Cyberscope

Audit Report

APEX

April 2024

Repository https://github.com/DJHellscream/bifkn314

Commit 12c59e58aa1fbfcbfdb6f5683e29427bd105a034

Audited by © cyberscope

Table of Contents

Review

Repository	https://github.com/DJHellscream/bifkn314
Commit	12c59e58aa1fbfcbfdb6f5683e29427bd105a034
Testing Deploy	https://testnet.bscscan.com/address/0xeb46479ba6060b5e20ee1c524c2068e2aa197676

Audit Updates

Initial Audit	31 Mar 2024 https://github.com/cyberscope-io/audits/blob/main/9-apex/v1/audit.pdf
Corrected Phase 2	15 Apr 2024

Source Files

Filename	SHA256
contracts/PreventAutoSwap.sol	e7c8f18e514852e0338ba93f4311e0496518172e78bfd76d9b5978d2f1e7777e
contracts/ERC20.sol	130e88f3316b429cbade7f8d9175c858ad60fdda9f39ff8c5b967f9623b6f596
contracts/BIFKNERC20.sol	149a32ce7ed7e289b421df5ef8f2c1ba47ea5edffd3fe4a7f1051e222a4ae5a0
contracts/BIFKN314LP.sol	c865fd94eca6af5d506281ef663ba1c797e50b635c5a78b2738fb7eb5c40b147
contracts/BIFKN314Factory.sol	9ea3594660e85d38205dd2e64992102b0ce7c814a5a1af205e0ef78942c585cd

contracts/BIFKN314.sol	e3f82f313f25d39975014beac297c6b518ef0b14fa79a98d7759daacbade20b1
contracts/interfaces/IERC314Events.sol	539857e17d07a0a03a6cd6d5a5fc53cc90c8631455d2b65a518e805714842298
contracts/interfaces/IERC314Errors.sol	15973b1961a3d3e468e5d75ea9d0f74bca5c6574d288b82e1f6f4b33105fc6b1
contracts/interfaces/IERC314.sol	823b456a460b7304e58b240fb83d514887f0cd17fb57d74b6d4fe35ed5ecd4e8
contracts/interfaces/IBIFKN314Factory.sol	9ea3594660e85d38205dd2e64992102b0ce7c814a5a1af205e0ef78942c585cd
contracts/interfaces/IBIFKN314CALLEE.sol	1494f32b31b5ed41bbf32a73aabfb7ffda6c7b194904de08e8e86dda1c68cb65
@openzeppelin/contracts/utils/ReentrancyGuard.sol	8d0bac508a25133c9ff80206f65164cef959ec084645d1e7b06050c2971ae0fc
@openzeppelin/contracts/utils/Context.sol	847fda5460fee70f56f4200f59b82ae622bb03c79c77e67af010e31b7e2cc5b6
@openzeppelin/contracts/utils/math/Math.sol	a6ee779fc42e6bf01b5e6a963065706e882b016affbedfd8be19a71ea48e6e15
@openzeppelin/contracts/token/ERC20/IERC20.sol	6f2faae462e286e24e091d7718575179644dc60e79936ef0c92e2d1ab3ca3cee
@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol	1d079c20a192a135308e99fa5515c27acfbb071e6cdb0913b13634e630865939
@openzeppelin/contracts/interfaces/draft-IERC6093.sol	4aea87243e6de38804bf8737bf86f750443d3b5e63dd0fd0b7ad92f77cdbd3e3
@openzeppelin/contracts/access/Ownable.sol	38578bd71c0a909840e67202db527cc6b4e6b437e0f39f0c909da32c1e30cb81

Overview

BIFKN314

The BIFKN314 contract implements an Automated Market Maker (AMM) system and its operations, focusing on a specific token alongside the native currency of the blockchain. It inherits capabilities from BIFKNERC20, ReentrancyGuard, and interfaces such as IERC314Errors, IERC314Events, and IERC314, ensuring a robust framework for its functionalities.

Key functionalities include managing liquidity through addLiquidity and removeLiquidity functions. The contract also incorporates features for swapping between the native currency and the token it manages, enhancing its utility within the ecosystem. Furthermore, flashswap functionality is present, that allows users to execute flash swaps, a sophisticated trading mechanism enabling the exchange of an amount of native currency and tokens without requiring the initial balance of those assets.

Moreover, the contract has administrative functions, empowering the contract owner with the ability to manage and configure key operational parameters. These functions are instrumental in ensuring the contract's adaptability to different market conditions and governance requirements. Central to these capabilities is the facility to enable trading, designation of a fee collector, adjustment of the trading fee rate, etc.

BIFKN314Factory

The BIFKN314Factory contract serves as a deployment center for the BIFKN314 token contracts, facilitating the creation of new instances while managing their associated liquidity provider (LP) tokens. This contract embodies the automated market maker (AMM) model's principles by providing infrastructure for the seamless launch and integration of new tokens within its ecosystem. As an extension of its primary functionality, the BIFKN314Factory is designed to handle fee settings, including the designation of a feeTo address for collecting fees, a feeToSetter for administrative control over fees, and setting fee distribution thresholds to optimize operational efficiency.

BIFKNERC20

The BIFKNERC20 contract is a customizable ERC20 token with enhanced features. Designed with a focus on security and versatility, it incorporates advanced functionalities such as EIP-2612 permits and token burning, extending the standard capabilities of ERC20 tokens. Furthermore, it establishes a robust domain separation scheme through the DOMAIN_SEPARATOR, which is crucial for the EIP-2612 permit system.

BIFKN314LP

The BIFKN314LP contract is designed to function as the liquidity provider (LP) token within ecosystem. By inheriting the BIFKNERC20 standard, this contract extends the functionalities of an ERC20 token to specifically cater to the needs of liquidity provision in the AMM pool. The LP token represents a stake in the liquidity pool.

PreventAutoSwap

The PreventAutoSwap contract is designed to enhance control over automated token swapping mechanisms. Aimed at preventing unintended automatic swapping actions, this module is an essential utility for contracts that require granular management of their swapping functionalities where automated swaps can have significant implications on liquidity and token value.

Findings Breakdown

Critical	0
Medium	0
Minor / Informative	11

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	11	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
	BMWM	Bypassable Maximum Wallet Mechanism	Unresolved
	CCR	Contract Centralization Risk	Unresolved
	IFCV	Inaccurate Factory Contract Validation	Unresolved
	PTRP	Potential Transfer Revert Propagation	Unresolved
	RSW	Redundant Storage Writes	Unresolved
	SS	Stops Swaps	Unresolved
	UCD	Unreliable Contract Detection	Unresolved
	L13	Divide before Multiply Operation	Unresolved
	L16	Validate Variable Setters	Unresolved
	L17	Usage of Solidity Assembly	Unresolved
	L19	Stable Compiler Version	Unresolved

BMWM - Bypassable Maximum Wallet Mechanism

Criticality	Minor / Informative
Location	contracts/BIFKN314.sol#L1000
Status	Unresolved

Description

The `_checkMaxWallet` function within the contract is designed to restrict the amount of tokens a single wallet can hold. However, the function includes an exemption for contracts. However, a user can bypass the max wallet limit by having a contract to hold and manage their tokens. This bypass can undermine the intended protections and allow for accumulation of tokens beyond the set limits.

```
function _checkMaxWallet(address _recipient, uint256 _amount)
internal view{
    // Only apply the max wallet check if the recipient is not
this
    // and the recipient is also not a contract.
    // Need to allow for tokens to be sent to staking contracts
    if (_recipient == address(this) || _isContract(_recipient))
{
        return;
    }
    ...
}
```

Recommendation

It is recommended to refine the logic within the `_checkMaxWallet` function, so as to differentiate between legitimate contract interactions and exploitative use cases designed to bypass the wallet limits.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	contracts/BIFKN314.sol#L230,257,335,801,813...
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. Furthermore, the contract owner has to call functions like `addLiquidity` so that the system is functional afterwards.

```
function setTradingFeeRate(uint256 feeRate) public onlyOwner {
    if (feeRate > MAX_FEE_RATE) revert InvalidFeeRate(); // 5%
    tradingFeeRate = feeRate;

    emit TradingFeeRateSet(feeRate);
}

function setMaxWalletPercent(uint256 maxWalletPercent_) public
onlyOwner {
    if (maxWalletPercent_ > 10000) revert
InvalidMaxWalletPercent(); // 100%
    if (maxWalletEnabled && maxWalletPercent_ == 0)
        revert InvalidMaxWalletPercent();
    maxWalletPercent = maxWalletPercent_;

    emit MaxWalletPercentSet(maxWalletPercent_);
}
```

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.

IFCV - Inaccurate Factory Contract Validation

Criticality	Minor / Informative
Location	contracts/BIFKN314.sol#L209
Status	Unresolved

Description

The constructor of the `BIFKN314` contract employs a logic that automatically sets the factory address based on whether the sender is a contract or an externally owned account (EOA). This logic assigns the sender as the factory if it is a contract and defaults to `address(0)` if it is not, without further validating the nature or functionality of the contract sender. This approach can inadvertently grant factory privileges to any contract. Such an oversight could lead to scenarios where non-factory contracts obtain unintended access and capabilities.

```
constructor() BIFKNERC20() {
    address sender = _msgSender();
    address factoryAddress = sender;
    // if the sender is not a contract, set the factory address
to address(0)
    // This will disable fee distribution.
    // This is useful for testing and for EOA deployments
    // where the factory contract is not available.
    if (!_isContract(sender)) factoryAddress = address(0);

    factory = IBIFKN314Factory(factoryAddress);

    _transferOwnership(sender);
}
```

Recommendation

It is crucial to refine the validation mechanism within the constructor to ensure only designated and verified factory contracts are assigned the factory role. By enhancing the validation logic, the `BIFKN314` contract can better safeguard against unauthorized access and ensure its interactions are confined to legitimate and expected entities.

PTRP - Potential Transfer Revert Propagation

Criticality	Minor / Informative
Location	contracts/BIFKN314.sol#L1161
Status	Unresolved

Description

The contract employs a pattern where transfers are executed as part of key functionalities, including fee distribution and liquidity operations. These transfers carry the risk of causing the entire transaction to revert if the recipient fails to accept the Ether. If the address belongs to a contract then it may revert from incoming payment. As a result, the error will propagate to the token's contract and revert the transfer.

```
if (!success) revert FailedToSendNativeCurrency();
```

Recommendation

The contract should tolerate the potential revert from the underlying contracts when the interaction is part of the main transfer flow. This could be achieved by not allowing set contract addresses or by sending the funds in a non-revertable way.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	contracts/BIFKN314Factory.sol#L234,245,260,273,286 contracts/BIFKN314.sol#L813,827,842,857,870
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 setFeeTo(address feeTo_) external onlyFeeToSetter {
    feeTo = feeTo_;

    emit FeeToSet(feeTo_);
}

function setFeeToSetter(address feeToSetter_) external
onlyFeeToSetter {
    if (feeToSetter_ == address(0)) revert InvalidAddress();
    feeToSetter = feeToSetter_;

    emit FeeToSetterSet(feeToSetter_);
}

...
```

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.

SS - Stops Swaps

Criticality	Minor / Informative
Location	contracts/BIFKN314.sol#L1026
Status	Unresolved

Description

Initially, the swaps are disabled for all users. The contract owner has to set the `tradingEnabled` to true by calling the `setTradingEnabled` function.

```
function _checkForSwapErrors(
    uint256 tokensSold,
    uint256 nativeReserve,
    uint256 tokenReserve
) internal view {
    if (!isInitialized) revert ContractIsNotInitialized();
    if (!tradingEnabled) revert SwapNotEnabled();

function setTradingEnabled() public onlyOwner {
    tradingEnabled = true;
}
```

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.

UCD - Unreliable Contract Detection

Criticality	Minor / Informative
Location	contracts/BIFKN314.sol#L216
Status	Unresolved

Description

The constructor of the BIFKN314 contract utilizes the `_isContract` check to determine the nature of the message sender. This method checks if the bytecode size at the sender's address is greater than zero to conclude if it's a contract. While this approach is reliable for confirming the presence of a contract, its application to infer that an address is an externally owned account (EOA) when the check returns false can be unreliable. Relying on this method to set the factory address can lead to incorrect initialization if the address is misidentified due to the inherent limitations of the check used. This can impact the contract's functionality.

```
constructor() BIFKNERC20() {
    address sender = _msgSender();
    address factoryAddress = sender;
    // if the sender is not a contract, set the factory address
to address(0)
    // This will disable fee distribution.
    // This is useful for testing and for EOA deployments
    // where the factory contract is not available.
    if (!_isContract(sender)) factoryAddress = address(0);

    factory = IBIFKN314Factory(factoryAddress);

    _transferOwnership(sender);
}
```

Recommendation

It is recommended to revise the constructor logic to robustly confirm the sender’s status. Rather than relying on the absence of contract code to determine that an address is an EOA, the implementation should include correct checks for contract presence when setting variables like the factory address. This adjustment ensures that the factory address is set appropriately, preventing potential security and operational issues arising from incorrect address categorization.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	contracts/BIFKN314.sol#L366,378,397,1056,1059,1063,1064
Status	Unresolved

Description

It is important to be aware of the order of operations when performing arithmetic calculations. This is especially important when working with large numbers, as the order of operations can affect the final result of the calculation. Performing divisions before multiplications may cause loss of prediction.

```
uint256 price1Ratio = (tokenReserve * scalingFactor) /
                nativeReserve
price1CumulativeLast += price1Ratio * timeElapsed
```

Recommendation

To avoid this issue, it is recommended to carefully consider the order of operations when performing arithmetic calculations in Solidity. It's generally a good idea to use parentheses to specify the order of operations. The basic rule is that the multiplications should be prior to the divisions.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	contracts/BIFKN314Factory.sol#L233
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.

```
feeTo = feeTo_
```

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.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	contracts/BIFKN314.sol#L1174
Status	Unresolved

Description

Using assembly can be useful for optimizing code, but it can also be error-prone. It's important to carefully test and debug assembly code to ensure that it is correct and does not contain any errors.

Some common types of errors that can occur when using assembly in Solidity include Syntax, Type, Out-of-bounds, Stack, and Revert.

```
assembly {
            size := extcodesize(addressToCheck)
        }
```

Recommendation

It is recommended to use assembly sparingly and only when necessary, as it can be difficult to read and understand compared to Solidity code.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	contracts/BIFKN314.sol#L2
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 ^0.8.20;
```

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

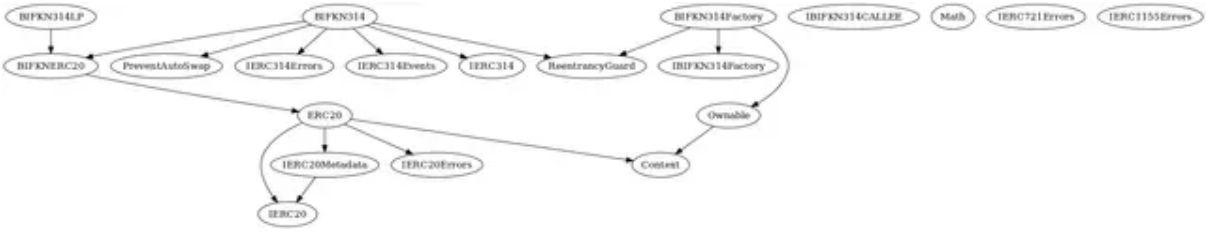
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
PreventAutoSwap	Implementation			
	_preventAutoSwapBefore	Private	✓	
	_preventAutoSwapAfter	Private	✓	
	_autoSwapIsPrevented	Internal		
BIFKNERC20	Implementation	ERC20		
		Public	✓	ERC20
	initialize	Public	✓	-
	permit	External	✓	-
	burn	Public	✓	-
	burnFrom	Public	✓	-
BIFKN314LP	Implementation	BIFKNERC20		
		Public	✓	BIFKNERC20
	initialize	Public	✓	onlyOwner
	mint	Public	✓	onlyOwner
BIFKN314Factory	Implementation	IBIFKN314Factory, ReentrancyG		

		uard, Ownable		
		Public	✓	-
	deployBIFKN314	External	Payable	nonReentrant
	allTokensLength	Public		-
	getAllTokens	Public		-
	getTokensByDeployer	Public		-
	setFeeTo	External	✓	onlyFeeToSetter
	setFeeToSetter	External	✓	onlyFeeToSetter
	setFeeRate	External	✓	onlyFeeToSetter
	setFeeDistributionThreshold	External	✓	onlyFeeToSetter
	setDeploymentFee	External	✓	onlyOwner
	withdrawFees	External	✓	onlyFeeToSetter
	_transferFees	Internal	✓	
	_checkConstructorParams	Internal		
BIFKN314	Implementation	BIFKNERC20, ReentrancyGuard, PreventAutoSwap, IERC314Errors, IERC314Events, IERC314		
		Public	✓	BIFKNERC20
	initialize	Public	✓	onlyOwner
	setSupplyAndMint	Public	✓	onlyOwner

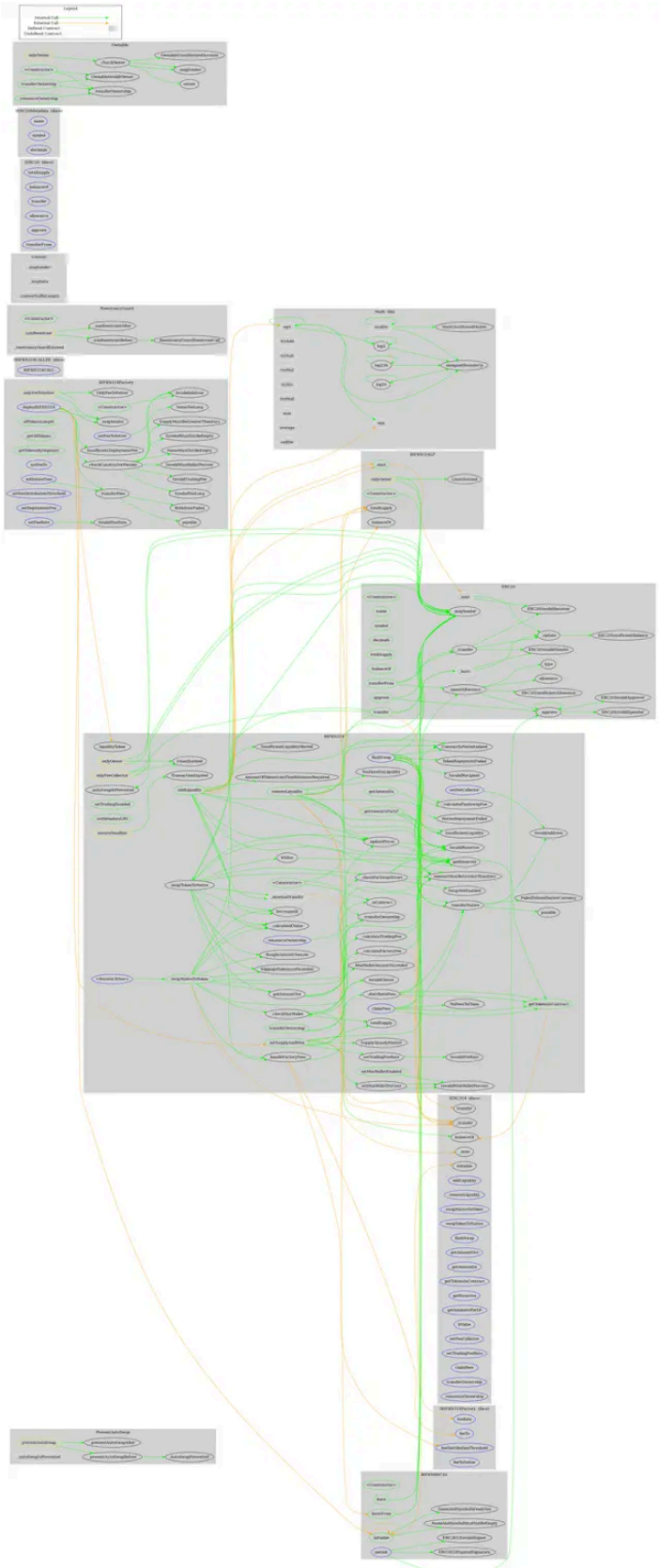
	transfer	Public	✓	-
	_internalTransfer	Internal	✓	
	addLiquidity	Public	Payable	nonReentrant ensureDeadline
	removeLiquidity	Public	✓	nonReentrant ensureDeadline
	swapNativeToToken	Public	Payable	nonReentrant ensureDeadline
	swapTokenToNative	Public	✓	nonReentrant ensureDeadline
	flashSwap	External	Payable	nonReentrant preventAutoSwap
	getAmountOut	Public		-
	getAmountIn	Public		-
	getTokensInContract	Public		-
	getReserves	Public		-
	getAmountsForLP	Public		-
	kValue	Public		-
	setTradingEnabled	Public	✓	onlyOwner
	setFeeCollector	External	✓	onlyOwner
	setTradingFeeRate	Public	✓	onlyOwner
	setMaxWalletPercent	Public	✓	onlyOwner
	setMaxWalletEnabled	Public	✓	onlyOwner
	setMetadataURI	Public	✓	onlyOwner
	claimFees	External	✓	onlyFeeCollector
	transferOwnership	Public	✓	onlyOwner
	renounceOwnership	External	✓	onlyOwner

	_transferOwnership	Internal	✓	
	_calculateKValue	Internal		
	_calculateTradingFee	Internal		
	_calculateFactoryFee	Internal		
	_checkMaxWallet	Internal		
	_checkForSwapErrors	Internal		
	_updatePrices	Private	✓	
	_handleFactoryFees	Internal	✓	
	_distributeFees	Internal	✓	
	_transferNative	Internal	✓	
	_isContract	Internal		
	_calculateFlashswapFee	Internal		
		External	Payable	-
IBIFKN314Factory	Interface			
	feeTo	External		-
	feeRate	External		-
	feeToSetter	External		-
	feeDistributionThreshold	External		-

Inheritance Graph



Flow Graph



Summary

APEX contract implements a token, utility, financial, exchange and rewards mechanism. This audit investigates security issues, business logic concerns and potential improvements.

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