead of the typical `type(uint256).max`, half of that is used.

```
uint256 public constant MAX = type(uint256).max / 2;
```

The `MAX` variable is initialized to half the maximum value of `uint256`, which is defined as `type(uint256).max / 2`. Consequently, the subsequent calculation of the initial reflection, `totalReflection.r`, is based on this reduced `MAX` value.

This implementation results in the initial reflection being set to only half of what is typically expected for standard reflection tokens. The discrepancy in the initial reflection value could potentially lead to unexpected behavior, impacting token economics, liquidity provisioning, and investor expectations.

```
totalReflection.r = (MAX - (MAX % initialSupply));
```

Recommendation

To correct this issue and ensure the proper functioning of the reflection token mechanism, the team is advised to initialize the `MAX` variable with the full maximum value of `uint256`.

Following these recommendations will help mitigate potential discrepancies and ensure that the reflection token operates as intended, maintaining the integrity of the contract and safeguarding the interests of token holders and ecosystem participants.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	contracts/TokenFiERC20.sol#L212,221
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 excludeAccount(address account) external onlyReflection
onlyFromAdminOrLauncher
function includeAccount(address account) external onlyReflection
onlyFromAdminOrLauncher
```

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.

RCS - Redundant Code Segments

Criticality	Minor / Informative
Location	contracts/TokenFiERC20.sol#L70,75,85,100,106,111,251,253,290
Status	Unresolved

Description

The contract is currently containing code segments, that do not provide any actual functionality. Such redundant code segments can lead to confusion and misinterpretation of the contract's purpose and functionality. Moreover, they contribute to unnecessary bloat in the contract, potentially impacting its efficiency and clarity.

```
treasury = 0x46CB73A78b3d164Bb35E39816546BE863793e732;// address(this);
...
// uint256 maxFee = _input.fees.transferFee.percentage +
_input.fees.burn.percentage + _input.fees.reflection.percentage +
_input.fees.buyback.percentage;
// require(maxFee <= MULTIPLIER_BASIS, "TokenFiERC20: fees sum must be
less than 100%");
...
// buybackHandler = _input.buybackHandler;
...
// Exempt the buyback handler, treasury and burn address
// _exemptedFromTax.add(_input.buybackHandler);
...
// _grantRole(DEFAULT_ADMIN_ROLE, _input.tokenStore);
...
//
IBlastPoints(0x2fc95838c71e76ec69ff817983BFf17c710F34E0).configurePointsO
perator(0x7eB11F470551e7Add74C93e32F2AE3AaCf4Fa66F); // blast-testnet
...
// solhint-disable-next-line
```

Recommendation

It is recommended to remove these redundant code segments from the contract. Eliminating these non-functional parts will streamline the contract, making it more efficient and easier to comprehend. This action will also reduce the potential for confusion among users and developers who interact with or audit the contract.

URR - Unnecessary Reflection Recalculation

Criticality	Minor / Informative
Location	contracts/TokenFiERC20.sol#L221
Status	Unresolved

Description

The `includeAccount` function in the contract is designed to include an account in reflection rewards. However, it has been identified that this function executes unnecessary recalculations of reflection values for the account being included.

When the `includeAccount` function is called to include a specific account, it iterates over the array of excluded accounts `_excluded` to find the target account. Once the account is found, the contract recalculates the reflection values for the account, despite not being required for inclusion.

Specifically, the unnecessary recalculations occur in the following lines: `solidity totalReflection.r = totalReflection.r - _rOwned[account]; _rOwned[account] = _tOwned[account] * currentRate; ... totalReflection.r = totalReflection.r + _rOwned[account];` These lines first subtract the reflection balance of the account from the total reflection, then update the reflection balance of the account, and finally add the updated reflection balance back to the total reflection. However, these recalculations are redundant for the purpose of including the account in reflection rewards.

Recommendation

To optimize gas usage and improve efficiency, it is advised to remove the unnecessary recalculations of reflection values from the `includeAccount` function. Since the function's primary objective is to include the account in reflection rewards, there is no need to adjust the reflection values during this process. By removing these redundant calculations, gas costs can be reduced, leading to a more efficient and cost-effective smart contract.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	contracts/TokenFiERC20.sol#L31
Status	Unresolved

Description

State variables can be declared as constant using the constant keyword. This means that the value of the state variable cannot be changed after it has been set. Additionally, the constant variables decrease gas consumption of the corresponding transaction.

```
address public buybackHandler
```

Recommendation

Constant state variables can be useful when the contract wants to ensure that the value of a state variable cannot be changed by any function in the contract. This can be useful for storing values that are important to the contract's behavior, such as the contract's address or the maximum number of times a certain function can be called. The team is advised to add the constant keyword to state variables that never change.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	contracts/TokenFiERC20.sol#L22,124,161,524
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.

```
IBlast public BLAST
ITokenLauncherLiquidityPoolFactory.BuyBackDetails memory _buybackDetails
ITokenLauncherERC20.Fees memory _fees
address _newTokenLauncher
```

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.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	contracts/TokenFiERC20.sol#L155
Status	Unresolved

Description

Events are a way to record and log information about changes or actions that occur within a contract. They are often used to notify external parties or clients about events that have occurred within the contract, such as the transfer of tokens or the completion of a task.

It's important to carefully design and implement the events in a contract, and to ensure that all required events are included. It's also a good idea to test the contract to ensure that all events are being properly triggered and logged.

```
function updateFees(ITokenLauncherERC20.Fees memory _fees) external
onlyRole(DEFAULT_ADMIN_ROLE) {
    if (isReflectionToken) {
        require(_fees.reflection.percentage > 0, "TokenFiERC20:
reflection percentage must be non-zero");
    } else {
        require(_fees.reflection.percentage == 0, "TokenFiERC20:
reflection percentage must be zero");
    }
    uint256 maxFee = _fees.transferFee.percentage +
_fees.burn.percentage + _fees.reflection.percentage +
_fees.buyback.percentage;
    require(maxFee <= MULTIPLIER_BASIS, "TokenFiERC20: fees sum must be
less than 100%");
    fees = _fees;
}
```

Recommendation

By including all required events in the contract and thoroughly testing the contract's functionality, the contract ensures that it performs as intended and does not have any missing events that could cause issues with its arithmetic.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	contracts/TokenFiERC20.sol#L526
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.

```
tokenLauncher = _newTokenLauncher
```

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.

L18 - Multiple Pragma Directives

Criticality	Minor / Informative
Location	contracts/TokenFiERC20.sol#L3 interfaces/uniswap/contracts/IUniswapV2Router01.sol#L1 interfaces/uniswap/contracts/IUniswapV2Router02.sol#L1
Status	Unresolved

Description

If the contract includes multiple conflicting pragma directives, it may produce unexpected errors. To avoid this, it's important to include the correct pragma directive at the top of the contract and to ensure that it is the only pragma directive included in the contract.

```
pragma solidity >=0.8.0;
pragma solidity >=0.6.2;
```

Recommendation

It is important to include only one pragma directive at the top of the contract and to ensure that it accurately reflects the version of Solidity that the contract is written in.

By including all required compiler options and flags in a single pragma directive, the potential conflicts could be avoided and ensure that the contract can be compiled correctly.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	contracts/TokenFiERC20.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.

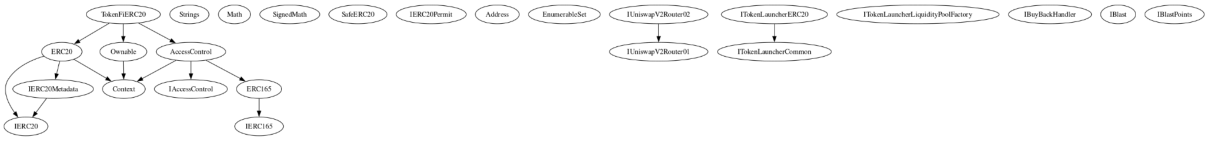
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
TokenFiERC20	Implementation	ERC20, AccessControl, Ownable		
		Public	✓	ERC20
	decimals	Public		-
	setBuybackDetails	External	✓	onlyRole
	addExchangePool	External	✓	onlyRole
	addExemptAddress	External	✓	onlyRole
	updateFees	External	✓	onlyRole
	isExemptedFromTax	External		-
	isExchangePool	External		-
	balanceOf	Public		-
	totalSupply	Public		-
	isExcludedFromReflectionRewards	Public		-
	reflect	External	✓	onlyReflection
	reflectionFromToken	Public		-
	tokenFromReflection	Public		-
	excludeAccount	External	✓	onlyReflection onlyFromAdminOrLauncher

	includeAccount	External	✓	onlyReflection onlyFromAdminOrLauncher
	totalFees	Public		-
	_balanceOfReflection	Private		
	_transferStandard	Private	✓	
	_transferToExcluded	Private	✓	
	_transferFromExcluded	Private	✓	
	_transferBothExcluded	Private	✓	
	_reflectFee	Private	✓	
	_getValues	Private		
	_getTValues	Private		
	_getRValues	Private		
	_getRate	Private		
	_getCurrentSupply	Private		
	removeExchangePool	External	✓	onlyRole
	removeExemptAddress	External	✓	onlyRole
	_transfer	Internal	✓	
	_transferReflection	Private	✓	
	_transferInternal	Private	✓	
	_isSwap	Internal		
	_isExemptedFromTax	Internal		
	_mintReflection	Private	✓	
	mint	External	✓	onlyRole
	updateTokenLauncher	External	✓	onlyRole

	claimYield	External	✓	onlyOwner
	claimAllYield	External	✓	onlyOwner
	claimMyContractsGas	External	✓	onlyOwner
	claimMyContractsGasMax	External	✓	onlyOwner

Inheritance Graph



Flow Graph

See the detailed image in the github repository.

Summary

Blasthoge 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 manipulate the fees and mint tokens. if the contract owner abuses the mint functionality, then the contract will be highly inflated. 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

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