SOLID GROUP

X

Security Assessment

June 17th , 2021

For

Lucky Project

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 technology 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

Solid Group 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.

Solid Group hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against Solid Group, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption)

Solid Group’s position is that each company and individual are responsible for their own due diligence and continuous security.

Solid Group LTD in no way claims any guarantee of security or functionality of the technology we agree to analyze.

Files In Scope

Commit:

First Revision
https://raw.githubusercontent.com/luckyBSC/LuckyBSC/c6a1948e36c960f733213a1606b05623ffd4b9ab/Lucky.sol

Final Revision:

https://github.com/luckyBSC/LuckyBSC/blob/7991a332e77a2aba087d86473e27887e4f63b14c/Lucky.sol

Constructor

- totalSupply = 888888888888888 * 10**9 (Can't be changed by the owner)
- 0% < _taxFee < 10% (Can be changed by the owner)
- 0% < liquidityFee < 10% (Can be changed by the owner)
- uint256 luckyDrawPrize = 88 * 10**18 (= 88 BUSD)
- uint256 jackpotPrize = 88888 * 10**18 (= 88888 BUSD)

First 30 minutes there is buy/sell limitation of `600 * 10**9` **tokens in one transaction.**

Privilege Functions

- **`setMinimumLotteryBalance` - The owner can set the minimum balance needed to be eligible for the lottery prize.**
- **`excludeFromReward`**
- **`includeInReward`**
- **`excludeFromFee`**
- **`includeInFee`**
- **`setTaxFeePercent`**
- **`setLiquidityFeePercent`**
- **`setMaxTxPercent`**
- **`setSwapAndLiquifyEnabled`**

General notes

⚠️ The presale is managed by the owner of the team, our recommendation is to manage the presale using a third party service to increase investors' confidence.

First Revision Findings

Issue #1 | Logical Issue | LOW | hasLaunched | Status: FIXED

Description

This function should only be executed once. The owner can call this function more than once and reactivate launch limitations (For example limit the number of tokens that can be sold/bought).

Recommendation

This function should only be executed once. Add a require statement to enforce that:

```
function hasLaunched() public onlyOwner {
    require(launchBlock == 0);
    isLive = true;
    launchBlock = block.number;
}
```

Issue #2 | Logical Issue | Medium | _transfer | Status: FIXED

Description

```
if (presale[from].additionalTokens < amount) {
    presale[from].additionalTokens = 0;
    uint256 moveRem = amount.sub(presale[from].additionalTokens);...}

uint256 moveRem = amount.sub(presale[from].additionalTokens)
```

subtraction by zero has no effect.

Issue #3 | Best Practice | High | _transfer | Status: Fixed

Description

_transfer should always work, even if there is a bug in the contract, to ensure that investors' funds are safe. If the function is **critical (such as _transfer) always make sure its error cases are handled gracefully!**

_transfer calls swapAndLiquify, sendBUSD , and lotteryCheck which could fail.

Recommendation

Use try-catch.

Issue #4 | Logical Issue | High | _transfer | Status: Fixed

Description

```
if (presale[from].additionalTokens < amount) {
        presale[from].additionalTokens = 0;
        uint256 moveRem = amount.sub(presale[from].additionalTokens);
        require(moveRem <= getAmountMovable(from), "exceeds presale limit");
        presale[from].amountMoved = presale[from].amountMoved.add(moveRem);
```

The way you handle presalers is wrong. Consider the following scenario: Investor A purchased 100 tokens on presale. On listing, he bought 40 tokens. amount of tokens that are supposed to be unlocked for him are 100 * 10% + 40 = 50 tokens. Now he is trying to sell 50 tokens (he should be able to do that because 50 tokens are unlocked).

The condition, `presale[from].additionalTokens < amount` is true.

```
presale[from].additionalTokens = 0;
```

```
uint256 moveRem = 50
```

However, `getAmountMovable(from)` will return 10, and the transaction will be reverted.

Recommendation

Switch between the lines 2,3.

Our more general recommendation is to change the way you handle presalers. instead of tracking the amount of tokens presaler purchased on listing you should only track the number of unlocked tokens which is the amount the investor bought on listing and the amount of tokens that are released according to vesting.

Issue #5 | Logical Issue | Informational | getAmountMovable | Status: Fixed

Description

Use of magic numbers.

```
if (presale[user].dayStartBlock.add(28800) < block.number) {
```

Recommendation

Magic numbers should be avoided. Change it to a constant variable describing the purpose of this value.

Issue #6 | Logical Issue | Low | getAmountMovable | Status: Fixed

Description

Usually, it's customary to work with time instead of block.number, when calculating the number of tokens that should be unlocked after a certain period.

Issue #7 | Logical Issue | High | getAmountMovable | Status: Fixed

Description

The calculation includes the vesting time since the first block, but the first day of vesting (28800 blocks) was already rewarded. this means the user will get rewarded twice on the first day of vesting.

Recommendation

Exclude the first day from blocksSincePresale.

```
uint256 blocksSincePresale = block.number.sub(presale[user].dayStartBlock) - 28800;
```

Issue #8 | Owner Capabilities | High | setTaxFeePercent | Status: Fixed

`setLiquidityFeePercent`

Description

The owner of the contract can set the tax fee and liquidity fee by calling setTaxFeePercent or setLiquidityFeePercent to any value he desires. If the owner sets the fee to 100% the token will be untradeable.

Recommendation

Add low and high boundaries for the setter functions.

```
function setTaxFeePercent(uint256 taxFee) external onlyOwner() {
    _taxFee = taxFee;
}
```

```
    function setLiquidityFeePercent(uint256 liquidityFee) external
onlyOwner() {
        _liquidityFee = liquidityFee;
    }
```

Issue #9 | Owner Capabilities | High | setMaxTxPercent | Status: Fixed

Description

The owner of the contract can set _maxTxAmount to zero and make the token untradeable.

```
  function setMaxTxPercent(uint256 maxTxPercent) external onlyOwner() {
    _maxTxAmount = _tTotal.mul(maxTxPercent).div(
      10**2
    );
  }
```

in _transfer:

```
if(from != owner() && to != owner())
        require(amount <= _maxTxAmount, "Transfer amount exceeds the maxTxAmount.");
```

The condition will always be false.

```
require(amount <= _maxTxAmount, "Transfer amount exceeds the maxTxAmount.");
```

Recommendation

```
 function setMaxTxPercent(uint256 maxTxPercent) external onlyOwner() {
     require(maxTxPercent!=0, "can't set....");
     _maxTxAmount = _tTotal.mul(maxTxPercent).div(
       10**2
     );
  }
```

Issue #10 | Logical Issue | Low | _rebalanceTickets | Status : Fixed

Description

Judging by the comment it seems like the condition is wrong.

```
   //remove user from list if minimum tickets not met
  if (balanceOf(user) == minimumLotteryBalance) {
```

Recommendation

Change the condition

```
//remove user from list if minimum tickets not met
if (balanceOf(user) <= minimumLotteryBalance) {
```

Issue #11 | LOGICAL ISSUE | LOW | Lottery Code

Description

The pair is not excluded from the lottery.

Recommendation

We recommend excluding the pair from lottery,

Issue #12 | Logical Issue | Informational | lotteryCheck | Status: Fixed

Description

Should be 1500$, not 150$.

if (luckyDrawAmount >= 150000000000000000000)

Issue #13 | **Logical Issue\Owner Capabilities** | High | **unlock , lock | Status:** Fixed

Description

An owner can gain ownership of the contract even if he calls renounceOwnership function.

This can be achieved by performing the following steps:

1. The owner of the contract can call lock() to lock the contract (the lock function saves the previous owner into a variable)
2. After the locking period has passed the owner of the contract can call unlock() and regain the ownership.
3. The owner of the contract can then call the renounceOwnership function. Now, the contract allegedly has no owner (users can verify it by looking for the renounceOwnership transaction and making sure that the owner is set to the zero address)
4. The owner of the contract can call the unlock function again, and get the ownership back.

Recommendation:

Our recommendation is to remove the lock and unlock function if not needed

FIXED **Issue #14** | LOGICAL ISSUE | MEDIUM | includeInReward

Descriptio:
The code is vulnerable to the SafeMoon bug (described here) - excluding an address from fee and then later including it back will cause the address to receive all RFI rewards for the time it was excluded (at the expense of other holders)

Recommendation:
Properly calculate _rOwned of the included address in includeInReward() based on its _tOwned amount

Issue #15 | **Best Practice** | Informational | **Status:** Fixed

Use readable prefixes like ether, gwei, etc or powers of 10 (e.g 1e8).

Issue #16 | Best Practice | Informational | lotteryCheck | Status: Fixed

8880000000000000000 move this magic number into a const, especially if it is used so much.

FIXED Issue #17 | BEST PRACTICE | INFORMATIONAL | lotteryCheck

Lack of events in the contract. Should emit even when there is a lottery winner.

Issue #18 | Logical Issue | High | getRandomNumber | Status: Fixed

Description

By using % topUserId, the top user can never win, and id 0 may sometimes win (which is not a valid user id).

Recommendation

Fix this issue by adding 1 to the final random result.

Issue #19 | Gas Optimization | Informational | _rebalanceTickets | Status: Fixed

Description

Can get gas refund by setting idAddress[topUserId] = 0

Issue #20 | Logical Issue | High | jackpotCheck, lotteryCheck | Status: Fixed

These functions should be declared as internal since the user can spam until he wins the rewards.

Issue #21 | Logical Issue | Low | _setUserID | Status: Fixed

Description

Should by = not ==.

```
userId[user] == topUserId.add(1);
```

Issue #22 | Owner Capabilities | High | setTaxFeePercent, setLiquidityFeePercent| Status: Fixed

Description

The owner of the contract can make the token untradable. by calling setTaxFeePercent(24) setLiquidityFeePercent(24). Total slippage will be around 44%, which will make the token untradable due to high slippage. (Pancakeswap won't get the transaction go through).

Recommendation

Lower the maximum set limit for both setter functions.

NOT FIXED Issue #2 | LOGICAL ISSUE/GAS OPTIMIZATION | LOW | lotteryCheck

Description

If there was a winner in lotteryCheck the call for jackpotCheck will fail immediately and it's redundant.

Recommendation

call jackpotCheck only if there wasn't a winner in lotteryCheck.

```
if (winnerValid) {
        if (IERC20(BUSD).balanceOf(address(this)) < luckyDrawPrize) {
            return;
        }
        previousWinner = winningUser;
        previousWinningBlock = block.number;
        IERC20(BUSD).transfer(idAddress[winningUser], luckyDrawPrize);
        _winningUsers.push(idAddress[winningUser]);
        _winningAmount.push(luckyDrawPrize);
        jackpotAmount = IERC20(BUSD).balanceOf(address(this));
        luckyDrawAmount = 0;
        trysUntilDraw.push(currentTrys);
        currentTrys = 0;
        emit WinnerSelected(idAddress[winningUser], luckyDrawPrize);
    }
else{
    //are jackpot requirements met?
    jackpotCheck();
 }
```

Issue #23 | Logical Issue/Gas Optimization | Informational | jackpotCheck | Status: Fixed

Description

The winner win jackpotAmount and not jackpotPrize.

```
emit WinnerSelected(idAddress[winningUser], jackpotPrize);
```

Recommendation

Change the event parameters.

```
emit WinnerSelected(idAddress[winningUser], jackpotAmount);
```

Issue #24 | Logical Issue/Gas Optimization | Informational | _transfer | Status: Fixed

Description

The pair and the contract are not eligible for winning the rewards. Therefore, no need to call _rebalanceTickets if the from/to address is the pair or the contract.

Recommendation

```
 if (from != pairAddress || from != address(this)){
 _rebalanceTickets(from);
 }
 if (to != pairAddress || to != address(this)){
 _rebalanceTickets(to);
 }
```

Issue #25 | Logical Issue/Gas Optimization | Informational | _transfer | Status: Fixed

Description

lotteryCheck shouldn't be called during swapAndLiquify.

Recommendation

```
if (!inSwapAndLiquify){
      lotteryCheck();
}
```

Issue #6 | Logical Issue | High | getAmountMovable | Status: Fixed

Description

On line 1267 dayStartTime was initialized using block.timestamp, but it is compared to block.number.

Recommendation

Compare it to block.timestamp as well.