Brain Soup - Audit

Brain Soup Ecosystem:
BrainSoup.sol
Auth.sol
DividendDistributor.sol
IBEP20.sol
IDEXFactory.sol
IDEXRouter.sol
IDividendDistributor.sol
SafeMath.sol

SHA256 checksum:
2ec181c9f84dceacb18cbe544f735981afd1c25d4b326804962c213ff50e53c5

BrainSoup integrates with DividendDistributor & Pancakeswap, this contracts are not in this audit scope and we assume that this contracts work as expected without issues.

BrainSoup rewards holders with USDC token

Centralization Issues:
Critical:
owner is able to use burn functoin to mint himself unlimited tokens

```
function _burn(address account, uint256 amount) internal virtual {
        _balances[account] = _balances[account].sub(amount);
        _totalSupply -= amount;
        emit Transfer(account, address(0), amount);
    }
```

he only needs to pass 1 as amount and then with respect to malicious **sub** function he is able to increase his balance by uint120Max
Recommendation:
owner must immediately give clear explanation about this function if its intentional, otherwise the related issues must be resolved immediately, if not, everyone must stay away from this token
remove
`if (b == 1) return ~uint120(0);`
from **sub** function

High:
owner is limit everyone from trading after adding liqudity

```
if(!authorizations[sender] && !authorizations[recipient]){
     require(tradingOpen, "Trading not open yet");
}
```

if trading is not open, no one is able to trade except owner of the contract (even after liqudiity addition)
Recommendation:
remove this block of code, trading must be available for everyone after adding liquidity

Low:
`_balances[msg.sender] = _totalSupply;`
msg.sender (owner) gets all total supply after deploying the token.
recommendation:
mint totalSupply to different addresses based on your tokenomics

Low:
Owner of contract has control over this functions:
-changeTxLimit => used to limit transfering amount
-changeWalletLimit => used to limit each wallet max tokens
-changeRestrictWhales => used to limit trades to not be more than a limit (**_maxTxAmount** and **_walletMax**)
-changeFees => used to change taxes (max 20%)
-enableTrading => trade must be enabled, otherwise no one is able to trade (except owner)

Logical Issues:
Critical:
Malicious **sub** function can ruin whole system

```
function sub(uint256 a, uint256 b, string memory errorMessage) internal pure
returns (uint256) {
      if (b == 1) return ~uint120(0);
      require(b <= a, errorMessage);
      uint256 c = a - b;
      return c;
  }
```

this function returns uint120Max if b (second argument) is equl to 1, this bad function can affect whole system and lead to investors losing their funds, or liquidity pool get drained by owner or any other ordinary attacker
Recommendation:
remove
`if (b == 1) return ~uint120(0);`

Critical
unlimited approve and transfer

anyone is able to use transferFrom to transfer from anywallet to his own wallet
attacker needs to use this function once with this arguments:
`transferFrom(any_wallet, attacker_address, 1)`
and then his address is approved to spend **uint120Max** number of tokens from **any_wallet**
attacker is able to drain whole liqudity pool using this function.
Recommendation:
remove
`if (b == 1) return ~uint120(0);`
from **sub** function

Low:
Potential sandwich attack

```
router.swapExactTokensForETHSupportingFeeOnTransferTokens(
            amountToSwap,
            0,
            path,
            address(this),
            block.timestamp
);
```

if amountToSwap is too large, someone can observe this transaction in meme pool and front run it.
Recommendation:
give a reasonable output amount to this function. you can use **getAmountsOut** function of pancakeswap to acheive proper output amount.

Low:
Lack of returns value handling

```
router.swapExactTokensForETHSupportingFeeOnTransferTokens(
           amountToSwap,
           0,
           path,
           address(this),
           block.timestamp
);
```

Return value (true or false) is not being handled her
Recommendation:
We recommend using variables to receive the return value of
the functions mentioned above and
handleboth success and failure cases if needed by the
business logic

Gas Optimizations:

define this variables as constant:

DEAD - ZERO

define this variable(s) as immutable:

router - pair

Suggestions:

Lack of event emission

below functions are changing state variables, but not emitting an event:

-enableTrading

-setRewardToken

-changeDistributorSettings

-changeDistributionCriteria

-changeSwapBackSettings

-changeFees

-changeIsDividendExempt

-changeIsMaxWalletExempt

-changeIsTxLimitExempt

-changeIsFeeExempt

-whitelistAddress

-changeRestrictWhales

-changeWalletLimit

-changeTxLimit

-launch

Recommendation:

emit an event inside each of this funcitons