Cyberscope

Audit Report

GemPad

March 2024

Network	BSC
Address	0x78aae7e000bf6fc98a6b717d5ec8ef2bcd04f428
Audited by	© cyberscope

Table of Contents

Table of Contents 1
Review 3
Audit Updates 3
Source Files 4
Findings Breakdown 5
Diagnostics 6
DDP - Decimal Division Precision 8
Description 8
Recommendation 9
FRV - Fee Restoration Vulnerability 10
Description 10
Recommendation 11
FVC - Fee Variable Consolidation 12
Description 12
Recommendation 12
IDI - Immutable Declaration Improvement 14
Description 14
Recommendation 14
IVU - Inefficient Variable Usage 15
Description 15
Recommendation 16
MEE - Missing Events Emission 17
Description 17
Recommendation 17
MU - Modifiers Usage 19
Description 19
Recommendation 19
PLPI - Potential Liquidity Provision Inadequacy 20
Description 20
Recommendation 20
PVC - Price Volatility Concern 22
Description 22
Recommendation 23
RSML - Redundant SafeMath Library 24
Description 24
Recommendation 24
RSW - Redundant Storage Writes 25
Description 25
Recommendation 25

RVU - Redundant Variable Usage 27
Description 27
Recommendation 27
RC - Repetitive Calculations 28
Description 28
Recommendation 28
ST - Stops Transactions 30
Description 30
Recommendation 30
UTE - Unchecked Transfer Emission 32
Description 32
Recommendation 32
US - Untrusted Source 34
Description 34
Recommendation 34
L04 - Conformance to Solidity Naming Conventions 35
Description 35
Recommendation 36
L07 - Missing Events Arithmetic 37
Description 37
Recommendation 37
L16 - Validate Variable Setters 38
Description 38
Recommendation 38
Functions Analysis **39**
Inheritance Graph **41**
Flow Graph **42**
Summary **43**
Disclaimer **44**
About Cyberscope **45**

Review

Contract Name	GEMS
Compiler Version	v0.8.13+commit.abaa5c0e
Optimization	200 runs
Explorer	https://bscscan.com/address/0x78aae7e000bf6fc98a6b717d5ec8ef2bcd04f428
Address	0x78aae7e000bf6fc98a6b717d5ec8ef2bcd04f428
Network	BSC
Symbol	GEMS
Decimals	9
Total Supply	100,000,000
Badge Eligibility	Yes

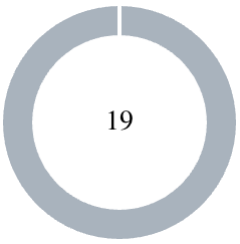
Audit Updates

Initial Audit	15 Mar 2024

Source Files

Filename	SHA256
contracts/GEMS.sol	2cc4a66fda216fed324064a95e5bc49349f49db1e74bbf8532f50a9bb94ba3bb
@openzeppelin/contracts/utils/Context.sol	1458c260d010a08e4c20a4a517882259a23a4baa0b5bd9add9fb6d6a1549814a
@openzeppelin/contracts/utils/math/SafeMath.sol	15941f3904992a62ed117e93d9e2d5c4c22bd09a7ff97fdd5f49273cf09703ac
@openzeppelin/contracts/token/ERC20/IERC20.sol	c2b06bb4572bb4f84bfc5477dadc0fcc497cb66c3a1bd53480e68bedc2e154a6
@openzeppelin/contracts/token/ERC20/ERC20.sol	f7831910f2ed6d32acff6431e5998baf50e4a00121303b27e974aab0ec637d79
@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol	af5c8a77965cc82c33b7ff844deb9826166689e55dc037a7f2f790d057811990
@openzeppelin/contracts/access/Ownable.sol	75e3c97011e75627ffb36f4a2799a4e887e1a3e27ed427490e82d7b6f51cc5c9

Findings Breakdown



Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	19	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
	DDP	Decimal Division Precision	Unresolved
	FRV	Fee Restoration Vulnerability	Unresolved
	FVC	Fee Variable Consolidation	Unresolved
	IDI	Immutable Declaration Improvement	Unresolved
	IVU	Inefficient Variable Usage	Unresolved
	MEE	Missing Events Emission	Unresolved
	MU	Modifiers Usage	Unresolved
	PLPI	Potential Liquidity Provision Inadequacy	Unresolved
	PVC	Price Volatility Concern	Unresolved
	RSML	Redundant SafeMath Library	Unresolved
	RSW	Redundant Storage Writes	Unresolved
	RVU	Redundant Variable Usage	Unresolved
	RC	Repetitive Calculations	Unresolved
	ST	Stops Transactions	Unresolved

●	UTE	Unchecked Transfer Emission	Unresolved
●	US	Untrusted Source	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L16	Validate Variable Setters	Unresolved

DDP - Decimal Division Precision

Criticality	Minor / Informative
Location	contracts/GEMS.sol#L818
Status	Unresolved

Description

Division of decimal (fixed point) numbers can result in rounding errors due to the way that division is implemented in Solidity. Thus, it may produce issues with precise calculations with decimal numbers.

Solidity represents decimal numbers as integers, with the decimal point implied by the number of decimal places specified in the type (e.g. decimal with 18 decimal places). When a division is performed with decimal numbers, the result is also represented as an integer, with the decimal point implied by the number of decimal places in the type. This can lead to rounding errors, as the result may not be able to be accurately represented as an integer with the specified number of decimal places.

Hence, the splitted shares will not have the exact precision and some funds may not be calculated as expected.

```
    _rewardFee = amount.mul(_protectBlockRewardFee).div(1000);
    _liquidityFee =
amount.mul(_protectBlockLiquidityFee).div(1000);
    _marketingFee =
amount.mul(_protectBlockMarketingFee).div(1000);
    buyOrSellSwitch = PROTECT;
}
// Buy
else if (automatedMarketMakerPairs[from]) {
    _rewardFee = amount.mul(buyRewardFee).div(1000);
    _liquidityFee = amount.mul(buyLiquidityFee).div(1000);
    _marketingFee = amount.mul(buyMarketingFee).div(1000);
    buyOrSellSwitch = BUY;
}
// Sell
else if (automatedMarketMakerPairs[to]) {
    _rewardFee = amount.mul(sellRewardFee).div(1000);
    _liquidityFee = amount.mul(sellLiquidityFee).div(1000);
    _marketingFee = amount.mul(sellMarketingFee).div(1000);
    buyOrSellSwitch = SELL;
} else {
    _rewardFee = amount.mul(transferRewardFee).div(1000);
    _liquidityFee = amount.mul(transferLiquidityFee).div(1000);
    _marketingFee = amount.mul(transferMarketingFee).div(1000);
}
```

Recommendation

The team is advised to take into consideration the rounding results that are produced from the solidity calculations. The contract could calculate the subtraction of the divided funds in the last calculation in order to avoid the division rounding issue.

FRV - Fee Restoration Vulnerability

Criticality	Minor / Informative
Location	contracts/GEMS.sol#L860,871
Status	Unresolved

Description

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

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

```
    function removeAllFee() private {
        if (_rewardFee == 0 && _liquidityFee == 0 && _marketingFee
== 0) return;

        _previousRewardFee = _rewardFee;
        _previousLiquidityFee = _liquidityFee;
        _previousMarketingFee = _marketingFee;
        _rewardFee = 0;
        _liquidityFee = 0;
        _marketingFee = 0;
    }

    function restoreAllFee() private {
        _rewardFee = _previousRewardFee;
        _liquidityFee = _previousLiquidityFee;
        _marketingFee = _previousMarketingFee;
    }
```

Recommendation

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

FVC - Fee Variable Consolidation

Criticality	Minor / Informative
Location	contracts/GEMS.sol#L397
Status	Unresolved

Description

The contract is currently managing its fee variables through multiple distinct state variables, specifically for different transaction types such as buys, sells, and transfers. Each transaction type has its own set of reward, liquidity, and marketing fees, leading to a total of nine separate variables to manage these fees. This approach, while functional, results in increased complexity and redundancy within the contract's codebase. Managing these fees as separate variables can make the contract more difficult to read, maintain, and update, especially when changes to fee structures are required. Additionally, this method increases the risk of errors during updates, as each fee variable must be individually modified.

```
uint16 public buyRewardFee;
uint16 public buyLiquidityFee;
uint16 public buyMarketingFee;
uint16 public sellRewardFee;
uint16 public sellLiquidityFee;
uint16 public sellMarketingFee;
uint16 public transferRewardFee;
uint16 public transferLiquidityFee;
uint16 public transferMarketingFee;
```

Recommendation

It is recommended to consolidate these fee variables into a structured format, utilizing a struct to group related fees together based on their transaction type. For example, creating a `FeeStructure` struct that contains rewardFee, liquidityFee, and marketingFee as fields, and then using three instances of this struct for buy, sell, and transfer fees, respectively. This approach simplifies the contract by reducing the number of state variables and centralizing fee management, making the contract easier to read, maintain, and update. It also minimizes the risk of errors during fee adjustments, as related fees are grouped

together logically. Implementing this recommendation will enhance the contract's clarity and maintainability, aligning with best practices in smart contract development.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	contracts/GEMS.sol#L520
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.

```
pancakePair
```

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.

IVU - Inefficient Variable Usage

Criticality	Minor / Informative
Location	contracts/GEMS.sol#L381,793
Status	Unresolved

Description

The contract is utilizing a `buyOrSellSwitch` variable to indicate the type of transaction being processed (e.g., `BUY`, `SELL`, `TRANSFER`, `PROTECT`). This variable is updated based on the transaction context, such as whether the transaction is a buy, sell, or transfer, and the protection block. Despite being updated in various parts of the contract to reflect the transaction type, the `buyOrSellSwitch` variable is declared as `private` and does not influence any functional aspect of the contract beyond its assignment. This lack of utilization renders the updates to `buyOrSellSwitch` redundant, as its value does not contribute to the contract's logic or outcome. The presence of this variable without a corresponding use case within the contract's logic can lead to confusion and unnecessary complexity, detracting from the contract's clarity and efficiency.

```
uint8 private constant BUY = 1;
uint8 private constant SELL = 2;
uint8 private constant TRANSFER = 3;
uint8 private constant PROTECT = 4;
uint8 private buyOrSellSwitch;
...
buyOrSellSwitch = TRANSFER
if (...
    ) {
        ...
        buyOrSellSwitch = PROTECT;
    }
    // Buy
    else if (automatedMarketMakerPairs[from]) {
        ...
        buyOrSellSwitch = BUY;
    }
    // Sell
    else if (automatedMarketMakerPairs[to]) {
        ...
        buyOrSellSwitch = SELL;
```

Recommendation

It is recommended to reconsider the usage of the `buyOrSellSwitch` variable. Since the value is updated based on each transfer case, the contract could leverage this by using it to emit an event that indicates the type of transaction being processed. This would not only give purpose to the `buyOrSellSwitch` variable but also enhance the contract's transparency and provide stakeholders with valuable information about transaction types in real-time. Emitting an event based on the `buyOrSellSwitch` value could facilitate tracking and analysis of transaction types without impacting the contract's core functionality. If the variable remains unused and does not contribute to the contract's logic or informational output, removing it could simplify the contract and reduce unnecessary code, aligning with best practices for smart contract development.

MEE - Missing Events Emission

Criticality	Minor / Informative
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 updateProtectBlockCount(
  uint256 protectBlockCount
) external onlyOwner {
  _protectBlockCount = protectBlockCount;
  require(
    _protectBlockCount <= maxProtectBlockLimit,
    "Exceeds max protect block"
  );
}
..
function updateGasPriceLimit(uint256 gas) external onlyOwner {
    _gasPriceLimit = gas * 1 gwei;
    require(10000000 < _gasPriceLimit, "gasPricelimit > 10000000");
}
..
function disableTransferDelay() external onlyOwner {
    _transferDelayEnabled = false;
}
...
function setProtectBlockFee(
    uint16 protectBlockRewardFee,
    uint16 protectBlockLiquidityFee,
    uint16 protectBlockMarketingFee
) external onlyOwner {
}
```

Recommendation

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

MU - Modifiers Usage

Criticality	Minor / Informative
Location	contracts/GEMS.sol#L634,649,664
Status	Unresolved

Description

The contract is using repetitive statements on some methods to validate some preconditions. In Solidity, the form of preconditions is usually represented by the modifiers. Modifiers allow you to define a piece of code that can be reused across multiple functions within a contract. This can be particularly useful when you have several functions that require the same checks to be performed before executing the logic within the function.

```
require(
    _buyRewardFee + _buyLiquidityFee + _buyMarketingFee <=
maxFeeLimit,
    "Must keep fees below 30%"
)
require(
    _sellRewardFee + _sellLiquidityFee + _sellMarketingFee <=
maxFeeLimit,
    "Must keep fees <= 30%"
)
require(
    _transferRewardFee + _transferLiquidityFee +
_transferMarketingFee <= maxFeeLimit,
    "Must keep fees <= 30%"
);
```

Recommendation

The team is advised to use modifiers since it is a useful tool for reducing code duplication and improving the readability of smart contracts. By using modifiers to perform these checks, it reduces the amount of code that is needed to write, which can make the smart contract more efficient and easier to maintain.

PLPI - Potential Liquidity Provision Inadequacy

Criticality	Minor / Informative
Location	contracts/GEMS.sol#L920
Status	Unresolved

Description

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

```
function swapTokensForBNB(uint256 tokenAmount) private {
    address[] memory path = new address[](2);
    path[0] = address(this);
    path[1] = pancakeRouter.WETH();
    _approve(address(this), address(pancakeRouter), tokenAmount);
    pancakeRouter.swapExactTokensForETHSupportingFeeOnTransferTokens(
        tokenAmount,
        0, // accept any amount of ETH
        path,
        address(this),
        block.timestamp
    );
}
```

Recommendation

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

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

Additionally, the contract could be designed to tolerate potential reverts from the swap functionality, especially when it is a part of the main transfer flow. This can be achieved by executing the contract's token swaps in a non-reversible manner, thereby ensuring a more resilient and predictable operation.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	contracts/GEMS.sol#L778,865
Status	Unresolved

Description

The contract accumulates tokens from the taxes to swap them for ETH. The variable `minimumFeeTokensToTake` sets a threshold where the contract will trigger the swap functionality. If the variable is set to a big number, then the contract will swap a huge amount of tokens for ETH.

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

```
uint256 contractTokenBalance = balanceOf(address(this));
bool overMinimumTokenBalance = contractTokenBalance >=
    minimumFeeTokensToTake
// Take Fee
if (
    !inSwapAndLiquify &&
    overMinimumTokenBalance &&
    automatedMarketMakerPairs[to]
) {
    takeFee();
}
...
    function takeFee() private lockTheSwap {
        uint256 contractBalance = balanceOf(address(this));
        bool success;
        uint256 totalTokensTaken = _liquidityTokensToSwap
            .add(_marketingFeeTokensToSwap)
            .add(_rewardFeeTokens);
        if (totalTokensTaken == 0 || contractBalance < totalTokensTaken)
{
            return;
        }

        // Halve the amount of liquidity tokens
        uint256 tokensForLiquidity = _liquidityTokensToSwap / 2;
        uint256 initialBNBBalance = address(this).balance;

swapTokensForBNB(tokensForLiquidity.add(_marketingFeeTokensToSwap));
    ...
    }
```

Recommendation

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

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	contracts/GEMS.sol
Status	Unresolved

Description

SafeMath is a popular Solidity library that provides a set of functions for performing common arithmetic operations in a way that is resistant to integer overflows and underflows.

Starting with Solidity versions that are greater than or equal to 0.8.0, the arithmetic operations revert to underflow and overflow. As a result, the native functionality of the Solidity operations replaces the SafeMath library. Hence, the usage of the SafeMath library adds complexity, overhead and increases gas consumption unnecessarily in cases where the explanatory error message is not used.

```
library SafeMath {...}
```

Recommendation

The team is advised to remove the SafeMath library in cases where the revert error message is not used. Since the version of the contract is greater than `0.8.0` then the pure Solidity arithmetic operations produce the same result.

If the previous functionality is required, then the contract could exploit the `unchecked { ... }` statement.

Read more about the breaking change on https://docs.soliditylang.org/en/v0.8.16/080-breaking-changes.html#solidity-v0-8-0-breaking-changes.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	contracts/GEMS.sol#L608,616,621
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 excludeFromMaxTransaction(address updAds, bool isEx)
        public
        onlyOwner
    {
        isExcludedMaxTransactionAmount[updAds] = isEx;
        emit ExcludedMaxTransactionAmount(updAds, isEx);
    }

    function excludeFromFee(address account) external onlyOwner {
        isExcludedFromFee[account] = true;
        emit ExcludedFromFee(account, true);
    }

    function includeInFee(address account) external onlyOwner {
        isExcludedFromFee[account] = false;
        emit ExcludedFromFee(account, false);
    }
```

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.

RVU - Redundant Variable Usage

Criticality	Minor / Informative
Location	contracts/GEMS.sol#L538
Status	Unresolved

Description

The contract uses the `tradingActive` variable to indicate if the transfers are open to the public. The variable `_tradingActiveBlock` indicates the specific block where the transfers have opened. Both variables are configured once when the `enableTrading()` is called. As a result, the usage of the `tradingActive` variable is redundant since the `_tradingActiveBlock` will contain a value greater than zero after the execution of the `enableTrading` method.

```
function enableTrading() external onlyOwner {
  require(!tradingActive, "already enabled");
  tradingActive = true;
  _tradingActiveBlock = block.number;
  _transferDelayEnabled = true;
  emit TradingActivated();
}
```

Recommendation

The team is advised to remove the tradingActive variable and use the `_tradingActiveBlock` variable when it is required to check of the tradings have been enabled. This will decrease the gas consumption and optimize the codebase.

RC - Repetitive Calculations

Criticality	Minor / Informative
Location	contracts/GEMS.sol#L714
Status	Unresolved

Description

The contract contains methods with multiple occurrences of the same calculation being performed. The calculation is repeated without utilizing a variable to store its result, which leads to redundant code, hinders code readability, and increases gas consumption. Each repetition of the calculation requires computational resources and can impact the performance of the contract, especially if the calculation is resource-intensive.

```
if (
  tradingActive &&
  _protectBlockCount > block.number.sub(_tradingActiveBlock) &&
  !inSwapAndLiquify
) {
...
if (
  _protectBlockCount > block.number.sub(_tradingActiveBlock) &&
  (automatedMarketMakerPairs[from] ||
automatedMarketMakerPairs[to])
) {

...

uint256 _transferAmount =
amount.sub(_rewardFee).sub(_liquidityFee).sub(
    _marketingFee
);
super._transfer(from, to, _transferAmount);
uint256 _feeTotal =
_rewardFee.add(_liquidityFee).add(_marketingFee);
```

Recommendation

To address this finding and enhance the efficiency and maintainability of the contract, it is recommended to refactor the code by assigning the calculation result to a variable once

and then utilizing that variable throughout the method. By storing the calculation result in a variable, the contract eliminates the need for redundant calculations and optimizes code execution.

Refactoring the code to assign the calculation result to a variable has several benefits. It improves code readability by making the purpose and intent of the calculation explicit. It also reduces code redundancy, making the method more concise, easier to maintain, and gas effective. Additionally, by performing the calculation once and reusing the variable, the contract improves performance by avoiding unnecessary computations

ST - Stops Transactions

Criticality	Minor / Informative
Location	contracts/GEMS.sol#L753
Status	Unresolved

Description

The contract owner has the authority to stop the sales for all users excluding the addresses in the `isExcludedMaxTransactionAmount` mapping. The owner may take advantage of it by setting the `maxTransactionAmount` or `maxWallet` to a very small value. As a result, the users will not be able to buy and sell tokens.

```
//when buy
if (
    automatedMarketMakerPairs[from] &&
    !isExcludedMaxTransactionAmount[to]
) {
    require(
        amount <= maxTransactionAmount,
        "Buy transfer over max amount"
    );
    require(
        amount + balanceOf(to) <= maxWallet,
        "Cannot exceed max wallet"
    );
}
//when sell
else if (
    automatedMarketMakerPairs[to] &&
    !isExcludedMaxTransactionAmount[from]
) {
    require(
        amount <= maxTransactionAmount,
        "Sell transfer over max amount"
    );
}
```

Recommendation

The contract could embody a check for not allowing setting the `maxTransactionAmount` and `maxWallet` less than a reasonable amount. A suggested implementation could check that the minimum amount should be more than a fixed percentage of the total supply. 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.

UTE - Unchecked Transfer Emission

Criticality	Minor / Informative
Location	contracts/GEMS.sol#L899
Status	Unresolved

Description

The contract is designed to transfer funds to a specified `marketingFeeAddress` and subsequently emit a `MarketingFeeTaken` event to signal the completion of this transaction. This operation is executed through a low-level call function, which is intended to send the specified amount of BNB to the `marketingFeeAddress`. The code snippet provided attempts to perform this transfer using the `call` method, encapsulated within a success flag to indicate the transaction's outcome. However, the implementation does not include a check to verify whether the transfer was successful before emitting the `MarketingFeeTaken` event. Consequently, the event is emitted regardless of the actual transfer's success or failure, potentially leading to a misleading representation of the contract's state. This oversight can result in scenarios where the event suggests that marketing fees have been successfully taken and allocated, even in cases where the fund transfer might have failed due to reasons such as gas limitations, execution errors, or the recipient's inability to receive funds (e.g., a contract without a payable fallback function).

```
(success, ) = address(marketingFeeAddress).call{
    value: bnbForMarketing
}("");
emit MarketingFeeTaken(_marketingFeeTokensToSwap, bnbForMarketing);
```

Recommendation

It is recommended to implement a conditional check that evaluates the `success` of the funds transfer before emitting the `MarketingFeeTaken` event. This can be achieved by utilizing the `success` flag returned by the `call` function. Only if the call successfully transfers the funds, should the `MarketingFeeTaken` event be emitted. This approach ensures that the event accurately reflects the outcome of the transaction, enhancing the contract's reliability and trustworthiness. Additionally, in the case of a failed transfer,

appropriate error handling mechanisms should be employed, such as logging an error event, to provide clear feedback on the operation's failure. This modification not only improves the contract's accuracy in reporting state changes but also enhances its security by preventing potential discrepancies between the contract's intended and actual states.

US - Untrusted Source

Criticality	Minor / Informative
Location	contracts/GEMS.sol#L890
Status	Unresolved

Description

The contract uses an external contract in order to determine the transaction's flow. The external contract is untrusted. As a result, it may produce security issues and harm the transactions.

```
IStake stake=IStake(treasuryAddress);
_approve(address(this), address(stake), _rewardFeeTokens);
stake.depositReward(_rewardFeeTokens);
```

Recommendation

The contract should use a trusted external source. A trusted source could be either a commonly recognized or an audited contract. The pointing addresses should not be able to change after the initialization.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	contracts/GEMS.sol#L15,251,253,284,356,367,368,546,555,561,627,628,629,642,643,644,657,658,659,688,696,706
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.

```
function WETH() external pure returns (address);
function DOMAIN_SEPARATOR() external view returns (bytes32);
function PERMIT_TYPEHASH() external pure returns (bytes32);
function MINIMUM_LIQUIDITY() external pure returns (uint256);
function INIT_CODE_PAIR_HASH() external view returns (bytes32);
uint16 constant maxFeeLimit = 300
uint8 constant maxProtectBlockLimit = 10
uint256 _maxTransactionAmount
uint256 _maxWallet
uint256 _minimumFeeTokensToTake
uint16 _buyRewardFee
uint16 _buyLiquidityFee
uint16 _buyMarketingFee
uint16 _sellRewardFee

...
```

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/GEMS.sol#L574,599,605,676
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.

```
_protectBlockCount = protectBlockCount
_gasPriceLimit = gas * 1 gwei
_transferDelayEnabled = false;
_protectBlockRewardFee = protectBlockRewardFee
```

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/GEMS.sol#L482,483,484,689,700,710
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.

```
liquidityAddress = payable(_liquidityAddress)
marketingFeeAddress = payable(_marketingFeeAddress)
treasuryAddress = _treasuryAddress
```

Recommendation

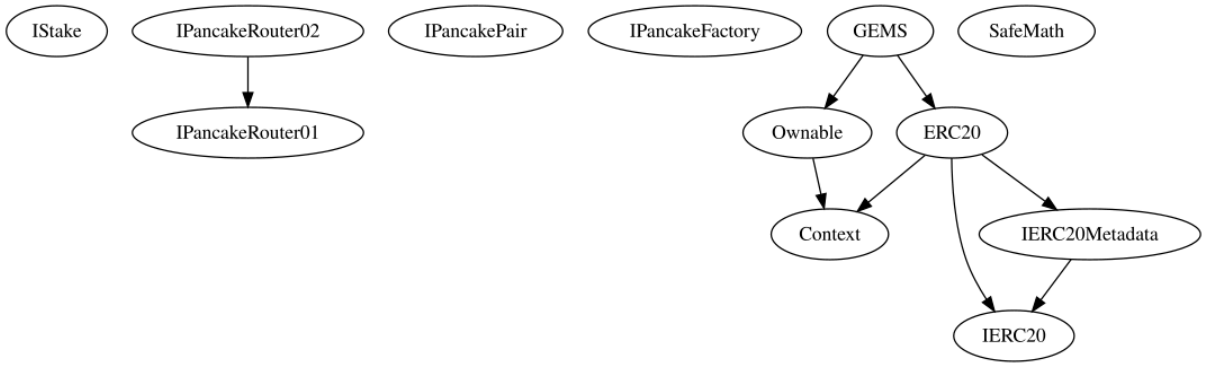
By adding the proper check, the contract will not allow the variables to be configured with zero value. This will ensure that the contract can handle all possible input values and avoid unexpected behavior or errors. Hence, it can help to prevent the contract from being exploited or operating unexpectedly.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IStake	Interface			
	depositReward	External	✓	-
GEMS	Implementation	ERC20, Ownable		
		Public	✓	ERC20
	decimals	Public		-
	enableTrading	External	✓	onlyOwner
	updateMaxTransactionAmount	External	✓	onlyOwner
	updateMaxWallet	External	✓	onlyOwner
	updateMinimumTokensBeforeFeeTaken	External	✓	onlyOwner
	updateProtectBlockCount	External	✓	onlyOwner
	setAutomatedMarketMakerPair	Public	✓	onlyOwner
	_setAutomatedMarketMakerPair	Private	✓	
	updateGasPriceLimit	External	✓	onlyOwner
	disableTransferDelay	External	✓	onlyOwner
	excludeFromMaxTransaction	Public	✓	onlyOwner
	excludeFromFee	External	✓	onlyOwner
	includeInFee	External	✓	onlyOwner
	updateBuyFee	External	✓	onlyOwner

	updateSellFee	External	✓	onlyOwner
	updateTransferFee	External	✓	onlyOwner
	setProtectBlockFee	External	✓	onlyOwner
	updateTreasuryAddress	External	✓	onlyOwner
	updateMarketingFeeAddress	External	✓	onlyOwner
	updateLiquidityAddress	External	✓	onlyOwner
	_transfer	Internal	✓	
	removeAllFee	Private	✓	
	restoreAllFee	Private	✓	
	takeFee	Private	✓	lockTheSwap
	swapTokensForBNB	Private	✓	
	addLiquidity	Private	✓	
		External	Payable	-

Inheritance Graph



Flow Graph

Summary

Gempad is a token, aiming to provide a platform for various financial activities, such as trading and staking, while ensuring transparency and security through smart contract implementation. The project incorporates three distinct types of fees: liquidity fees, marketing fees, and staking fees. These fees serve different purposes within the ecosystem, including facilitating liquidity provision, supporting marketing efforts, and enabling staking rewards.

As a smart contract auditing firm, our assessment of Gempad's codebase will focus on verifying the integrity, efficiency, and security of the smart contracts governing fee distribution, as well as ensuring compliance with the specified fee structure and fee cap. Our audit will involve thorough code review, testing, and analysis to identify and address any potential vulnerabilities, bugs, or issues that may compromise the functionality or security of the platform.

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