Disclaimer

SolidGroup reports are not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team. Solid group do not cover testing or auditing the integration with external contract or services (such as Unicrypt, Uniswap, PancakeSwap etc’…)
SolidGroup Audits do 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. SolidGroup Audits should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.**
SolidGroup Reports represent an extensive auditing 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. **SolidGroup’s position is that each company and individual are responsible for their own due diligence and continuous security.** SolidGroup in no way claims any guarantee of security or functionality of the technology we agree to analyze.

BEP-20's Conformance

This test checks for BEP-20's conformance.

- All the functions are present
- All the events are present
- Functions return the correct type
- Functions that must be view are view
- Events' parameters are correctly indexed
- The functions emit the events
- Derived contracts do not break the conformance

Function	present	type	Correct Return value	events	
totalSupply	✅	✅ view	✅		
balanceOf(address)	✅	✅ view	✅		
transfer(address,uint256)	✅	✅ external	✅	✅ Transfer	
transferFrom(address, address, uint256)	✅	✅ external	✅	✅ Transfer	
approve(address,uint256)	✅	✅ external	✅	✅ Approval	
allowance(address, address)	✅	✅ view	✅		
name	✅	✅ view	✅		
symbol	✅	✅ view	✅		

Check Events:

✅ Transfer

✅ Approve

The contract that was tested is the token's contract: **RegimentToken**.sol

Findings

RegimentToken.sol

⚠ Make sure that the team set the MasterChef contract as the owner.

Issue #1:

Type	Severity	Location	Fixed
Gas Optimization	● Informational	RegimentToken.sol	❌

Description:

transferFrom often fails if users do not call approve(), and this implementation will use as much gas as possible in those failed cases.

```
    function transferFrom(address sender, address recipient, uint256 amount) public virtual
override returns (bool) {
        _transfer(sender, recipient, amount);
        _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, "ERC20:
transfer amount exceeds allowance"));
        return true;
    }
```

Recommendation:

Functions should fail ASAP to use minimal gas. Therefore we recommend swapping those lines.

```
    function transferFrom(address sender, address recipient, uint256 amount) public virtual
override returns (bool) {
        _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, "ERC20:
transfer amount exceeds allowance"));
        _transfer(sender, recipient, amount);
        return true;
    }
```

Issue #2:

Type	Severity	Location	Fixed
Gas Optimization	● Informational	RegimentToken.sol	✅

Description:

The public **burnRate** should be declared as external.

Recommendation:

This function is only called from outside of the contract, consider using the external attribute instead of public to save gas.

Summary

● Informational severity Issues	1
● Low severity issues	0
● Medium severity issues	0
● High severity issues	0

MasterChef.sol

Issue #1:

Type	Severity	Location	Fixed
Gas Optimization	● Informational	MasterChef.sol	✅

Description:

The public **add**, **set**, **deposit**, **withdraw**, **emergencyWithdraw**, **dev, setFeeAddress** and **updateEmissionRate** functions should be declared as external.

Recommendation:

These functions are only called from outside of the contract, consider using the external attribute instead of public to save gas.

Issue #2:

Type	Severity	Location	Fixed
Best Practice	● Informational	MasterChef.sol	✅

Description:

Event is missing in updateEmissionRate function.

Recommendation:

Our recommendation is to add events in critical parts of the contract, such us when updating emission rate.
. Events are great for integrating with DApps in the future. We recommend considering emitting events when state is changed.
For Example:

```
    function updateEmissionRate(uint256 _rtfPerBlock) public onlyOwner {
        massUpdatePools();
        rtfPerBlock = _rtfPerBlock;
        emit UpdateEmissionRate(msg.sender, _rtfPerBlock);
    }
```

Issue #3:

Type	Severity	Location	Fixed
Best Practice	● Informational	MasterChef.sol	✅

Description:

Events are missing in dev and setFeeAddress functions.

Recommendation:

Our recommendation is to add events in critical parts of the staking contract, such as the dev address is changed, or when the fee address is changed.
. Events are great for integrating with DApps in the future. We recommend considering emitting events when state is changed.
For Example

```
    function dev(address _devaddr) public {
        require(msg.sender == devaddr, "dev: wut?");
        devaddr = _devaddr;
        emit SetDev(_devaddr);
    }
```

```
    function setFeeAddress(address _feeAddress) public{
        require(msg.sender == feeAddress, "setFeeAddress: FORBIDDEN");
        feeAddress = _feeAddress;
        emit SetFeeAddress(_feeAddress);
    }
```

Issue #3:

Type	Severity	Location	Fixed
Re-entrancing	● Medium	MasterChef.sol	✅

Description:

Due to unknown implementation of contract _lpToken, the implementation of function safeTransfer() is also unknown and may have malicious logical implementation that calls back to the function deposit(), which can lead to another invocation of safeTransfer() without updating user.amount. This is dangerous to the user.amount and will incorrectly calculate the user's balance eventually.

Recommendation:

We advise developers to swap the pool.lpToken.safeTransfer(feeAddress, depositFee); and user.amount = user.amount.add(_amount).sub(depositFee); to follow checks-effects interaction pattern

Issue #4:

Type	Severity	Location	Fixed
Re-entrancing	● Medium	MasterChef.sol	✅

Description:

Due to unknown implementation of contract _lpToken, the implementation of function safeTransfer() is also unknown and may have malicious logical.

Recommendation:

Add re-entrancing guard for emergencyWithdraw, deposit, and withdraw functions.
For example:

```
function deposit(uint256 _pid, uint256 _amount) public nonReentrant{
      PoolInfo storage pool = poolInfo[_pid];
      UserInfo storage user = userInfo[_pid][msg.sender];
      updatePool(_pid);
      if (user.amount > 0) {
          uint256 pending = user.amount.mul(pool.accRegPerShare).div(1e12).sub(user.rewardDebt);
          if(pending > 0) {
              safeRtfTransfer(msg.sender, pending);
          }
      }
      if(_amount > 0) {
          pool.lpToken.safeTransferFrom(address(msg.sender), address(this), _amount);
          if(pool.depositFeeBP > 0){
              uint256 depositFee = _amount.mul(pool.depositFeeBP).div(10000);
              pool.lpToken.safeTransfer(feeAddress, depositFee);
              user.amount = user.amount.add(_amount).sub(depositFee);
          }else{
              user.amount = user.amount.add(_amount);
          }
      }
      user.rewardDebt = user.amount.mul(pool.accRegPerShare).div(1e12);
      emit Deposit(msg.sender, _pid, _amount);
  }
```

Issue #5:

Type	Severity	Location	Fixed
Volatile Code	Informational	MasterChef.sol	✅

Description:

Return value of function transfer() are ignored in function safeRtfTransfer()

Recommendation:

Our recommendation it to handle the return value of transfer () to check if transfer is executed without any error.

Summary

Informational severity Issues	0
Low severity issues	0
Medium severity issues	0
High severity issues	0