Cyberscope

Audit Report

APEX

April 2024

Repository	https://github.com/DJHellscream/bifkn314
Commit	71c6d4cc9c841f6e1786b694cb109d879b5495ee
Audited by	© cyberscope

Table of Contents

Review

Repository	https://github.com/DJHellscream/bifkn314
Commit	71c6d4cc9c841f6e1786b694cb109d879b5495ee
Testing Deploy	https://testnet.bscscan.com/address/0x458836c3e0689c57781bb2932cd452c7669d0403

Audit Updates

Initial Audit	31 Mar 2024

Source Files

Filename	SHA256
contracts/PreventAutoSwap.sol	7c662f9d4473e983ed292c6b02a8541e1ec7f5fa9517e7a418e8ab7107d2c248
contracts/ERC20.sol	619ec216849e46ff8bf5e9747b1cd1ca80e0824a8b7cdc3acfc7f457f1552325
contracts/BIFKNERC20.sol	beef568641aad50c32aebd746baccd30eb37d0ae9a31858cca3635de3eb3de63
contracts/BIFKN314LP.sol	34bdb3f27c587f7c21c45100e8d7ef52f12b3b1027ee859def90105432a06e89
contracts/BIFKN314Factory.sol	fbe43941b5d94872aa39e0d42e5e888844dc31061a3f69d5d9d3d13b394885d7
contracts/BIFKN314.sol	759aca423b07bad7b3177158258e65e6cd6017c99952e9e28b3c6c267752dfd9
contracts/interfaces/IERC314Events.sol	017f45f7db9593fa166a348b2529e73f902615218f6917573f70da5456bc0efc

contracts/interfaces/IERC314Errors.sol	d9d3c45c91394536825b83c79d94e0b79f61aae001ed4c924320f5aebfe3e64e
contracts/interfaces/IERC314.sol	52ac3dd8b4f39c7e87b98987df8c85fb5ebe4c4377551cb57c7f803e1b2ff460
contracts/interfaces/IBIFKN314Factory.sol	1d01983dd94af0637072a846ad5f7b2a5218c0e531503ceb6a91dbfb3e31d732
contracts/interfaces/IBIFKN314CALLEE.sol	d8453c19e3d38fdd451aff86903ac4e7f11f109dfc8d8a558b63777f0a73166d
@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, thatallows 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.

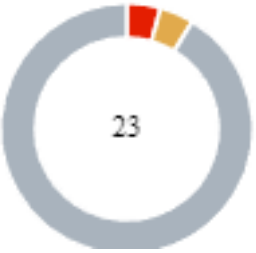
BIFKN314LP

The BIFKN314LP contract is designed to function as the liquidity provider (LP) token within the 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	23

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	1	0	0	0
Medium	1	0	0	0
Minor / Informative	21	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
	EWMA	Ether Withdrawal Mechanism Absence	Unresolved
	SORE	Swap Outputs Rounding Errors	Unresolved
	BMWM	Bypassable Maximum Wallet Mechanism	Unresolved
	CCR	Contract Centralization Risk	Unresolved
	IDI	Immutable Declaration Improvement	Unresolved
	IFCV	Inaccurate Factory Contract Validation	Unresolved
	ITAC	Inaccurate Token Amount Calculation	Unresolved
	MEE	Misleading Event Emission	Unresolved
	MPD	Misleading Parameter Description	Unresolved
	MEE	Missing Events Emission	Unresolved
	PTRP	Potential Transfer Revert Propagation	Unresolved
	PUR	Potential Underflow Risk	Unresolved
	RFAI	Redundant Factory Address Information	Unresolved
	RSM	Redundant State Modification	Unresolved

●	RSW	Redundant Storage Writes	Unresolved
●	RCGO	Reserve Calculation Gas Optimization	Unresolved
●	SS	Stops Swaps	Unresolved
●	UC	Unreachable Condition	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved
●	L19	Stable Compiler Version	Unresolved

EWMA - Ether Withdrawal Mechanism Absence

Criticality	Critical
Location	contracts/BIFKN314Factory.sol#L120
Status	Unresolved

Description

The `BIFKN314Factory` contract accumulates Ether through the `deployBIFKN314` function, which requires a deployment fee in order to deploy new `BIFKN314` contract instances. This design inherently leads to the accumulation of Ether within the contract without a predefined method for its withdrawal. The lack of an explicit withdrawal mechanism results in Ether being effectively locked within the contract.

```
function deployBIFKN314(
        string memory _tokenName,
        string memory _tokenSymbol,
        uint256 _totalSupply,
        address _owner,
        uint256 _tradingFee,
...
        _tokenName,
        _tokenSymbol,
        _contractAddress,
        _liquidityTokenAddress,
        allTokens.length
    );
}
```

Recommendation

It is recommended that a withdrawal function is added to the contract. This function should be designed to allow the contract owner, or another authorized party, to transfer accumulated Ether to a designated address.

SORE - Swap Outputs Rounding Errors

Criticality	Medium
Location	contracts/BIFKN314.sol#L681,719
Status	Unresolved

Description

The `getAmountOut` and `getAmountIn` functions are responsible for determining the dynamics of token swaps. These functions are designed to calculate the amount of tokens either received or required for swaps, factoring in reserves and associated fees. A notable concern arises from Solidity's treatment of integer division, which truncates fractional results, potentially leading to rounding errors that impact the accuracy of these calculations. Certain scenarios are present, specifically in the `getAmountOut`, where this issue manifests as a risk of yielding a 0 output for valid non-zero inputs under certain conditions. This is a scenario that could significantly impair the swap's intended operations.

```
function getAmountOut(
    uint256 _inputAmount,
    uint256 _inputReserve,
    uint256 _outputReserve
)
    public
    view
    returns (
        uint256 _outputAmount,
        uint256 _factoryFee,
        uint256 _tradingFee
    )
{
    uint256 feeFactor = SCALE_FACTOR - (BASE_SWAP_RATE +
tradingFeeRate);
    // if reserves are greater than 0
    if (_inputReserve > 0 && _outputReserve > 0) {
        _factoryFee = _calculateFactoryFee(_inputAmount);
        _tradingFee = _calculateTradingFee(_inputAmount);
        uint256 inputAmountWithFee = _inputAmount * feeFactor;
        uint256 numerator = inputAmountWithFee *
_outputReserve;
        uint256 denominator = (_inputReserve * SCALE_FACTOR) +
            inputAmountWithFee;
        unchecked {
            _outputAmount = numerator / denominator;
        }
    } else {
        revert InvalidReserves();
    }
}
```

Recommendation

To mitigate this issue, it's recommended to review and refine the swap calculation logic to improve its handling of division and rounding. Enhancing the precision and reliability of these calculations, by minimizing the impact of Solidity's integer division behavior. Furthermore, it is suggested toimplement more detailed validation checks before and after calculations to ensure swap integrity and prevent scemarios that could result in extreme rounding discrepancies.

BMWM - Bypassable Maximum Wallet Mechanism

Criticality	Minor / Informative
Location	contracts/BIFKN314.sol#L999
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 {
    // 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#L233,257,335,805,815,...
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 the authority to make changes to key parameters of the contract, that can severely modify its functionality. 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;
}

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

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	contracts/BIFKN314LP.sol#L35
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.

```
ammAddress
```

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.

IFCV - Inaccurate Factory Contract Validation

Criticality	Minor / Informative
Location	contracts/BIFKN314.sol#L211
Status	Unresolved

Description

The constructor logic within the `BIFKN314` contract attempts to check the nature of the deployer based on whether the sender is a contract or an externally owned account (EOA), setting the factory address accordingly. This logic presumes that any contract deployer must be the intended factory contract, automatically assigning the deployer's address as the factory. However, this assumption overlooks the possibility that other contracts, not associated with the intended factory functionality could deploy the BIFKN314 contract. Such a scenario could inadvertently grant unintended contracts the status and privileges intended for the factory contract.

```
constructor() BIFKNERC20() {
    address sender = _msgSender();
    // This assumes the contract is deployed by the factory
contract
    if (_isContract(sender)) {
        factory = IBIFKN314Factory(sender);
        factoryAddress = sender;
```

Recommendation

To ensure that the `BIFKN314` contract is only associated with the legitimate factory contract, a more robust validation mechanism should be implemented. Instead of relying solely on the deployer being a contract, additional checks should be introduced to verify the deployer's identity.

ITAC - Inaccurate Token Amount Calculation

Criticality	Minor / Informative
Location	contracts/BIFKN314.sol#L478
Status	Unresolved

Description

The `removeLiquidity` function utilizes `getTokensInContract()` to determine the amount of tokens to be returned to the liquidity provider. This approach, while functional, does not accurately reflect the underlying mechanics of liquidity pools as defined in AMM protocols. Specifically, it bypasses the concept of token reserves, which are pivotal for maintaining the constant product formula and ensuring the balance and integrity of the liquidity pool. By not accounting for the token reserves directly in the calculation, the process omits the impact of accrued trading fees on the reserves, which could lead to discrepancies in the amounts returned to liquidity providers.

```
(uint256 _nativeReserve, ) = getReserves();

_nativeAmount = (_nativeReserve * _amount) / lpTotalSupply;
_tokenAmount = (getTokensInContract() * _amount) /
lpTotalSupply;
```

Recommendation

To align with the standard practices of AMM protocols and ensure the accurate calculation of amounts during the liquidity removal process, it is recommended to modify the `removeLiquidity` function to utilize the token reserves obtained from the `getReserves()` function directly. Such an adjustment would ensure that the calculation accurately reflects the pool's state, including the impact of trading activities and accrued fees, thereby maintaining the integrity of the liquidity pool.

MEE - Misleading Event Emission

Criticality	Minor / Informative
Location	contracts/BIFKN314.sol#L1102
Status	Unresolved

Description

In the `_distributeFees` internal function, an event `FeeDistributed` is emitted at the function's conclusion, signaling the completion of fee distribution. This event is triggered regardless of whether the conditions for fee distribution, specifically the threshold check for token fees based on their equivalent native amount, have been met. The function checks if the native fees or the token fees (converted to their native equivalent) meet or exceed a predefined distribution threshold before proceeding with the distribution. However, the event emission does not conditionally depend on the actual execution of these distributions. As a result, the event could misleadingly indicate that fees were distributed when, in reality, the distribution may not have occurred due to the threshold condition not being satisfied for token fees. This discrepancy could lead to confusion and inaccuracies in interpreting the contract's state and actions, particularly in tracking fee distributions.

```
function _distributeFees(
    address _feeTo,
    uint256 _distributionThreshold
) internal {
        ...
        if (_nativeAmount >= _distributionThreshold) {
            if (_tokenFees > getTokensInContract()) {
                _tokenFees = getTokensInContract();
            }
            super._transfer(address(this), _feeTo, _tokenFees);
            accruedTokenFactoryFees = 0;
        }
    }

    emit IBIFKN314Factory.FeeDistributed(_feeTo, _nativeFees,
_tokenFees);
}
```

Recommendation

To ensure accurate representation of contract actions, it is recommended to conditionally emit the FeeDistributed event only after successful fee distributions.

MPD - Misleading Parameter Description

Criticality	Minor / Informative
Location	contracts/BIFKN314.sol#L768
Status	Unresolved

Description

The `getAmountsForLP` function documentation inaccurately describes the `_amount` parameter as "The address of the liquidity provider," which is misleading. This parameter is actually of type `uint256` and represents an amount, not an address. Such inaccuracies in documentation can lead to confusion about the function's purpose and usage.

```
/**
     * @dev Gets the amount of tokens held by the liquidity
provider.
     * @param _amount The address of the liquidity provider.
     * @return _nativeAmount The amount of native currency held
by the liquidity provider.
     * @return _tokenAmount The amount of tokens held by the
liquidity provider.
     */
    function getAmountsForLP(
        uint256 _amount
    ) public view returns (uint256 _nativeAmount, uint256
_tokenAmount) {
        if (_amount == 0) revert AmountMustBeGreaterThanZero();
        (uint256 _nativeReserve, uint256 _tokenReserve) =
getReserves();

        if (_nativeReserve == 0 || _tokenReserve == 0) revert
InvalidReserves();

        uint256 _holderBalance = _amount;
        uint256 _totalLPSupply = liquidityToken.totalSupply();

        if (_totalLPSupply > 0) {
            _nativeAmount = (_holderBalance * _nativeReserve) /
_totalLPSupply;
            _tokenAmount = (_holderBalance * _tokenReserve) /
_totalLPSupply;
        } else {
            revert InsufficientLiquidity();
        }
    }
```

Recommendation

It is recommended to update the comment/documentation for the `getAmountsForLP` function to accurately reflect the nature of the _amount parameter. This correction will ensure clarity and reduce the potential for misunderstandings.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	contracts/BIFKN314Factory.sol#L227,236,249,260,271 contracts/BIFKN314.sol#L805,815,827,840,864
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 setFeeTo(address _feeTo) external onlyFeeToSetter {
    feeTo = _feeTo;
}

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

function setFeeRate(uint256 _feeRate) external onlyFeeToSetter
{
    if (_feeRate > 10) revert InvalidFeeRate();

    feeRate = _feeRate;
}

...
```

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.

PTRP - Potential Transfer Revert Propagation

Criticality	Minor / Informative
Location	contracts/BIFKN314.sol#L892,1148
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.

PUR - Potential Underflow Risk

Criticality	Minor / Informative
Location	contracts/BIFKN314.sol#L750
Status	Unresolved

Description

The contract includes reserve calculation logic that subtracts accrued fees from the contract's balance (`address(this).balance` for native currency, and `getTokensInContract()` for the token balance). This design inherently assumes that the total accrued fees will never exceed the total balance of the contract. However, in scenarios where accrued fees are allowed to accumulate without distribution, there's a potential risk that these deductions could exceed the available balance, which can lead to an underflow.

```
function getReserves()
    public
    view
    returns (uint256 _amountNative, uint256 _amountToken)
{
    _amountNative =
        address(this).balance -
        accruedNativeTradingFees -
        accruedNativeFactoryFees;

    _amountToken =
        getTokensInContract() -
        accruedTokenTradingFees -
        accruedTokenFactoryFees;
}
```

Recommendation

It is recommended to implement a mechanism for regularly monitoring and managing accrued fees. Accrued fees should be reviewed and compared to the contract’s total balance to ensure that the subtraction operations will not exceed the available funds.

RFAI - Redundant Factory Address Information

Criticality	Minor / Informative
Location	contracts/BIFKN314.sol#L212
Status	Unresolved

Description

Within the constructor of the `BIFKN314` contract, there exists an unnecessary duplication of information related to the factory address. Specifically, the address of the factory contract is stored in two separate state variables. This redundancy leads to an increased gas cost during contract deployment due to the additional storage operation required.

```
constructor() BIFKNERC20() {
    address sender = _msgSender();
    // This assumes the contract is deployed by the factory
contract
    if (_isContract(sender)) {
        factory = IBIFKN314Factory(sender);
        factoryAddress = sender;
        ...
```

Recommendation

It is recommended to store the factory address into a single state variable. This approach would reduce the gas cost associated with contract deployment by eliminating redundant storage operations. It would also simplify the contract's structure, making it easier to understand, maintain, and modify in the future.

RSM - Redundant State Modification

Criticality	Minor / Informative
Location	contracts/BIFKN314.sol#L1005
Status	Unresolved

Description

Within the `_checkMaxWallet` function of the contract, there exists a conditional block that modifies the state variable `maxWalletEnabled` by setting it to `false`, immediately before executing a revert statement due to an `InvalidMaxWalletPercent` condition. This attempt to update the contract's state has no meaning, as the revert operation undoes all changes made to the state in the current transaction.

```
if (maxWalletEnabled) {
    // This check is a fail-safe to prevent a weird state
    // where the maxWalletPercent is set to 0 but
maxWalletEnabled is true
    if (maxWalletPercent == 0) {
        maxWalletEnabled = false;
        revert InvalidMaxWalletPercent();
            }
```

Recommendation

It is advisable to remove the state modification operation that precedes the `revert` statement. Since the revert negates all state changes executed in the transaction, including this modification serves no practical purpose.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	contracts/BIFKN314.sol#L815,827,840,864 contracts/BIFKN314Factory.sol#L227,236,249,260,271
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;
}

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

function setFeeRate(uint256 _feeRate) external onlyFeeToSetter
{
    if (_feeRate > 10) revert InvalidFeeRate();

    feeRate = _feeRate;
}

...
```

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.

RCGO - Reserve Calculation Gas Optimization

Criticality	Minor / Informative
Location	contracts/BIFKN314.sol#L505,560,750
Status	Unresolved

Description

The `swapTokenToNative` and `swapNativeToToken` functions within the contract both exhibit a pattern of calling `_checkForSwapErrors`, which internally fetches the reserve balances through `getReserves()`, followed by a direct subsequent call to `getReserves()`. This approach unnecessarily increases gas consumption.

```
function swapTokenToNative(
    uint256 _tokensSold,
    uint256 _minimumNativeOut,
    uint256 _deadline
) public nonReentrant ensureDeadline(_deadline) {
    _checkForSwapErrors(_tokensSold);

    address sender = _msgSender();
    (uint256 _nativeReserve, uint256 _tokenReserve) =
getReserves();

    uint256 _currentKValue = kValue();
    ...

function _checkForSwapErrors(uint256 _tokensSold) internal view
{
    if (!isInitialized) revert ContractIsNotInitialized();
    if (!tradingEnabled) revert SwapNotEnabled();
    if (_tokensSold == 0) {
        revert AmountMustBeGreaterThanZero();
    }
    (uint256 _nativeReserve, uint256 _tokenReserve) =
getReserves();
    if (_nativeReserve == 0 || _tokenReserve == 0) revert
InvalidReserves();
}
```

Recommendation

To address the identified inefficiency and optimize gas usage across swap operations, a refactoring is recommended to consolidate the reserve balance calculations within the `swapTokenToNative` and `swapNativeToToken` functions. By adjusting the internal logic to call `getReserves()` once at the beginning of each swap function and passing the obtained reserve balances as arguments to any subsequent functions requiring this data, the contract can avoid redundant calculations.

SS - Stops Swaps

Criticality	Minor / Informative
Location	contracts/BIFKN314.sol#L1024
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) internal view
{
    if (!isInitialized) revert ContractIsNotInitialized();
    if (!tradingEnabled) revert SwapNotEnabled();
    if (_tokensSold == 0) {
        revert AmountMustBeGreaterThanZero();
    }
    (uint256 _nativeReserve, uint256 _tokenReserve) =
getReserves();
    if (_nativeReserve == 0 || _tokenReserve == 0) revert
InvalidReserves();
}

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.

UC - Unreachable Condition

Criticality	Minor / Informative
Location	contracts/BIFKN314.sol#L1002
Status	Unresolved

Description

The `_checkMaxWallet` function includes a conditional check for an invalid state where `maxWalletPercent` is zero while `maxWalletEnabled` is true. This check is designed as a fail-safe against inconsistent configuration of maximum wallet limits. However, the contract's setter functions are constructed to prevent this scenario from occurring. These protective measures ensure that `maxWalletPercent` cannot be zero when `maxWalletEnabled` is set to true, rendering the conditional check within `_checkMaxWallet` obsolete.

```
if (maxWalletEnabled) {
    // This check is a fail-safe to prevent a weird state
    // where the maxWalletPercent is set to 0 but
maxWalletEnabled is true
    if (maxWalletPercent == 0) {
        maxWalletEnabled = false;
        revert InvalidMaxWalletPercent();
    }
```

Recommendation

Considering the contract's design prevents the `maxWalletPercent` from being zero while `maxWalletEnabled` is true, it is recommended to remove the redundant conditional check from the `_checkMaxWallet` function.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	contracts/BIFKNERC20.sol#L16 contracts/BIFKN314Factory.sol#L121,122,123,124,125,126,127,215,225,234,247,259,269 contracts/BIFKN314.sol#L234,235,258,259,260,261,262,298,299,336,337,338,448,449,450,502,503,556,557,558,608,609,610,611,682,683,684,720,721,722,773,815,827,840,853,911
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.

```
bytes32 public DOMAIN_SEPARATOR
string memory _tokenName
string memory _tokenSymbol
uint256 _totalSupply
address _owner
uint256 _tradingFee
uint256 _maxWalletPercent
string memory _metadataURI
address _deployer
address _feeTo
address _feeToSetter
uint256 _feeRate
uint256 _feeDistributionThreshold
uint256 _deploymentFee

...
```

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.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	contracts/BIFKN314.sol#L366,378,397,1052,1055,1059,1060
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 _price0Ratio = (_nativeReserve * _scalingFactor) /
                _tokenReserve
price0CumulativeLast += _price0Ratio * _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#L226 contracts/BIFKN314.sol#L891
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
(bool success, ) = payable(sender).call{value:
accruedNativeAmount}("")
```

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#L1161
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(_address)
        }
```

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/PreventAutoSwap.sol#L3 contracts/interfaces/IBIFKN314Factory.sol#L2 contracts/BIFKNERC20.sol#L3 contracts/BIFKN314LP.sol#L3 contracts/BIFKN314Factory.sol#L2 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
	_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		-

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