BLOCKHAT SECURITY

Smart Staking

Smart Contract Security Audit

Prepared by BlockHat

October 20th, 2023 - September 23rd, 2023

BlockHat.io

contact@blockhat.io

Document Properties

Client	$MART
Version	1.0
Classification	Public

Scope

Link	Address
https://bscscan.com/address/0xb2c4a53CAC0C58559127a7525a7C55DC98431F52#code	0xb2c4a53CAC0C58559127a7525a7C55DC98431F52

Contacts

COMPANY	CONTACT
BlockHat	contact@blockhat.io

Contents

1 Introduction

$MART engaged BlockHat to conduct a security assessment on the Smart Staking beginning on October 20th, 2023 and ending September 23rd, 2023. In this report, we detail our methodical approach to evaluate potential security issues associated with the implementation of smart contracts, by exposing possible semantic discrepancies between the smart contract code and design document, and by recommending additional ideas to optimize the existing code. Our findings indicate that the current version of smart contracts can still be enhanced further due to the presence of many security and performance concerns.

This document summarizes the findings of our audit.

1.1 About Smart Staking

Core Features and Vision Transactional Dynamics of the $MART Token Every interaction with the $MART token is crafted to cultivate an ecosystem of sustainability and rewards. Each buy or sell transaction is accompanied by a 10% fee, strategically distributed to foster growth, reward holders, and ensure longevity:

- Redistribution in BUSD : 4%
- Towards Project Development: 2%
- For Marketing: 2%
- Liquidity Addition: 1%
- Token Burn 1%
- symbol: $MART

Issuer	$MART
Website	`https://smartstaking.io/`
Type	Solidity Smart Contract
Audit Method	Whitebox

1.2 Approach & Methodology

BlockHat used a combination of manual and automated security testing to achieve a balance between efficiency, timeliness, practicability, and correctness within the audit's scope. While manual testing is advised for identifying problems in logic, procedure, and implementation, automated testing techniques help to expand the coverage of smart contracts and can quickly detect code that does not comply with security best practices.

1.2.1 Risk Methodology

Vulnerabilities or bugs identified by BlockHat are ranked using a risk assessment technique that considers both the LIKELIHOOD and IMPACT of a security incident. This framework is effective at conveying the features and consequences of technological vulnerabilities.

Its quantitative paradigm enables repeatable and precise measurement, while also revealing the underlying susceptibility characteristics that were used to calculate the Risk scores. A risk level will be assigned to each vulnerability on a scale of 5 to 1, with 5 indicating the greatest possibility or impact.

- Likelihood quantifies the probability of a certain vulnerability being discovered and exploited in the untamed.
- Impact quantifies the technical and economic costs of a successful attack.
- Severity indicates the risk's overall criticality.

Probability and impact are classified into three categories: H, M, and L, which correspond to high, medium, and low, respectively. Severity is determined by probability and impact and is categorized into four levels, namely Critical, High, Medium, and Low.

Impact	High	Critical	High	Medium
	Medium	High	Medium	Low
	Low	Medium	Low	Low
		High	Medium	Low
			Likelihood	

2 Findings Overview

2.1 Summary

The following is a synopsis of our conclusions from our analysis of the Smart Staking implementation. During the first part of our audit, we examine the smart contract source code and run the codebase via a static code analyzer. The objective here is to find known coding problems statically and then manually check (reject or confirm) issues highlighted by the tool. Additionally, we check business logics, system processes, and DeFi-related components manually to identify potential hazards and/or defects.

2.2 Key Findings

In general, these smart contracts are well-designed and constructed, but their implementation might be improved by addressing the discovered flaws, which include 1 critical-severity, 4 high-severity, 6 medium-severity, 2 low-severity, 1 informational-severity vulnerabilities.

Vulnerabilities	Severity	Status
Division By Zero	CRITICAL	Fixed
Incorrect Token Distribution Logic in swapAndSendToFee Function	HIGH	Fixed
Centralized risk in addLiquidity	HIGH	Fixed
Non-Withdrawable BNB: swapAndLiquify function	HIGH	Fixed
Missing Value Check for swapTokensAtAmount	HIGH	Fixed
Missing Zero Address Check	MEDIUM	Fixed
Missing Value Checks	MEDIUM	Fixed
Use of .transfer instead of call	MEDIUM	Fixed
Centralization Risk :blacklistAddress Function	MEDIUM	Fixed
Missing Balance Check Before Dividend Withdrawal	MEDIUM	Acknowledged
Potential Sandwich Attacks	MEDIUM	Acknowledged

Inefficient Use of success Boolean and Redundant State Modifications in _withdrawDividendOfUser Function	LOW	Fixed
Misleading Function Name setBUSDRewardsFee	LOW	Fixed
Blocking Transfers	INFORMATIONAL	Fixed

3 Finding Details

A smartstaking.sol

A.1 Division By Zero [CRITICAL]

Description:

If totalFees is zero, the calculations within the function, specifically contractTokenBalance.mul(marketingFee).div(totalFees) and the division operations in swapAndSendToFee, will throw due to a division-by-zero error. This will result in the failure of all transfers, essentially freezing all token operations.

Code:

Listing 1: smartstaking.sol

```
    function setBUSDRewardsFee(uint256 _rewardFee, uint256 _liquidityFee
        ↪ , uint256 _marketingFee, uint256 _devFee, uint256 _burnFee)
        ↪ external onlyOwner{
        BUSDRewardsFee = _rewardFee;
        liquidityFee = _liquidityFee;
        marketingFee = _marketingFee;
        devFee = _devFee;
        burnFee = _burnFee;
        totalFees = BUSDRewardsFee.add(liquidityFee).add(marketingFee).
            ↪ add(devFee).add(burnFee);

        require(totalFees <= 20, "Fees Must be 20% Or less");
    }
```

Risk Level:

Likelihood – 4
Impact - 4

Listing 2: smartstaking.sol

```
    function swapAndSendToFee(uint256 tokens) private {

        uint256 initialBalance = address(this).balance;

        swapTokensForEth(tokens);
        uint256 newBalance = address(this).balance.sub(initialBalance);
        uint256 marketingAmount = newBalance.div(totalFees.mul(
            ↪ marketingFee));
        uint256 devWalletAmount = newBalance.div(totalFees.sub(
            ↪ marketingFee).mul(devFee));
        if(marketingAmount > 0) {
            _marketingWalletAddress.transfer(marketingAmount);
        }

        if(devWalletAmount > 0) {
            _devWalletAddress.transfer(devWalletAmount);
        }


    }
```

Recommendation:

Implement a condition to check if totalFees is greater than zero.

Status - Fixed

A.2 Incorrect Token Distribution Logic in swapAndSendToFee Function [HIGH]

Description:

The token distribution logic for various fees, including marketing, buyback, and liquidity fees. The primary function _transfer determines how many tokens should be allocated for each fee category based on the contractTokenBalance and the specific fee percentages. The tokens are then processed via the swapAndSendToFee and swapAndLiquify functions. The core vulnerability lies within the swapAndSendToFee function, which seems to miscalculate the distribution of Ether (obtained by swapping the tokens) between the marketing and development (dev) wallets. The logic employed to determine the amount of Ether to send to the marketing and dev wallets does not align with the way the tokens are initially distributed for these fees.

Code:

Listing 3: smartstaking.sol

```
    function swapAndSendToFee(uint256 tokens) private {

        uint256 initialBalance = address(this).balance;

        swapTokensForEth(tokens);
        uint256 newBalance = address(this).balance.sub(initialBalance);
        uint256 marketingAmount = newBalance.div(totalFees.mul(
            ↪ marketingFee));
        uint256 devWalletAmount = newBalance.div(totalFees.sub(
            ↪ marketingFee).mul(devFee));
        if(marketingAmount > 0) {
            _marketingWalletAddress.transfer(marketingAmount);
        }
        if(devWalletAmount > 0) {
            _devWalletAddress.transfer(devWalletAmount);
        }



    }
```

Risk Level:

Likelihood – 4
Impact - 4

Recommendation:

Redefine the Ether distribution logic in swapAndSendToFee to directly correspond with the token distribution proportions. A clearer method would be to use the token proportions directly to distribute the Ether.

Status - Fixed

A.3 Centralized risk in addLiquidity [HIGH]

Description:

The addLiquidity function calls the uniswapV2Router.addLiquidityETH function with the to address specified as owner() for acquiring the generated LP tokens from the pool. As a result, over time the _owner address will accumulate a significant portion of LP tokens. If the _owner is an EOA (Externally Owned Account), mishandling of its private key can have devastating consequences to the project as a whole

Code:

Listing 4: smartstaking.sol

```
        function addLiquidity(uint256 tokenAmount, uint256 ethAmount)
            ↪ private {
```

```
        // approve token transfer to cover all possible scenarios
        _approve(address(this), address(uniswapV2Router), tokenAmount);

        // add the liquidity
        uniswapV2Router.addLiquidityETH{value: ethAmount}(
            address(this),
            tokenAmount,
            0, // slippage is unavoidable
            0, // slippage is unavoidable
            owner(),
            block.timestamp
        );

    }
```

Risk Level:

Likelihood – 2
Impact - 4

Recommendation:

We recommend updating the uniswapV2Router.addLiquidityETH function to replace its address with the contract's address, using address(this). This modification ensures that LP tokens are managed within the contract's logic, providing an added layer of security against theft in case the _owner account gets compromised.

For broader security enhancements, it's crucial to strengthen centralized privileges or roles in the protocol. This can be achieved through decentralized mechanisms or by utilizing smart-contract based accounts that adhere to advanced security practices, such as multisignature wallets.

To further bolster security and mitigate potential risks, consider the following solutions:

1. Implementing a time-lock mechanism with a reasonable latency, such as 48 hours, to provide awareness of any privileged operations.

2. Assigning critical roles to multisignature wallets, which prevents vulnerabilities associated with a single private key compromise.

3. Introducing modules like DAO, governance, or voting to enhance transparency and foster active participation from users.

Status - Fixed

A.4 Non-Withdrawable BNB: swapAndLiquify function [HIGH]

Description:

The swapAndLiquify function converts half of the contractTokenBalance tokens to BNB. The other half of tokens and part of the converted BNB are deposited into the pool on pancakeswap as liquidity. For every swapAndLiquify function call, a smal amount of BNB leftover in the contract. This is because the price of token drops after swapping the first half of tokens into BNBs, and the other half of tokens require less than the converted BNB to be paired with it when adding liquidity. The contract doesn't appear to provide a way to withdraw those BNB, and they wil be locked in the contract forever

Risk Level:

Likelihood – 2
Impact - 4

Recommendation:

It's not ideal that more and more BNB are locked into the contract over time. The simplest solution is to add a function in the contract to withdraw BNB. Other approaches that benefit the token holders can be:

- Distribute BNB to token holders proportional to the amount of token they hold.

- Use leftover BNB to buy back tokens from the market to increase the price of token

Status - Fixed

A.5 Missing Value Check for swapTokensAtAmount [HIGH]

Description:

The function swaptokenchange allows the contract owner to change the swapTokensAtAmount variable without any checks for validity or bounds. Depending on what swapTokensAtAmount is used for in the contract, this could potentially be a significant issue, especially if incorrect or malicious values could disrupt the contract's functionality or expose it to vulnerabilities.

Code:

Listing 5: smartstaking.sol

```
    function swaptokenchange(uint256 newSwapAmount) external onlyOwner{

        swapTokensAtAmount = newSwapAmount;
    }
```

Risk Level:

Likelihood – 4
Impact - 3

Recommendation:

Implement checks to ensure that newSwapAmount falls within reasonable bounds. What "reasonable" means would depend on the specific use-case for this variable in the contract.

Status - Not Fixed

A.6 Missing Zero Address Check [MEDIUM]

Description:

The setAutomatedMarketMakerPair function does not contain a zero-address check for the pair parameter. This could potentially lead to bugs or misuse of the contract, especially when considering that this function is modifiable only by the owner.

Code:

Listing 6: smartstaking.sol

```
    function setAutomatedMarketMakerPair(address pair, bool value)
        ↪ public onlyOwner {
        require(pair != uniswapV2Pair, "BusmartStakingrn: The PancakeSwap
            ↪  pair cannot be removed from automatedMarketMakerPairs");

        _setAutomatedMarketMakerPair(pair, value);
    }
```

Risk Level:

Likelihood – 3
Impact - 3

Recommendation:

Add a require statement to ensure that the pair address is not a zero address. This would safeguard against inadvertent or malicious attempts to set the pair to an invalid address.

Status - Fixed

A.7 Missing Value Checks [MEDIUM]

Description:

The setMarketingWallet and setDevWallet functions lack essential checks to validate the input addresses. These functions directly set the _marketingWalletAddress and _devWalletAddress, without verifying whether the provided addresses are valid or not. Such an oversight could lead to accidental loss of funds or could be exploited if the owner's account is compromised.

Code:

Listing 7: smartstaking.sol

```
    function setMarketingWallet(address payable wallet) external
        ↪ onlyOwner{
        _marketingWalletAddress = wallet;
    }

    function setDevWallet(address payable wallet) external onlyOwner{
        _devWalletAddress = wallet;
    }
```

Risk Level:

Likelihood – 3
Impact - 3

Recommendation:

Non-zero Address Check: Add a require statement to ensure that the provided address is not the zero address.

Status - Fixed

A.8 Use of .transfer instead of call [MEDIUM]

Description:

The swapAndSendToFee function uses the .transfer method for sending ETH to _marketingWalletAddress and _devWalletAddress. This approach is generally considered less safe for a couple of reasons:

1. If the receiving contract has a fallback function that consumes more than 2300 gas, the .transfer will fail.
2. It lacks flexibility and custom error handling that could be useful for debugging and development.

Code:

Listing 8: smartstaking.sol

```
    function swapAndSendToFee(uint256 tokens) private {

        uint256 initialBalance = address(this).balance;

        swapTokensForEth(tokens);
        uint256 newBalance = address(this).balance.sub(initialBalance);
        uint256 marketingAmount = newBalance.div(totalFees.mul(
            ↪ marketingFee));
        uint256 devWalletAmount = newBalance.div(totalFees.sub(
            ↪ marketingFee).mul(devFee));
        if(marketingAmount > 0) {
            _marketingWalletAddress.transfer(marketingAmount);
        }

        if(devWalletAmount > 0) {
            _devWalletAddress.transfer(devWalletAmount);
        }
        }
```

Risk Level:

Likelihood – 3
Impact - 3

Recommendation:

1. Use .call{value: x}("") or a safeTransfer function : These methods offer better security and more flexibility than .transfer. Using .call will also allow you to check the return value for custom error handling.
2. Gas Checks : If you do opt for .call, make sure you are not making assumptions on the gas needed for the external call.

Status - Fixed

A.9 Centralization Risk :blacklistAddress Function [MEDIUM]

Description:

The blacklistAddress function allows only the contract owner to add or remove addresses from the blacklist. While this might be intended for administrative convenience, it poses a risk of centralization. A single entity controlling who can and cannot interact with the contract could become a potential point of failure and diminishes trust in the decentralized system.

Code:

Listing 9: smartstaking.sol

```
    function blacklistAddress(address account, bool value) external
        ↪ onlyOwner{
        _isBlacklisted[account] = value;
    }
```

Risk Level:

Likelihood – 3

Impact - 3

Recommendation:

1. Decentralized Mechanism: Consider implementing a decentralized control mechanism like a DAO (Decentralized Autonomous Organization) or governance tokens to manage the blacklist. This would give the community a say in who gets blacklisted or removed from it.
2. Multi-Signature Control: Alternatively, control of the blacklist could be assigned to a multi-signature wallet, where multiple trusted parties must agree to blacklist or whitelist an address. This would reduce the risk associated with single point of control.
3. Time-Lock: Introduce a time-lock for sensitive changes to allow users time to react or exit the contract if they disagree with a proposed change.
4. Transparency: Always announce and explain any additions or removals from the blacklist, to maintain trust and transparency with the users.

Status - Fixed

A.10 Missing Balance Check Before Dividend Withdrawal [MEDIUM]

Description:

The function _withdrawDividendOfUser allows users to withdraw dividends without checking if the contract has sufficient BUSD tokens to cover the withdrawal. Failing to check can lead to failed transactions, wasted gas fees, and potential confusion for the users.

Code:

Listing 10: smartstaking.sol

```
    function swapAndSendDividends(uint256 tokens) private{
        swapTokensForBUSD(tokens);
        uint256 dividends = IERC20(BUSD).balanceOf(address(this));
        bool success = IERC20(BUSD).transfer(address(dividendTracker),
            ↪ dividends);

        if (success) {
            dividendTracker.distributeBUSDDividends(dividends);
            emit SendDividends(tokens, dividends);
        }
    }
}
```

Risk Level:

Likelihood – 3
Impact - 3

Recommendation:

Add a balance check before attempting the transfer to ensure that the contract has enough BUSD tokens.

Status - Acknowledged

A.11 Potential Sandwich Attacks [MEDIUM]

Description:

A sandwich attack occurs when an attacker capitalizes on a transaction that swaps tokens or adds liquidity without establishing boundaries on slippage or the minimum output value. By taking advantage of this oversight, the attacker can first influence the exchange rate by executing a transaction ahead of the target (frontrunning) to buy one of the assets. Subsequently, the attacker can profit by selling the asset right after the targeted transaction (backrunning). Therefore, transactions that activate functions without setting slippage restrictions or a minimum output value are susceptible to sandwich attacks, particularly when dealing with large input amounts.

Code:

Listing 11: smartstaking.sol

```
function addLiquidity(uint256 tokenAmount, uint256 ethAmount) private {
        // approve token transfer to cover all possible scenarios
        _approve(address(this), address(uniswapV2Router), tokenAmount);

        // add the liquidity
        uniswapV2Router.addLiquidityETH{value: ethAmount}(
            address(this),
            tokenAmount,
            0, // slippage is unavoidable
            0, // slippage is unavoidable
            owner(),
            block.timestamp
        );

    }
```

Listing 12: smartstaking.sol

```
uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(
            tokenAmount,
            0, // accept any amount of ETH
            path,
            address(this),
            block.timestamp
        );

    }
```

Risk Level:

Likelihood – 3
Impact - 3

Recommendation:

We recommend setting reasonable minimum output amounts, instead of 0, based on token prices when caling the aforementioned functions

Status - Acknowledged

A.12 Inefficient Use of success Boolean and Redundant State Modifications in _withdrawDividendOfUser Function [LOW]

Description:

The _withdrawDividendOfUser function first updates the withdrawnDividends mapping by adding the _withdrawableDividend and then checks the success boolean. If the transfer is not successful, it reverts the change by subtracting _withdrawableDividend. This implementation is not only inefficient but could be simplified by using a require statement to ensure that the transfer is successful before updating the state variable.

Code:

Listing 13: smartstaking.sol

```
function _withdrawDividendOfUser(address payable user) internal returns
  ↪ (uint256) {
  uint256 _withdrawableDividend = withdrawableDividendOf(user);
  if (_withdrawableDividend > 0) {
    withdrawnDividends[user] = withdrawnDividends[user].add(
      ↪ _withdrawableDividend);
    emit DividendWithdrawn(user, _withdrawableDividend);
    bool success = IERC20(BUSD).transfer(user, _withdrawableDividend);

    if(!success) {
      withdrawnDividends[user] = withdrawnDividends[user].sub(
        ↪ _withdrawableDividend);
      return 0;
    }

    return _withdrawableDividend;
  }
```

Risk Level:

Likelihood – 1
Impact - 2

Recommendation:

- 1. Use require Statement : Use a require statement to ensure that the token transfer is successful. This will make the code more readable and efficient.
- 2. Optimize State Changes : Update the state variable withdrawnDividends only after the transfer has been confirmed to be successful. This will remove the need for adding and then potentially subtracting the value, thereby making the function more gas-efficient.

Status - Fixed

A.13 Misleading Function Name setBUSDRewardsFee [LOW]

Description:

The function setBUSDRewardsFee is responsible for setting various fees within the contract, including BUSDRewardsFee, liquidityFee, marketingFee, devFee, and burnFee. However, its name suggests that it's only for setting the BUSDRewardsFee, which can be misleading.

Code:

Listing 14: smartstaking.sol

```
    function setBUSDRewardsFee(uint256 _rewardFee, uint256
       ↪ _liquidityFee, uint256 _marketingFee, uint256 _devFee,
       ↪ uint256 _burnFee) external onlyOwner{
        BUSDRewardsFee = _rewardFee;
        liquidityFee = _liquidityFee;
        marketingFee = _marketingFee;
        devFee = _devFee;
        burnFee = _burnFee;
        totalFees = BUSDRewardsFee.add(liquidityFee).add(marketingFee).
            ↪ add(devFee).add(burnFee);

        require(totalFees <= 20, "Fees Must be 20% Or less");
    }
```

Risk Level:

Likelihood – 1
Impact - 2 Consider renaming the function to reflect its broader scope more accurately. A more descriptive name might be setAllFees or updateFeeSettings.

Status - Fixed

A.14 Blocking Transfers [INFORMATIONAL]

Description:

The _transfer function in DividendPayingToken , responsible for transferring tokens between addresses, has been effectively disabled by the require(false); statement.

Code:

Listing 15: smartstaking.sol

```
    function _transfer(address from, address to, uint256 value) internal
        ↪ virtual override {
    require(false);

    int256 _magCorrection = magnifiedDividendPerShare.mul(value).
        ↪ toInt256Safe();
    magnifiedDividendCorrections[from] = magnifiedDividendCorrections[
        ↪ from].add(_magCorrection);
    magnifiedDividendCorrections[to] = magnifiedDividendCorrections[to].
        ↪ sub(_magCorrection);
```

```
}
```

Listing 16: smartstaking.sol

```
    function _transfer(address, address, uint256) pure internal override
      ↪ {
        require(false, "smartStaking_Dividend_Tracker: No transfers
          ↪ allowed");
    }
```

Risk Level:

Likelihood – 1
Impact - 1

Recommendation:

Re-enable the _transfer function if transfers are intended to be allowed. If not, make sure to document this choice clearly in the comments and rationale.

Status - Fixed

4 Static Analysis (Slither)

Description:

Block Hat expanded the coverage of the specific contract areas using automated testing methodologies. Slither, a Solidity static analysis framework, was one of the tools used. Slither was run on all-scoped contracts in both text and binary formats. This tool can be used to test mathematical relationships between Solidity instances statically and variables that allow for the detection of errors or inconsistent usage of the contracts' APIs throughout the entire codebase.

Results:

```
Compilation warnings/errors on SmartStaking.sol:
Warning: Contract code size is 27014 bytes and exceeds 24576 bytes (a
    ↪ limit introduced in Spurious Dragon). This contract may not be
    ↪ deployable on Mainnet. Consider enabling the optimizer (with a
    ↪ low "runs" value!), turning off revert strings, or using
    ↪ libraries.
    --> SmartStaking.sol:1247:1:
     |
1247 | contract SmartStaking is ERC20, Ownable {
     | ^ (Relevant source part starts here and spans across multiple
         ↪ lines).


INFO:Detectors:
SmartStaking.swapAndSendToFee(uint256) (SmartStaking.sol#1614-1631)
    ↪ sends eth to arbitrary user
        Dangerous calls:
        - _marketingWalletAddress.transfer(marketingAmount) (SmartStaking
            ↪ .sol#1623)
        - _devWalletAddress.transfer(devWalletAmount) (SmartStaking.sol
            ↪ #1627)
```

```
SmartStaking.addLiquidity(uint256,uint256) (SmartStaking.sol#1697-1712)
    ↪ sends eth to arbitrary user
        Dangerous calls:
        - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this)
            ↪ ,tokenAmount,0,0,owner(),block.timestamp) (SmartStaking.
            ↪ sol#1703-1710)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #functions-that-send-ether-to-arbitrary-destinations
INFO:Detectors:
Reentrancy in SmartStaking._transfer(address,address,uint256) (
    ↪ SmartStaking.sol#1531-1612):
        External calls:
        - swapAndSendToFee(marketingTokens) (SmartStaking.sol#1561)
                - uniswapV2Router.
                    ↪ swapExactTokensForETHSupportingFeeOnTransferTokens(
                    ↪ tokenAmount,0,path,address(this),block.timestamp) (
                    ↪ SmartStaking.sol#1668-1674)
        - swapAndSendToFee(buybackTokens) (SmartStaking.sol#1563)
                - uniswapV2Router.
                    ↪ swapExactTokensForETHSupportingFeeOnTransferTokens(
                    ↪ tokenAmount,0,path,address(this),block.timestamp) (
                    ↪ SmartStaking.sol#1668-1674)
        - swapAndLiquify(swapTokens) (SmartStaking.sol#1567)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(
                    ↪ address(this),tokenAmount,0,0,owner(),block.
                    ↪ timestamp) (SmartStaking.sol#1703-1710)
                - uniswapV2Router.
                    ↪ swapExactTokensForETHSupportingFeeOnTransferTokens(
                    ↪ tokenAmount,0,path,address(this),block.timestamp) (
                    ↪ SmartStaking.sol#1668-1674)
        - swapAndSendDividends(sellTokens) (SmartStaking.sol#1570)
                - success = IERC20(BUSD).transfer(address(dividendTracker)
                    ↪ ,dividends) (SmartStaking.sol#1717)
```

```
              - dividendTracker.distributeBUSDDividends(dividends) (
                  ↪ SmartStaking.sol#1720)
              - uniswapV2Router.
                  ↪ swapExactTokensForTokensSupportingFeeOnTransferTokens
                  ↪ (tokenAmount,0,path,address(this),block.timestamp)
                  ↪ (SmartStaking.sol#1688-1694)
        External calls sending eth:
        - swapAndSendToFee(marketingTokens) (SmartStaking.sol#1561)
              - _marketingWalletAddress.transfer(marketingAmount) (
                  ↪ SmartStaking.sol#1623)
              - _devWalletAddress.transfer(devWalletAmount) (
                  ↪ SmartStaking.sol#1627)
        - swapAndSendToFee(buybackTokens) (SmartStaking.sol#1563)
              - _marketingWalletAddress.transfer(marketingAmount) (
                  ↪ SmartStaking.sol#1623)
              - _devWalletAddress.transfer(devWalletAmount) (
                  ↪ SmartStaking.sol#1627)
        - swapAndLiquify(swapTokens) (SmartStaking.sol#1567)
              - uniswapV2Router.addLiquidityETH{value: ethAmount}(
                  ↪ address(this),tokenAmount,0,0,owner(),block.
                  ↪ timestamp) (SmartStaking.sol#1703-1710)
        State variables written after the call(s):
        - super._transfer(from,address(this),fees) (SmartStaking.sol
            ↪ #1589)
              - _balances[sender] = _balances[sender].sub(amount,ERC20:
                  ↪ transfer amount exceeds balance) (SmartStaking.sol
                  ↪ #406)
              - _balances[recipient] = _balances[recipient].add(amount)
                  ↪ (SmartStaking.sol#407)
        ERC20._balances (SmartStaking.sol#222) can be used in cross
            ↪ function reentrancies:
        - ERC20._burn(address,uint256) (SmartStaking.sol#441-449)
        - ERC20._mint(address,uint256) (SmartStaking.sol#420-428)
        - ERC20._transfer(address,address,uint256) (SmartStaking.sol
           ↪ #396-409)
        - ERC20.balanceOf(address) (SmartStaking.sol#287-289)
        - super._burn(from,burnShare) (SmartStaking.sol#1593)
               - _balances[account] = _balances[account].sub(amount,ERC20
                  ↪ : burn amount exceeds balance) (SmartStaking.sol
                  ↪ #446)
        ERC20._balances (SmartStaking.sol#222) can be used in cross
           ↪ function reentrancies:
        - ERC20._burn(address,uint256) (SmartStaking.sol#441-449)
        - ERC20._mint(address,uint256) (SmartStaking.sol#420-428)
        - ERC20._transfer(address,address,uint256) (SmartStaking.sol
           ↪ #396-409)
        - ERC20.balanceOf(address) (SmartStaking.sol#287-289)
        - super._transfer(from,to,amount) (SmartStaking.sol#1597)
               - _balances[sender] = _balances[sender].sub(amount,ERC20:
                  ↪ transfer amount exceeds balance) (SmartStaking.sol
                  ↪ #406)
               - _balances[recipient] = _balances[recipient].add(amount)
                  ↪ (SmartStaking.sol#407)
        ERC20._balances (SmartStaking.sol#222) can be used in cross
           ↪ function reentrancies:
        - ERC20._burn(address,uint256) (SmartStaking.sol#441-449)
        - ERC20._mint(address,uint256) (SmartStaking.sol#420-428)
        - ERC20._transfer(address,address,uint256) (SmartStaking.sol
           ↪ #396-409)
        - ERC20.balanceOf(address) (SmartStaking.sol#287-289)
        - swapping = false (SmartStaking.sol#1572)
        SmartStaking.swapping (SmartStaking.sol#1253) can be used in
           ↪ cross function reentrancies:
        - SmartStaking._transfer(address,address,uint256) (SmartStaking.
           ↪ sol#1531-1612)
   Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
      ↪ #reentrancy-vulnerabilities
```

```
INFO:Detectors:
Reentrancy in DividendPayingToken._withdrawDividendOfUser(address) (
    ↪ SmartStaking.sol#621-637):
        External calls:
        - success = IERC20(BUSD).transfer(user,_withdrawableDividend) (
            ↪ SmartStaking.sol#626)
        State variables written after the call(s):
        - withdrawnDividends[user] = withdrawnDividends[user].sub(
            ↪ _withdrawableDividend) (SmartStaking.sol#629)
        DividendPayingToken.withdrawnDividends (SmartStaking.sol#591) can
            ↪  be used in cross function reentrancies:
        - DividendPayingToken._withdrawDividendOfUser(address) (
            ↪ SmartStaking.sol#621-637)
        - DividendPayingToken.withdrawableDividendOf(address) (
            ↪ SmartStaking.sol#650-652)
        - DividendPayingToken.withdrawnDividendOf(address) (SmartStaking.
            ↪ sol#657-659)
Reentrancy in smartStakingDividendTracker.process(uint256) (SmartStaking
    ↪ .sol#1875-1920):
        External calls:
        - processAccount(address(account),true) (SmartStaking.sol#1901)
                - success = IERC20(BUSD).transfer(user,
                    ↪ _withdrawableDividend) (SmartStaking.sol#626)
        State variables written after the call(s):
        - lastProcessedIndex = _lastProcessedIndex (SmartStaking.sol
            ↪ #1917)
        smartStakingDividendTracker.lastProcessedIndex (SmartStaking.sol
            ↪ #1732) can be used in cross function reentrancies:
        - smartStakingDividendTracker.getAccount(address) (SmartStaking.
            ↪ sol#1786-1829)
        - smartStakingDividendTracker.getLastProcessedIndex() (
            ↪ SmartStaking.sol#1776-1778)
        - smartStakingDividendTracker.lastProcessedIndex (SmartStaking.
            ↪ sol#1732)
        - smartStakingDividendTracker.process(uint256) (SmartStaking.sol
          ↪ #1875-1920)
Reentrancy in SmartStaking.updateDividendTracker(address) (SmartStaking.
    ↪ sol#1363-1378):
        External calls:
        - newDividendTracker.excludeFromDividends(address(
          ↪ newDividendTracker)) (SmartStaking.sol#1370)
        - newDividendTracker.excludeFromDividends(address(this)) (
          ↪ SmartStaking.sol#1371)
        - newDividendTracker.excludeFromDividends(owner()) (SmartStaking.
          ↪ sol#1372)
        - newDividendTracker.excludeFromDividends(address(uniswapV2Router
          ↪ )) (SmartStaking.sol#1373)
        State variables written after the call(s):
        - dividendTracker = newDividendTracker (SmartStaking.sol#1377)
        SmartStaking.dividendTracker (SmartStaking.sol#1255) can be used
          ↪ in cross function reentrancies:
        - SmartStaking._setAutomatedMarketMakerPair(address,bool) (
          ↪ SmartStaking.sol#1440-1449)
        - SmartStaking._transfer(address,address,uint256) (SmartStaking.
          ↪ sol#1531-1612)
        - SmartStaking.claim() (SmartStaking.sol#1518-1520)
        - SmartStaking.constructor() (SmartStaking.sol#1325-1357)
        - SmartStaking.dividendTokenBalanceOf(address) (SmartStaking.sol
          ↪ #1479-1481)
        - SmartStaking.dividendTracker (SmartStaking.sol#1255)
        - SmartStaking.excludeFromDividends(address) (SmartStaking.sol
          ↪ #1483-1485)
        - SmartStaking.getAccountDividendsInfo(address) (SmartStaking.sol
          ↪ #1487-1498)
        - SmartStaking.getAccountDividendsInfoAtIndex(uint256) (
          ↪ SmartStaking.sol#1500-1511)
        - SmartStaking.getClaimWait() (SmartStaking.sol#1463-1465)
        - SmartStaking.getLastProcessedIndex() (SmartStaking.sol
          ↪ #1522-1524)
        - SmartStaking.getNumberOfDividendTokenHolders() (SmartStaking.
          ↪ sol#1526-1528)
        - SmartStaking.getTotalDividendsDistributed() (SmartStaking.sol
          ↪ #1467-1469)
        - SmartStaking.processDividendTracker(uint256) (SmartStaking.sol
          ↪ #1513-1516)
        - SmartStaking.swapAndSendDividends(uint256) (SmartStaking.sol
          ↪ #1714-1723)
        - SmartStaking.updateClaimWait(uint256) (SmartStaking.sol
          ↪ #1459-1461)
        - SmartStaking.updateDividendTracker(address) (SmartStaking.sol
          ↪ #1363-1378)
        - SmartStaking.withdrawableDividendOf(address) (SmartStaking.sol
          ↪ #1475-1477)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #reentrancy-vulnerabilities-1
INFO:Detectors:
SmartStaking._transfer(address,address,uint256).lastProcessedIndex (
    ↪ SmartStaking.sol#1605) is a local variable never initialized
SmartStaking._transfer(address,address,uint256).claims (SmartStaking.sol
    ↪ #1605) is a local variable never initialized
SmartStaking._transfer(address,address,uint256).iterations (SmartStaking
    ↪ .sol#1605) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #uninitialized-local-variables
INFO:Detectors:
SmartStaking.claim() (SmartStaking.sol#1518-1520) ignores return value
    ↪ by dividendTracker.processAccount(address(msg.sender),false) (
    ↪ SmartStaking.sol#1519)
SmartStaking._transfer(address,address,uint256) (SmartStaking.sol
    ↪ #1531-1612) ignores return value by dividendTracker.process(gas)
    ↪ (SmartStaking.sol#1605-1610)
```

```
SmartStaking.addLiquidity(uint256,uint256) (SmartStaking.sol#1697-1712)
    ↪ ignores return value by uniswapV2Router.addLiquidityETH{value:
    ↪ ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp)
    ↪  (SmartStaking.sol#1703-1710)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #unused-return
INFO:Detectors:
DividendPayingToken.constructor(string,string)._name (SmartStaking.sol
    ↪ #595) shadows:
        - ERC20._name (SmartStaking.sol#228) (state variable)
DividendPayingToken.constructor(string,string)._symbol (SmartStaking.sol
    ↪ #595) shadows:
        - ERC20._symbol (SmartStaking.sol#229) (state variable)
DividendPayingToken.dividendOf(address)._owner (SmartStaking.sol#643)
    ↪ shadows:
        - Ownable._owner (SmartStaking.sol#37) (state variable)
DividendPayingToken.withdrawableDividendOf(address)._owner (SmartStaking
    ↪ .sol#650) shadows:
        - Ownable._owner (SmartStaking.sol#37) (state variable)
DividendPayingToken.withdrawnDividendOf(address)._owner (SmartStaking.
    ↪ sol#657) shadows:
        - Ownable._owner (SmartStaking.sol#37) (state variable)
DividendPayingToken.accumulativeDividendOf(address)._owner (SmartStaking
    ↪ .sol#667) shadows:
        - Ownable._owner (SmartStaking.sol#37) (state variable)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #local-variable-shadowing
INFO:Detectors:
SmartStaking.swaptokenchange(uint256) (SmartStaking.sol#1404-1407)
    ↪ should emit an event for:
        - swapTokensAtAmount = newSwapAmount (SmartStaking.sol#1406)
SmartStaking.setBUSDRewardsFee(uint256,uint256,uint256,uint256,uint256)
    ↪ (SmartStaking.sol#1417-1426) should emit an event for:
        - BUSDRewardsFee = _rewardFee (SmartStaking.sol#1418)
        - liquidityFee = _liquidityFee (SmartStaking.sol#1419)
        - marketingFee = _marketingFee (SmartStaking.sol#1420)
        - devFee = _devFee (SmartStaking.sol#1421)
        - burnFee = _burnFee (SmartStaking.sol#1422)
        - totalFees = BUSDRewardsFee.add(liquidityFee).add(marketingFee).
          ↪ add(devFee).add(burnFee) (SmartStaking.sol#1423)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
   ↪ #missing-events-arithmetic
INFO:Detectors:
SmartStaking.updateUniswapV2Router(address)._uniswapV2Pair (SmartStaking
   ↪ .sol#1384-1385) lacks a zero-check on :
                - uniswapV2Pair = _uniswapV2Pair (SmartStaking.sol#1386)
SmartStaking.setMarketingWallet(address).wallet (SmartStaking.sol#1409)
   ↪ lacks a zero-check on :
                - _marketingWalletAddress = wallet (SmartStaking.sol#1410)
SmartStaking.setDevWallet(address).wallet (SmartStaking.sol#1413) lacks
   ↪ a zero-check on :
                - _devWalletAddress = wallet (SmartStaking.sol#1414)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
   ↪ #missing-zero-address-validation
INFO:Detectors:
DividendPayingToken._withdrawDividendOfUser(address) (SmartStaking.sol
   ↪ #621-637) has external calls inside a loop: success = IERC20(BUSD
   ↪ ).transfer(user,_withdrawableDividend) (SmartStaking.sol#626)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
   ↪ /#calls-inside-a-loop
INFO:Detectors:
Reentrancy in SmartStaking._transfer(address,address,uint256) (
   ↪ SmartStaking.sol#1531-1612):
        External calls:
        - swapAndSendToFee(marketingTokens) (SmartStaking.sol#1561)
                - uniswapV2Router.
                   ↪ swapExactTokensForETHSupportingFeeOnTransferTokens(
                   ↪ tokenAmount,0,path,address(this),block.timestamp) (
                    ↪ SmartStaking.sol#1668-1674)
            - swapAndSendToFee(buybackTokens) (SmartStaking.sol#1563)
                    - uniswapV2Router.
                        ↪ swapExactTokensForETHSupportingFeeOnTransferTokens(
                        ↪ tokenAmount,0,path,address(this),block.timestamp) (
                        ↪ SmartStaking.sol#1668-1674)
            External calls sending eth:
            - swapAndSendToFee(marketingTokens) (SmartStaking.sol#1561)
                    - _marketingWalletAddress.transfer(marketingAmount) (
                        ↪ SmartStaking.sol#1623)
                    - _devWalletAddress.transfer(devWalletAmount) (
                        ↪ SmartStaking.sol#1627)
            - swapAndSendToFee(buybackTokens) (SmartStaking.sol#1563)
                    - _marketingWalletAddress.transfer(marketingAmount) (
                        ↪ SmartStaking.sol#1623)
                    - _devWalletAddress.transfer(devWalletAmount) (
                        ↪ SmartStaking.sol#1627)
            State variables written after the call(s):
            - swapAndSendToFee(buybackTokens) (SmartStaking.sol#1563)
                    - _allowances[owner][spender] = amount (SmartStaking.sol
                        ↪ #472)
    Reentrancy in SmartStaking._transfer(address,address,uint256) (
        ↪ SmartStaking.sol#1531-1612):
            External calls:
            - swapAndSendToFee(marketingTokens) (SmartStaking.sol#1561)
                    - uniswapV2Router.
                        ↪ swapExactTokensForETHSupportingFeeOnTransferTokens(
                        ↪ tokenAmount,0,path,address(this),block.timestamp) (
                        ↪ SmartStaking.sol#1668-1674)
            - swapAndSendToFee(buybackTokens) (SmartStaking.sol#1563)
                    - uniswapV2Router.
                        ↪ swapExactTokensForETHSupportingFeeOnTransferTokens(
                        ↪ tokenAmount,0,path,address(this),block.timestamp) (
                        ↪ SmartStaking.sol#1668-1674)
            - swapAndLiquify(swapTokens) (SmartStaking.sol#1567)
                    - uniswapV2Router.addLiquidityETH{value: ethAmount}(
                        ↪ address(this),tokenAmount,0,0,owner(),block.
                        ↪ timestamp) (SmartStaking.sol#1703-1710)
                    - uniswapV2Router.
                        ↪ swapExactTokensForETHSupportingFeeOnTransferTokens(
                        ↪ tokenAmount,0,path,address(this),block.timestamp) (
                        ↪ SmartStaking.sol#1668-1674)
            External calls sending eth:
            - swapAndSendToFee(marketingTokens) (SmartStaking.sol#1561)
                    - _marketingWalletAddress.transfer(marketingAmount) (
                        ↪ SmartStaking.sol#1623)
                    - _devWalletAddress.transfer(devWalletAmount) (
                        ↪ SmartStaking.sol#1627)
            - swapAndSendToFee(buybackTokens) (SmartStaking.sol#1563)
                    - _marketingWalletAddress.transfer(marketingAmount) (
                        ↪ SmartStaking.sol#1623)
                    - _devWalletAddress.transfer(devWalletAmount) (
                        ↪ SmartStaking.sol#1627)
            - swapAndLiquify(swapTokens) (SmartStaking.sol#1567)
                    - uniswapV2Router.addLiquidityETH{value: ethAmount}(
                        ↪ address(this),tokenAmount,0,0,owner(),block.
                        ↪ timestamp) (SmartStaking.sol#1703-1710)
            State variables written after the call(s):
            - swapAndLiquify(swapTokens) (SmartStaking.sol#1567)
                    - _allowances[owner][spender] = amount (SmartStaking.sol
                        ↪ #472)
    Reentrancy in SmartStaking._transfer(address,address,uint256) (
        ↪ SmartStaking.sol#1531-1612):
            External calls:
            - swapAndSendToFee(marketingTokens) (SmartStaking.sol#1561)
                    - uniswapV2Router.
                        ↪ swapExactTokensForETHSupportingFeeOnTransferTokens(
                        ↪ tokenAmount,0,path,address(this),block.timestamp) (
                        ↪ SmartStaking.sol#1668-1674)
            - swapAndSendToFee(buybackTokens) (SmartStaking.sol#1563)
                    - uniswapV2Router.
                        ↪ swapExactTokensForETHSupportingFeeOnTransferTokens(
                        ↪ tokenAmount,0,path,address(this),block.timestamp) (
                        ↪ SmartStaking.sol#1668-1674)
            - swapAndLiquify(swapTokens) (SmartStaking.sol#1567)
                    - uniswapV2Router.addLiquidityETH{value: ethAmount}(
                        ↪ address(this),tokenAmount,0,0,owner(),block.
                        ↪ timestamp) (SmartStaking.sol#1703-1710)
                    - uniswapV2Router.
                        ↪ swapExactTokensForETHSupportingFeeOnTransferTokens(
                        ↪ tokenAmount,0,path,address(this),block.timestamp) (
                        ↪ SmartStaking.sol#1668-1674)
            - swapAndSendDividends(sellTokens) (SmartStaking.sol#1570)
                    - success = IERC20(BUSD).transfer(address(dividendTracker)
                        ↪ ,dividends) (SmartStaking.sol#1717)
                    - dividendTracker.distributeBUSDDividends(dividends) (
                        ↪ SmartStaking.sol#1720)
                    - uniswapV2Router.
                        ↪ swapExactTokensForTokensSupportingFeeOnTransferTokens
                        ↪ (tokenAmount,0,path,address(this),block.timestamp)
                        ↪ (SmartStaking.sol#1688-1694)
            External calls sending eth:
            - swapAndSendToFee(marketingTokens) (SmartStaking.sol#1561)
                    - _marketingWalletAddress.transfer(marketingAmount) (
                        ↪ SmartStaking.sol#1623)
                    - _devWalletAddress.transfer(devWalletAmount) (
                        ↪ SmartStaking.sol#1627)
            - swapAndSendToFee(buybackTokens) (SmartStaking.sol#1563)
                    - _marketingWalletAddress.transfer(marketingAmount) (
                        ↪ SmartStaking.sol#1623)
                    - _devWalletAddress.transfer(devWalletAmount) (
                        ↪ SmartStaking.sol#1627)
```

```
        - swapAndLiquify(swapTokens) (SmartStaking.sol#1567)
               - uniswapV2Router.addLiquidityETH{value: ethAmount}(
                  ↪ address(this),tokenAmount,0,0,owner(),block.
                  ↪ timestamp) (SmartStaking.sol#1703-1710)
        State variables written after the call(s):
        - swapAndSendDividends(sellTokens) (SmartStaking.sol#1570)
               - _allowances[owner][spender] = amount (SmartStaking.sol
                  ↪ #472)
        - super._burn(from,burnShare) (SmartStaking.sol#1593)
               - _totalSupply = _totalSupply.sub(amount) (SmartStaking.
                  ↪ sol#447)
Reentrancy in smartStakingDividendTracker.processAccount(address,bool) (
   ↪ SmartStaking.sol#1922-1932):
        External calls:
        - amount = _withdrawDividendOfUser(account) (SmartStaking.sol
           ↪ #1923)
               - success = IERC20(BUSD).transfer(user,
                  ↪ _withdrawableDividend) (SmartStaking.sol#626)
        State variables written after the call(s):
        - lastClaimTimes[account] = block.timestamp (SmartStaking.sol
           ↪ #1926)
Reentrancy in SmartStaking.swapAndLiquify(uint256) (SmartStaking.sol
   ↪ #1633-1654):
        External calls:
        - swapTokensForEth(half) (SmartStaking.sol#1645)
               - uniswapV2Router.
                  ↪ swapExactTokensForETHSupportingFeeOnTransferTokens(
                  ↪ tokenAmount,0,path,address(this),block.timestamp) (
                  ↪ SmartStaking.sol#1668-1674)
        - addLiquidity(otherHalf,newBalance) (SmartStaking.sol#1651)
               - uniswapV2Router.addLiquidityETH{value: ethAmount}(
                  ↪ address(this),tokenAmount,0,0,owner(),block.
                  ↪ timestamp) (SmartStaking.sol#1703-1710)
        External calls sending eth:
        - addLiquidity(otherHalf,newBalance) (SmartStaking.sol#1651)
               - uniswapV2Router.addLiquidityETH{value: ethAmount}(
                  ↪ address(this),tokenAmount,0,0,owner(),block.
                  ↪ timestamp) (SmartStaking.sol#1703-1710)
        State variables written after the call(s):
        - addLiquidity(otherHalf,newBalance) (SmartStaking.sol#1651)
               - _allowances[owner][spender] = amount (SmartStaking.sol
                  ↪ #472)
Reentrancy in SmartStaking.updateUniswapV2Router(address) (SmartStaking.
   ↪ sol#1380-1387):
        External calls:
        - _uniswapV2Pair = IUniswapV2Factory(uniswapV2Router.factory()).
           ↪ createPair(address(this),uniswapV2Router.WETH()) (
           ↪ SmartStaking.sol#1384-1385)
        State variables written after the call(s):
        - uniswapV2Pair = _uniswapV2Pair (SmartStaking.sol#1386)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
   ↪ #reentrancy-vulnerabilities-2
INFO:Detectors:
Reentrancy in SmartStaking._setAutomatedMarketMakerPair(address,bool) (
   ↪ SmartStaking.sol#1440-1449):
        External calls:
        - dividendTracker.excludeFromDividends(pair) (SmartStaking.sol
           ↪ #1445)
        Event emitted after the call(s):
        - SetAutomatedMarketMakerPair(pair,value) (SmartStaking.sol#1448)
Reentrancy in SmartStaking._transfer(address,address,uint256) (
   ↪ SmartStaking.sol#1531-1612):
        External calls:
        - swapAndSendToFee(marketingTokens) (SmartStaking.sol#1561)
               - uniswapV2Router.
                  ↪ swapExactTokensForETHSupportingFeeOnTransferTokens(
                  ↪ tokenAmount,0,path,address(this),block.timestamp) (
                  ↪ SmartStaking.sol#1668-1674)
```

```
		- swapAndSendToFee(buybackTokens) (SmartStaking.sol#1563)
			- uniswapV2Router.
			   ↪ swapExactTokensForETHSupportingFeeOnTransferTokens(
			   ↪ tokenAmount,0,path,address(this),block.timestamp) (
			   ↪ SmartStaking.sol#1668-1674)
		External calls sending eth:
		- swapAndSendToFee(marketingTokens) (SmartStaking.sol#1561)
			- _marketingWalletAddress.transfer(marketingAmount) (
			   ↪ SmartStaking.sol#1623)
			- _devWalletAddress.transfer(devWalletAmount) (
			   ↪ SmartStaking.sol#1627)
		- swapAndSendToFee(buybackTokens) (SmartStaking.sol#1563)
			- _marketingWalletAddress.transfer(marketingAmount) (
			   ↪ SmartStaking.sol#1623)
			- _devWalletAddress.transfer(devWalletAmount) (
			   ↪ SmartStaking.sol#1627)
		Event emitted after the call(s):
		- Approval(owner,spender,amount) (SmartStaking.sol#473)
			- swapAndSendToFee(buybackTokens) (SmartStaking.sol#1563)
	Reentrancy in SmartStaking._transfer(address,address,uint256) (
	   ↪ SmartStaking.sol#1531-1612):
		External calls:
		- swapAndSendToFee(marketingTokens) (SmartStaking.sol#1561)
			- uniswapV2Router.
			   ↪ swapExactTokensForETHSupportingFeeOnTransferTokens(
			   ↪ tokenAmount,0,path,address(this),block.timestamp) (
			   ↪ SmartStaking.sol#1668-1674)
		- swapAndSendToFee(buybackTokens) (SmartStaking.sol#1563)
			- uniswapV2Router.
			   ↪ swapExactTokensForETHSupportingFeeOnTransferTokens(
			   ↪ tokenAmount,0,path,address(this),block.timestamp) (
			   ↪ SmartStaking.sol#1668-1674)
		- swapAndLiquify(swapTokens) (SmartStaking.sol#1567)
```

```
            - uniswapV2Router.addLiquidityETH{value: ethAmount}(
                ↪ address(this),tokenAmount,0,0,owner(),block.
                ↪ timestamp) (SmartStaking.sol#1703-1710)
            - uniswapV2Router.
                ↪ swapExactTokensForETHSupportingFeeOnTransferTokens(
                ↪ tokenAmount,0,path,address(this),block.timestamp) (
                ↪ SmartStaking.sol#1668-1674)
        External calls sending eth:
        - swapAndSendToFee(marketingTokens) (SmartStaking.sol#1561)
            - _marketingWalletAddress.transfer(marketingAmount) (
                ↪ SmartStaking.sol#1623)
            - _devWalletAddress.transfer(devWalletAmount) (
                ↪ SmartStaking.sol#1627)
        - swapAndSendToFee(buybackTokens) (SmartStaking.sol#1563)
            - _marketingWalletAddress.transfer(marketingAmount) (
                ↪ SmartStaking.sol#1623)
            - _devWalletAddress.transfer(devWalletAmount) (
                ↪ SmartStaking.sol#1627)
        - swapAndLiquify(swapTokens) (SmartStaking.sol#1567)
            - uniswapV2Router.addLiquidityETH{value: ethAmount}(
                ↪ address(this),tokenAmount,0,0,owner(),block.
                ↪ timestamp) (SmartStaking.sol#1703-1710)
        Event emitted after the call(s):
        - Approval(owner,spender,amount) (SmartStaking.sol#473)
            - swapAndLiquify(swapTokens) (SmartStaking.sol#1567)
        - SwapAndLiquify(half,newBalance,otherHalf) (SmartStaking.sol
            ↪ #1653)
            - swapAndLiquify(swapTokens) (SmartStaking.sol#1567)
    Reentrancy in SmartStaking._transfer(address,address,uint256) (
        ↪ SmartStaking.sol#1531-1612):
        External calls:
        - swapAndSendToFee(marketingTokens) (SmartStaking.sol#1561)
            - uniswapV2Router.
                ↪ swapExactTokensForETHSupportingFeeOnTransferTokens(
                ↪ tokenAmount,0,path,address(this),block.timestamp) (
                ↪ SmartStaking.sol#1668-1674)
        - swapAndSendToFee(buybackTokens) (SmartStaking.sol#1563)
            - uniswapV2Router.
                ↪ swapExactTokensForETHSupportingFeeOnTransferTokens(
                ↪ tokenAmount,0,path,address(this),block.timestamp) (
                ↪ SmartStaking.sol#1668-1674)
        - swapAndLiquify(swapTokens) (SmartStaking.sol#1567)
            - uniswapV2Router.addLiquidityETH{value: ethAmount}(
                ↪ address(this),tokenAmount,0,0,owner(),block.
                ↪ timestamp) (SmartStaking.sol#1703-1710)
            - uniswapV2Router.
                ↪ swapExactTokensForETHSupportingFeeOnTransferTokens(
                ↪ tokenAmount,0,path,address(this),block.timestamp) (
                ↪ SmartStaking.sol#1668-1674)
        - swapAndSendDividends(sellTokens) (SmartStaking.sol#1570)
            - success = IERC20(BUSD).transfer(address(dividendTracker)
                ↪ ,dividends) (SmartStaking.sol#1717)
            - dividendTracker.distributeBUSDDividends(dividends) (
                ↪ SmartStaking.sol#1720)
            - uniswapV2Router.
                ↪ swapExactTokensForTokensSupportingFeeOnTransferTokens
                ↪ (tokenAmount,0,path,address(this),block.timestamp)
                ↪ (SmartStaking.sol#1688-1694)
        External calls sending eth:
        - swapAndSendToFee(marketingTokens) (SmartStaking.sol#1561)
            - _marketingWalletAddress.transfer(marketingAmount) (
                ↪ SmartStaking.sol#1623)
            - _devWalletAddress.transfer(devWalletAmount) (
                ↪ SmartStaking.sol#1627)
        - swapAndSendToFee(buybackTokens) (SmartStaking.sol#1563)
            - _marketingWalletAddress.transfer(marketingAmount) (
                ↪ SmartStaking.sol#1623)
```

```
                - _devWalletAddress.transfer(devWalletAmount) (
                   ↪ SmartStaking.sol#1627)
          - swapAndLiquify(swapTokens) (SmartStaking.sol#1567)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(
                   ↪ address(this),tokenAmount,0,0,owner(),block.
                   ↪ timestamp) (SmartStaking.sol#1703-1710)
          Event emitted after the call(s):
          - Approval(owner,spender,amount) (SmartStaking.sol#473)
                - swapAndSendDividends(sellTokens) (SmartStaking.sol#1570)
          - SendDividends(tokens,dividends) (SmartStaking.sol#1721)
                - swapAndSendDividends(sellTokens) (SmartStaking.sol#1570)
          - Transfer(account,address(0),amount) (SmartStaking.sol#448)
                - super._burn(from,burnShare) (SmartStaking.sol#1593)
          - Transfer(sender,recipient,amount) (SmartStaking.sol#408)
                - super._transfer(from,to,amount) (SmartStaking.sol#1597)
          - Transfer(sender,recipient,amount) (SmartStaking.sol#408)
                - super._transfer(from,address(this),fees) (SmartStaking.
                   ↪ sol#1589)
    Reentrancy in SmartStaking._transfer(address,address,uint256) (
       ↪ SmartStaking.sol#1531-1612):
          External calls:
          - swapAndSendToFee(marketingTokens) (SmartStaking.sol#1561)
                - uniswapV2Router.
                   ↪ swapExactTokensForETHSupportingFeeOnTransferTokens(
                   ↪ tokenAmount,0,path,address(this),block.timestamp) (
                   ↪ SmartStaking.sol#1668-1674)
          - swapAndSendToFee(buybackTokens) (SmartStaking.sol#1563)
                - uniswapV2Router.
                   ↪ swapExactTokensForETHSupportingFeeOnTransferTokens(
                   ↪ tokenAmount,0,path,address(this),block.timestamp) (
                   ↪ SmartStaking.sol#1668-1674)
          - swapAndLiquify(swapTokens) (SmartStaking.sol#1567)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(
                   ↪ address(this),tokenAmount,0,0,owner(),block.
                ↪ timestamp) (SmartStaking.sol#1703-1710)
            - uniswapV2Router.
                ↪ swapExactTokensForETHSupportingFeeOnTransferTokens(
                ↪ tokenAmount,0,path,address(this),block.timestamp) (
                ↪ SmartStaking.sol#1668-1674)
        - swapAndSendDividends(sellTokens) (SmartStaking.sol#1570)
            - success = IERC20(BUSD).transfer(address(dividendTracker)
                ↪ ,dividends) (SmartStaking.sol#1717)
            - dividendTracker.distributeBUSDDividends(dividends) (
                ↪ SmartStaking.sol#1720)
            - uniswapV2Router.
                ↪ swapExactTokensForTokensSupportingFeeOnTransferTokens
                ↪ (tokenAmount,0,path,address(this),block.timestamp)
                ↪ (SmartStaking.sol#1688-1694)
        - dividendTracker.setBalance(address(from),balanceOf(from)) (
            ↪ SmartStaking.sol#1599)
        - dividendTracker.setBalance(address(to),balanceOf(to)) (
            ↪ SmartStaking.sol#1600)
        - dividendTracker.process(gas) (SmartStaking.sol#1605-1610)
        External calls sending eth:
        - swapAndSendToFee(marketingTokens) (SmartStaking.sol#1561)
            - _marketingWalletAddress.transfer(marketingAmount) (
                ↪ SmartStaking.sol#1623)
            - _devWalletAddress.transfer(devWalletAmount) (
                ↪ SmartStaking.sol#1627)
        - swapAndSendToFee(buybackTokens) (SmartStaking.sol#1563)
            - _marketingWalletAddress.transfer(marketingAmount) (
                ↪ SmartStaking.sol#1623)
            - _devWalletAddress.transfer(devWalletAmount) (
                ↪ SmartStaking.sol#1627)
        - swapAndLiquify(swapTokens) (SmartStaking.sol#1567)
            - uniswapV2Router.addLiquidityETH{value: ethAmount}(
                ↪ address(this),tokenAmount,0,0,owner(),block.
                ↪ timestamp) (SmartStaking.sol#1703-1710)
        Event emitted after the call(s):
        - ProcessedDividendTracker(iterations,claims,lastProcessedIndex,
           ↪ true,gas,tx.origin) (SmartStaking.sol#1606)
  Reentrancy in smartStakingDividendTracker.processAccount(address,bool) (
     ↪ SmartStaking.sol#1922-1932):
        External calls:
        - amount = _withdrawDividendOfUser(account) (SmartStaking.sol
           ↪ #1923)
              - success = IERC20(BUSD).transfer(user,
                 ↪ _withdrawableDividend) (SmartStaking.sol#626)
        Event emitted after the call(s):
        - Claim(account,amount,automatic) (SmartStaking.sol#1927)
  Reentrancy in SmartStaking.processDividendTracker(uint256) (SmartStaking
     ↪ .sol#1513-1516):
        External calls:
        - (iterations,claims,lastProcessedIndex) = dividendTracker.
           ↪ process(gas) (SmartStaking.sol#1514)
        Event emitted after the call(s):
        - ProcessedDividendTracker(iterations,claims,lastProcessedIndex,
           ↪ false,gas,tx.origin) (SmartStaking.sol#1515)
  Reentrancy in SmartStaking.swapAndLiquify(uint256) (SmartStaking.sol
     ↪ #1633-1654):
        External calls:
        - swapTokensForEth(half) (SmartStaking.sol#1645)
              - uniswapV2Router.
                 ↪ swapExactTokensForETHSupportingFeeOnTransferTokens(
                 ↪ tokenAmount,0,path,address(this),block.timestamp) (
                 ↪ SmartStaking.sol#1668-1674)
        - addLiquidity(otherHalf,newBalance) (SmartStaking.sol#1651)
              - uniswapV2Router.addLiquidityETH{value: ethAmount}(
                 ↪ address(this),tokenAmount,0,0,owner(),block.
                 ↪ timestamp) (SmartStaking.sol#1703-1710)
        External calls sending eth:
        - addLiquidity(otherHalf,newBalance) (SmartStaking.sol#1651)
```

```
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(
                    ↪ address(this),tokenAmount,0,0,owner(),block.
                    ↪ timestamp) (SmartStaking.sol#1703-1710)
            Event emitted after the call(s):
            - Approval(owner,spender,amount) (SmartStaking.sol#473)
                - addLiquidity(otherHalf,newBalance) (SmartStaking.sol
                    ↪ #1651)
            - SwapAndLiquify(half,newBalance,otherHalf) (SmartStaking.sol
                ↪ #1653)
    Reentrancy in SmartStaking.swapAndSendDividends(uint256) (SmartStaking.
        ↪ sol#1714-1723):
            External calls:
            - swapTokensForBUSD(tokens) (SmartStaking.sol#1715)
                - uniswapV2Router.
                    ↪ swapExactTokensForTokensSupportingFeeOnTransferTokens
                    ↪ (tokenAmount,0,path,address(this),block.timestamp)
                    ↪ (SmartStaking.sol#1688-1694)
            - success = IERC20(BUSD).transfer(address(dividendTracker),
                ↪ dividends) (SmartStaking.sol#1717)
            - dividendTracker.distributeBUSDDividends(dividends) (
                ↪ SmartStaking.sol#1720)
            Event emitted after the call(s):
            - SendDividends(tokens,dividends) (SmartStaking.sol#1721)
    Reentrancy in SmartStaking.updateDividendTracker(address) (SmartStaking.
        ↪ sol#1363-1378):
            External calls:
            - newDividendTracker.excludeFromDividends(address(
                ↪ newDividendTracker)) (SmartStaking.sol#1370)
            - newDividendTracker.excludeFromDividends(address(this)) (
                ↪ SmartStaking.sol#1371)
            - newDividendTracker.excludeFromDividends(owner()) (SmartStaking.
                ↪ sol#1372)
            - newDividendTracker.excludeFromDividends(address(uniswapV2Router
                ↪ )) (SmartStaking.sol#1373)
```

```
         Event emitted after the call(s):
         - UpdateDividendTracker(newAddress,address(dividendTracker)) (
             ↪ SmartStaking.sol#1375)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #reentrancy-vulnerabilities-3
INFO:Detectors:
smartStakingDividendTracker.getAccount(address) (SmartStaking.sol
    ↪ #1786-1829) uses timestamp for comparisons
         Dangerous comparisons:
         - nextClaimTime > block.timestamp (SmartStaking.sol#1826-1828)
smartStakingDividendTracker.canAutoClaim(uint256) (SmartStaking.sol
    ↪ #1850-1856) uses timestamp for comparisons
         Dangerous comparisons:
         - lastClaimTime > block.timestamp (SmartStaking.sol#1851)
         - block.timestamp.sub(lastClaimTime) >= claimWait (SmartStaking.
             ↪ sol#1855)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #block-timestamp
INFO:Detectors:
SmartStaking._transfer(address,address,uint256) (SmartStaking.sol
    ↪ #1531-1612) has a high cyclomatic complexity (13).
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #cyclomatic-complexity
INFO:Detectors:
Context._msgData() (SmartStaking.sol#28-31) is never used and should be
    ↪ removed
DividendPayingToken._transfer(address,address,uint256) (SmartStaking.sol
    ↪ #677-683) is never used and should be removed
SafeMath.mod(uint256,uint256) (SmartStaking.sol#1119-1121) is never used
    ↪  and should be removed
SafeMath.mod(uint256,uint256,string) (SmartStaking.sol#1135-1138) is
    ↪ never used and should be removed
SafeMathInt.abs(int256) (SmartStaking.sol#1219-1222) is never used and
    ↪ should be removed
```

```
SafeMathInt.div(int256,int256) (SmartStaking.sol#1190-1196) is never
    ↪ used and should be removed
SafeMathInt.mul(int256,int256) (SmartStaking.sol#1178-1185) is never
    ↪ used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #dead-code
INFO:Detectors:
SmartStaking.totalFees (SmartStaking.sol#1270) is set pre-construction
    ↪ with a non-constant function or state variable:
        - BUSDRewardsFee.add(liquidityFee).add(marketingFee).add(devFee).
            ↪ add(burnFee)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #function-initializing-state
INFO:Detectors:
Parameter DividendPayingToken.dividendOf(address)._owner (SmartStaking.
    ↪ sol#643) is not in mixedCase
Parameter DividendPayingToken.withdrawableDividendOf(address)._owner (
    ↪ SmartStaking.sol#650) is not in mixedCase
Parameter DividendPayingToken.withdrawnDividendOf(address)._owner (
    ↪ SmartStaking.sol#657) is not in mixedCase
Parameter DividendPayingToken.accumulativeDividendOf(address)._owner (
    ↪ SmartStaking.sol#667) is not in mixedCase
Variable DividendPayingToken.BUSD (SmartStaking.sol#569) is not in
    ↪ mixedCase
Constant DividendPayingToken.magnitude (SmartStaking.sol#575) is not in
    ↪ UPPER_CASE_WITH_UNDERSCORES
Function IUniswapV2Pair.DOMAIN_SEPARATOR() (SmartStaking.sol#757) is not
    ↪  in mixedCase
Function IUniswapV2Pair.PERMIT_TYPEHASH() (SmartStaking.sol#758) is not
    ↪ in mixedCase
Function IUniswapV2Pair.MINIMUM_LIQUIDITY() (SmartStaking.sol#775) is
    ↪ not in mixedCase
Function IUniswapV2Router01.WETH() (SmartStaking.sol#797) is not in
    ↪ mixedCase
```

```
Parameter SmartStaking.setBUSDRewardsFee(uint256,uint256,uint256,uint256
    ↪ ,uint256)._rewardFee (SmartStaking.sol#1417) is not in mixedCase
Parameter SmartStaking.setBUSDRewardsFee(uint256,uint256,uint256,uint256
    ↪ ,uint256)._liquidityFee (SmartStaking.sol#1417) is not in
    ↪ mixedCase
Parameter SmartStaking.setBUSDRewardsFee(uint256,uint256,uint256,uint256
    ↪ ,uint256)._marketingFee (SmartStaking.sol#1417) is not in
    ↪ mixedCase
Parameter SmartStaking.setBUSDRewardsFee(uint256,uint256,uint256,uint256
    ↪ ,uint256)._devFee (SmartStaking.sol#1417) is not in mixedCase
Parameter SmartStaking.setBUSDRewardsFee(uint256,uint256,uint256,uint256
    ↪ ,uint256)._burnFee (SmartStaking.sol#1417) is not in mixedCase
Variable SmartStaking.BUSD (SmartStaking.sol#1259) is not in mixedCase
Variable SmartStaking._isBlacklisted (SmartStaking.sol#1263) is not in
    ↪ mixedCase
Variable SmartStaking.BUSDRewardsFee (SmartStaking.sol#1265) is not in
    ↪ mixedCase
Variable SmartStaking._marketingWalletAddress (SmartStaking.sol#1272) is
    ↪  not in mixedCase
Variable SmartStaking._devWalletAddress (SmartStaking.sol#1273) is not
    ↪ in mixedCase
Contract smartStakingDividendTracker (SmartStaking.sol#1726-1933) is not
    ↪  in CapWords
Parameter smartStakingDividendTracker.getAccount(address)._account (
    ↪ SmartStaking.sol#1786) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #conformance-to-solidity-naming-conventions
INFO:Detectors:
Redundant expression "this (SmartStaking.sol#29)" inContext (
    ↪ SmartStaking.sol#23-32)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #redundant-statements
INFO:Detectors:
```

```
Reentrancy in SmartStaking._transfer(address,address,uint256) (
    ↪ SmartStaking.sol#1531-1612):
        External calls:
        - swapAndSendToFee(marketingTokens) (SmartStaking.sol#1561)
                - _marketingWalletAddress.transfer(marketingAmount) (
                    ↪ SmartStaking.sol#1623)
                - _devWalletAddress.transfer(devWalletAmount) (
                    ↪ SmartStaking.sol#1627)
        - swapAndSendToFee(buybackTokens) (SmartStaking.sol#1563)
                - _marketingWalletAddress.transfer(marketingAmount) (
                    ↪ SmartStaking.sol#1623)
                - _devWalletAddress.transfer(devWalletAmount) (
                    ↪ SmartStaking.sol#1627)
        State variables written after the call(s):
        - swapAndSendToFee(buybackTokens) (SmartStaking.sol#1563)
                - _allowances[owner][spender] = amount (SmartStaking.sol
                    ↪ #472)
        Event emitted after the call(s):
        - Approval(owner,spender,amount) (SmartStaking.sol#473)
                - swapAndSendToFee(buybackTokens) (SmartStaking.sol#1563)
Reentrancy in SmartStaking._transfer(address,address,uint256) (
    ↪ SmartStaking.sol#1531-1612):
        External calls:
        - swapAndSendToFee(marketingTokens) (SmartStaking.sol#1561)
                - _marketingWalletAddress.transfer(marketingAmount) (
                    ↪ SmartStaking.sol#1623)
                - _devWalletAddress.transfer(devWalletAmount) (
                    ↪ SmartStaking.sol#1627)
        - swapAndSendToFee(buybackTokens) (SmartStaking.sol#1563)
                - _marketingWalletAddress.transfer(marketingAmount) (
                    ↪ SmartStaking.sol#1623)
                - _devWalletAddress.transfer(devWalletAmount) (
                    ↪ SmartStaking.sol#1627)
        External calls sending eth:
```

```
	- swapAndSendToFee(marketingTokens) (SmartStaking.sol#1561)
		- _marketingWalletAddress.transfer(marketingAmount) (
		   ↪ SmartStaking.sol#1623)
		- _devWalletAddress.transfer(devWalletAmount) (
		   ↪ SmartStaking.sol#1627)
	- swapAndSendToFee(buybackTokens) (SmartStaking.sol#1563)
		- _marketingWalletAddress.transfer(marketingAmount) (
		   ↪ SmartStaking.sol#1623)
		- _devWalletAddress.transfer(devWalletAmount) (
		   ↪ SmartStaking.sol#1627)
	- swapAndLiquify(swapTokens) (SmartStaking.sol#1567)
		- uniswapV2Router.addLiquidityETH{value: ethAmount}(
		   ↪ address(this),tokenAmount,0,0,owner(),block.
		   ↪ timestamp) (SmartStaking.sol#1703-1710)
	State variables written after the call(s):
	- swapAndLiquify(swapTokens) (SmartStaking.sol#1567)
		- _allowances[owner][spender] = amount (SmartStaking.sol
		   ↪ #472)
	- swapAndSendDividends(sellTokens) (SmartStaking.sol#1570)
		- _allowances[owner][spender] = amount (SmartStaking.sol
		   ↪ #472)
	- super._transfer(from,address(this),fees) (SmartStaking.sol
	   ↪ #1589)
		- _balances[sender] = _balances[sender].sub(amount,ERC20:
		   ↪ transfer amount exceeds balance) (SmartStaking.sol
		   ↪ #406)
		- _balances[recipient] = _balances[recipient].add(amount)
		   ↪ (SmartStaking.sol#407)
	- super._burn(from,burnShare) (SmartStaking.sol#1593)
		- _balances[account] = _balances[account].sub(amount,ERC20
		   ↪ : burn amount exceeds balance) (SmartStaking.sol
		   ↪ #446)
	- super._transfer(from,to,amount) (SmartStaking.sol#1597)
```

```
                - _balances[sender] = _balances[sender].sub(amount,ERC20:
                   ↪ transfer amount exceeds balance) (SmartStaking.sol
                   ↪ #406)
                - _balances[recipient] = _balances[recipient].add(amount)
                   ↪ (SmartStaking.sol#407)
        - super._burn(from,burnShare) (SmartStaking.sol#1593)
                - _totalSupply = _totalSupply.sub(amount) (SmartStaking.
                   ↪ sol#447)
        - swapping = false (SmartStaking.sol#1572)
        Event emitted after the call(s):
        - Approval(owner,spender,amount) (SmartStaking.sol#473)
                - swapAndSendDividends(sellTokens) (SmartStaking.sol#1570)
        - Approval(owner,spender,amount) (SmartStaking.sol#473)
                - swapAndLiquify(swapTokens) (SmartStaking.sol#1567)
        - ProcessedDividendTracker(iterations,claims,lastProcessedIndex,
           ↪ true,gas,tx.origin) (SmartStaking.sol#1606)
        - SendDividends(tokens,dividends) (SmartStaking.sol#1721)
                - swapAndSendDividends(sellTokens) (SmartStaking.sol#1570)
        - SwapAndLiquify(half,newBalance,otherHalf) (SmartStaking.sol
           ↪ #1653)
                - swapAndLiquify(swapTokens) (SmartStaking.sol#1567)
        - Transfer(account,address(0),amount) (SmartStaking.sol#448)
                - super._burn(from,burnShare) (SmartStaking.sol#1593)
        - Transfer(sender,recipient,amount) (SmartStaking.sol#408)
                - super._transfer(from,address(this),fees) (SmartStaking.
                   ↪ sol#1589)
        - Transfer(sender,recipient,amount) (SmartStaking.sol#408)
                - super._transfer(from,to,amount) (SmartStaking.sol#1597)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
   ↪ #reentrancy-vulnerabilities-4
INFO:Detectors:
Variable IUniswapV2Router01.addLiquidity(address,address,uint256,uint256
   ↪ ,uint256,uint256,address,uint256).amountADesired (SmartStaking.
   ↪ sol#802) is too similar to IUniswapV2Router01.addLiquidity(
    ↪ address,address,uint256,uint256,uint256,uint256,address,uint256).
    ↪ amountBDesired (SmartStaking.sol#803)
Variable DividendPayingToken._withdrawDividendOfUser(address).
    ↪ _withdrawableDividend (SmartStaking.sol#622) is too similar to
    ↪ smartStakingDividendTracker.getAccount(address).
    ↪ withdrawableDividends (SmartStaking.sol#1791)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #variable-names-too-similar
INFO:Detectors:
SmartStaking.constructor() (SmartStaking.sol#1325-1357) uses literals
    ↪ with too many digits:
        - _mint(owner(),10000000 * (10 ** 18)) (SmartStaking.sol#1356)
SmartStaking.updateGasForProcessing(uint256) (SmartStaking.sol
    ↪ #1452-1457) uses literals with too many digits:
        - require(bool,string)(newValue >= 200000 && newValue <= 500000,
            ↪ smartStaking: gasForProcessing must be between 200,000 and
            ↪  500,000) (SmartStaking.sol#1453)
SmartStaking.slitherConstructorVariables() (SmartStaking.sol#1247-1724)
    ↪ uses literals with too many digits:
        - gasForProcessing = 300000 (SmartStaking.sol#1277)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #too-many-digits
INFO:Detectors:
SafeMathInt.MAX_INT256 (SmartStaking.sol#1173) is never used in
    ↪ SafeMathInt (SmartStaking.sol#1171-1229)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #unused-state-variable
INFO:Detectors:
SmartStaking.deadWallet (SmartStaking.sol#1257) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #state-variables-that-could-be-declared-constant
INFO:Detectors:
smartStakingDividendTracker.minimumTokenBalanceForDividends (
    ↪ SmartStaking.sol#1739) should be immutable
```

```
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #state-variables-that-could-be-declared-immutable
INFO:Slither:SmartStaking.sol analyzed (18 contracts with 85 detectors),
    ↪  84 result(s) found
```

Conclusion:

Most of the vulnerabilities found by the analysis have already been addressed by the smart contract code review.

5 Conclusion

In this audit, we examined the design and implementation of Smart Staking contract and discovered several issues of varying severity. $MART team addressed all the issues raised in the initial report and implemented the necessary fixes.

The present code base is well-structured and ready for the mainnet.

BLOCKHAT
SECURITY

For a Smart Contract Audit, contact us at contact@blockhat.io