QuillAudits

Audit Report
November, 2023

For

YYDS

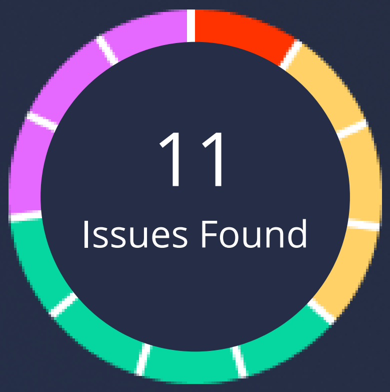
Table of Content

Table of Content

Executive Summary

Project Name	YYDS
Project URL	*https://swap-bsc.tokentool.club/#/swap*
Overview	The YYDS token mints 1,000,000 tokens to the tokenOwner at deployment. It implements bottlenecks to token amounts to transfer per transaction, as well as minimum tokens to be available for burns/swaps. It has an internal deflationary mechanism which activates on transfers when trade is enabled. The deflation transfers tokens out of the pair to its vault and other protocol-defined addresses (shareholder, treasurer, nodes, etc).
Audit Scope	*https://github.com/yydsfinance/yyds/blob/main/contracts/Token/YYDS.sol*
Contracts in Scope	YYDS.sol
Commit Hash	*9239dc8b038ab41c8e90f4ffa0d4694a47924d81*
Language	Solidity
Blockchain	Binance Smart Chain
Method	Manual Analysis, Functional Testing, Automated Testing
Review 1	6th November 2023 - 20th November 2023
Updated Code Received	NA
Review 2	NA
Fixed In	NA

Number of Security Issues per Severity



	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	**1**	**3**	**4**	**3**
Partially Resolved Issues	0	0	0	0
Resolved Issues	0	0	0	0

Checked Vulnerabilities

- ✓ Access Management
- ✓ Arbitrary write to storage
- ✓ Centralization of control
- ✓ Ether theft
- ✓ Improper or missing events
- ✓ Logical issues and flaws
- ✓ Arithmetic Correctness
- ✓ Race conditions/front running
- ✓ SWC Registry
- ✓ Re-entrancy
- ✓ Timestamp Dependence
- ✓ Gas Limit and Loops
- ✓ Exception Disorder
- ✓ Malicious libraries
- ✓ Compiler version not fixed
- ✓ Address hardcoded
- ✓ Divide before multiply
- ✓ Integer overflow/underflow
- ✓ ERC's conformance
- ✓ Missing Zero Address Validation
- ✓ Private modifier
- ✓ Revert/require functions
- ✓ Upgradeable safety
- ✓ Using throw
- ✓ Style guide violation
- ✓ Unsafe type inference
- ✓ Implicit visibility level

Techniques and Methods

Throughout the audit of smart contracts, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments, match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour mentioned in the whitepaper.
- Implementation of ERC standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods, and tools were used to review all the smart contracts.

Structural Analysis

In this step, we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

A static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual Analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behavior of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit

Foundry, Slither, Solidity Static Analysis.

Types of Severity

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below.

High Severity Issues

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium Severity Issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low Severity Issues

Low-level severity issues can cause minor impact and are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are four severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

High Severity Issues

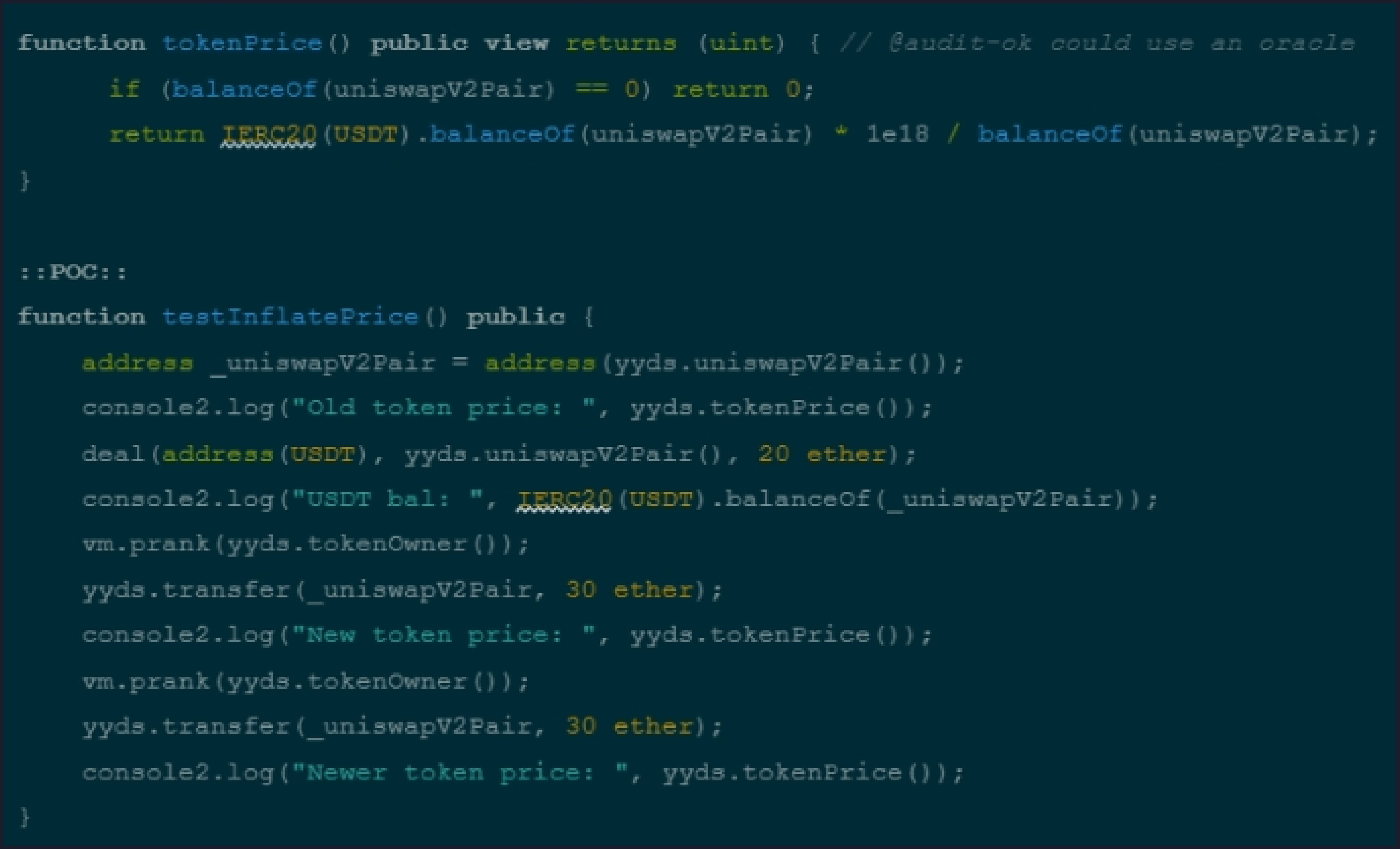
A.1: On-chain generated prices can be manipulated

File(s)

YYDS.sol

Description

On-chain generated prices are susceptible to price manipulation as these values can be affected by users of the protocol. For now, token price is calculated via the tokenPrice() function below.



```
function tokenPrice() public view returns (uint) { // @audit-ok could use an oracle
      if (balanceOf(uniswapV2Pair) == 0) return 0;
      return IERC20(USDT).balanceOf(uniswapV2Pair) * 1e18 / balanceOf(uniswapV2Pair);
}


::POC::
function testInflatePrice() public {
    address _uniswapV2Pair = address(yyds.uniswapV2Pair());
    console2.log("Old token price: ", yyds.tokenPrice());
    deal(address(USDT), yyds.uniswapV2Pair(), 20 ether);
    console2.log("USDT bal: ", IERC20(USDT).balanceOf(_uniswapV2Pair));
    vm.prank(yyds.tokenOwner());
    yyds.transfer(_uniswapV2Pair, 30 ether);
    console2.log("New token price: ", yyds.tokenPrice());
    vm.prank(yyds.tokenOwner());
    yyds.transfer(_uniswapV2Pair, 30 ether);
    console2.log("Newer token price: ", yyds.tokenPrice());
}
```

Recommendation(s)

Use other sources of price validation, as any user with malicious intent and sufficient tokens can easily adjust the token price in between transactions for better prices. Consider using oracles as sources of price validity.

Status

Acknowledged

A.1: On-chain generated prices can be manipulated

Client's Comment

This tokenPrice just to estimate the amount to swap, it cannot be used anywhere else, so if be manipulated, it does not matter.

Auditor's Comment

price = tokenPrice() is used in calculations. If price is adjusted, the if statement could be skipped from the execution thereby stopping shareholderAddr, Vault and nodes from receiving dividends causing a possible griefing issue for the protocol owners.

Medium Severity Issues

A.2: Improper token accounting

File(s)

YYDS.sol

Description

In swapAndDividend(), YYDS tokens are swapped to their USDT equivalent via the uniswapV2Router. After the swap, all the USDT in tokenReceiver is transferred to the marketingAddr and treasureAddr and this could lead to accounting errors if there was some USDT already in the tokenReceiver before the call to swapAndDividend().

Scenario

If another user sends in 1,000 USDT tokens to the tokenReceiver and the swap yields 500 USDT, it'll send ~1,333USDT to the marketingAddr and ~166USDT to the treasureAddr which is 300% more than expected.

Recommendation(s)

The balance before and after the swap can be used to obtain the difference, and transfers would be made based on this difference to avoid accounting errors or misplaced funds.

Status

Acknowledged

A.3: Use of two-step ownable

File(s)

YYDS.sol

Description

If ownership transfer is not done appropriately, the current contract owner can renounce/ transfer ownership, lose owner privileges, and lose the ability to burn tokens. When ownership is renounced, all of the contract's methods will be rendered unusable.

Remediation

Functions such as renounceOwnership and transferOwnership can be overridden or set up for 2-step verification to prevent mistaken privilege transfer or renouncing.

References

https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/Ownable2Step.sol and https://github.com/razzorsec/RazzorSec-Contracts/blob/main/AccessControl/SafeOwn.sol

Status

Acknowledged

A.4: Missing test cases

Description

The YYDS token contract does not have any test cases and has not shown any results of thoroughly testing the protocol before launch. It is advised to have >95% code coverage before deploying protocols so as to reduce the surface area open to errors.

Recommendation(s)

Add unit tests using a blockchain development suite like Hardhat or Foundry to thoroughly test the codebase as this is a timeboxed audit.

Status

Acknowledged

Low Severity Issues

A.5: No way to withdraw stuck tokens/BNB

Description

In the course of transacting with contract addresses and swaps, there could be tokens mistakenly sent into the YYDS token contract and will remain stuck there with no means to claim them.

Recommendation(s)

Create a withdrawStuckToken() function that allows users to withdraw tokens they sent into the contract by error. Special care should be taken to not allow YYDS or USDT to be withdrawn through this function's addition.

Status

Acknowledged

A.6: Outdated libraries

Description

The codebase is replete with old packages from libraries that have gotten updates and newer releases. If the older versions are still in use and they contain bugs, then the entire codebase would be opened up to a larger attack surface than necessary. The ERC20 library used is an older version without the newer internal checks and custom error messages, it also uses the SafeMath library which is redundant from solidity versions >0.8.0 due to internal overflow and underflow checks.

The SafeERC20 library also exposes the safeApprove function which is deprecated to favor the use of safeIncreaseAllowance and safeDecreaseAllowance instead.

Issue

https://github.com/OpenZeppelin/openzeppelin-contracts/issues/2219#issuecomment-622163352

https://github.com/OpenZeppelin/openzeppelin-contracts/blob/566a774222707e424896c0c390a84dc3c13bdcb2/contracts/token/ERC20/utils/SafeERC20.sol#L38

A.6: Outdated libraries

Remediation

Use tested newer libraries and packages when available. Solidity 0.8.0 also has internal checks for overflows and underflows, thereby reducing the need for external libraries like SafeMath which in turn would cause lesser contract sizes, deployment costs, and lower fees for users per transaction.

Status

Acknowledged

A.7: Variables can be converted into immutable instead of constant

Description

For addresses and values that do not change after deployment, they can be stored as immutable values to reduce gas costs for reading from storage. Some of these variables are tokenOwner, tokenReceiver, uniswapV2Pair and uniswapV2Router all set at deployment time in the constructor.

Recommendation(s)

Convert the variables tokenOwner, tokenReceiver, uniswapV2Pair and uniswapV2Router to immutable.

Status

Acknowledged

A.8: Missing event emission

Description

Some of the contract functions that update the token's state do not emit events when called. It is advisable for ease of tracking changes on the blockchain to emit events. The following functions could have events emitted when called: setBuyInterval, setMaxTokenVaulePerTx, setMaxTokenVauleSellPerTx, setIntervalDays, setBuyMarketingFee, setSellMarketingFee, setNumTokensSellToSwap, setMinAmountDeflation, and setMinAmountBurn.

Recommendation(s)

Emit events for these changes to the token's state for ease of tracking and logging.

Status

Acknowledged

Informational Issues

A.9: Unlocked pragma (pragma solidity ^0.8.0)

Description

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Remediation

Here all the in-scope contracts have an unlocked pragma, it is recommended to lock the same. Moreover, we strongly suggest not to use experimental Solidity features (e.g., pragma experimental ABIEncoderV2) or third-party unaudited libraries. If necessary, refactor the current code base to only use stable features.

Status

Acknowledged

A.10: Functions guaranteed to revert can be marked payable

Description

Throughout the codebase, there are functions marked as onlyOwner. These functions are guaranteed to revert when a regular user tries to call them. To reduce the amount of gas other users would waste if they call the function, the functions can carry the payable function modifier.

Remediation

Make the functions payable to reduce the amount of gas spent on failed transactions.

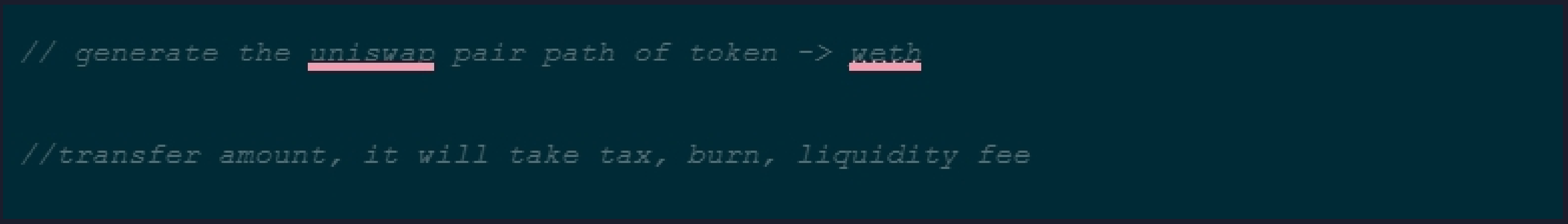
Status

Acknowledged

A.11: Wrong comments

Description

Some lines of code (comments) do not match the codebase specs. They can be removed or updated to match what exactly they do.



```
// generate the uniswap pair path of token -> weth


//transfer amount, it will take tax, burn, liquidity fee
```

Remediation

Remove the comments that aren't used or update them to match the specs of the codebase.

Status

Acknowledged

Functional Tests

Some of the tests performed are mentioned below:

✓ Should transfer 5% of pairBalance to 0xdead address

✓ Should transfer 0.2% of pairBalance to nodes[0]

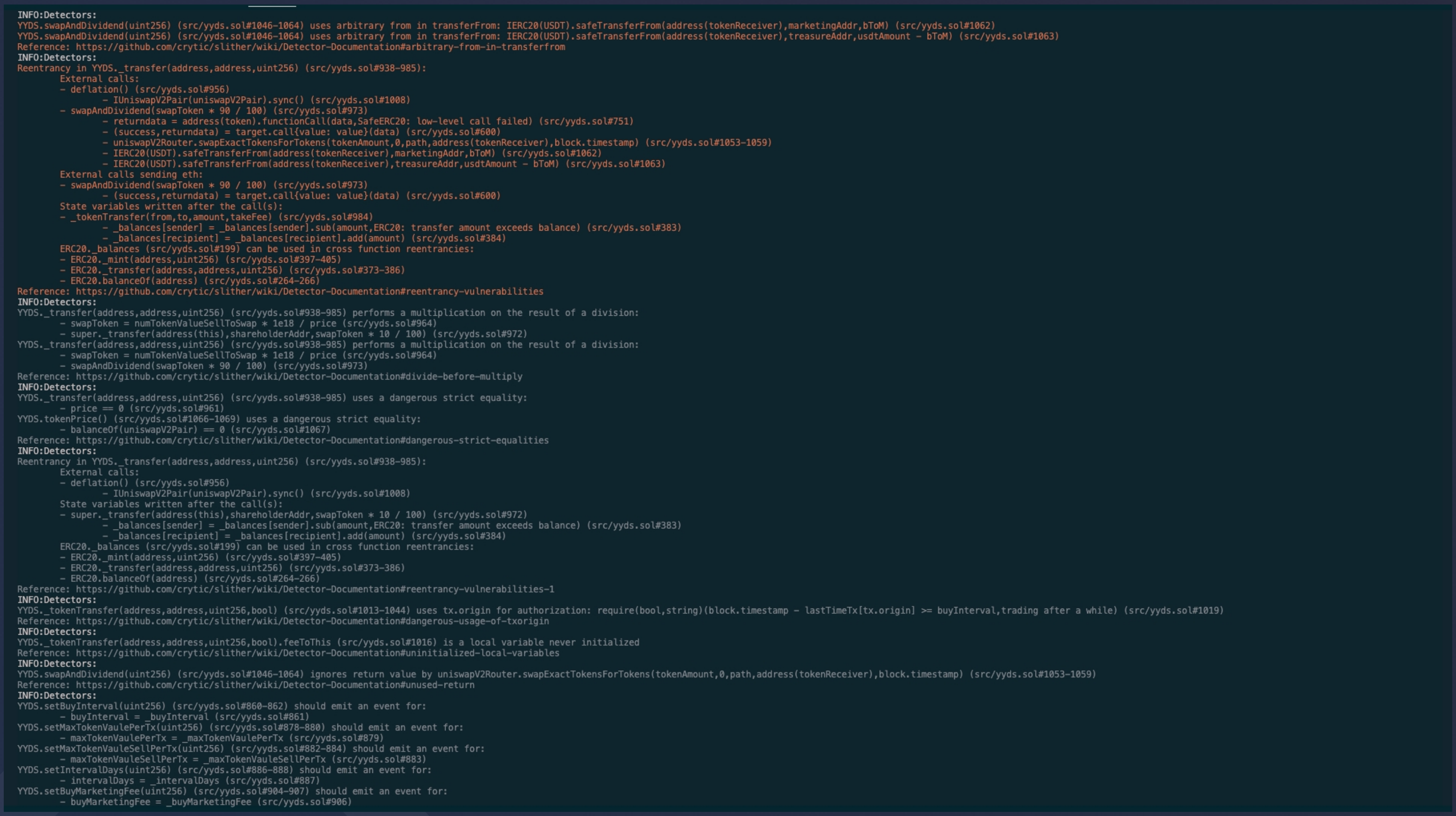
✓ Should transfer 0.3% of pairBalance to nodes[1]

✓ Should transfer an equal amount of tokens to other nodes

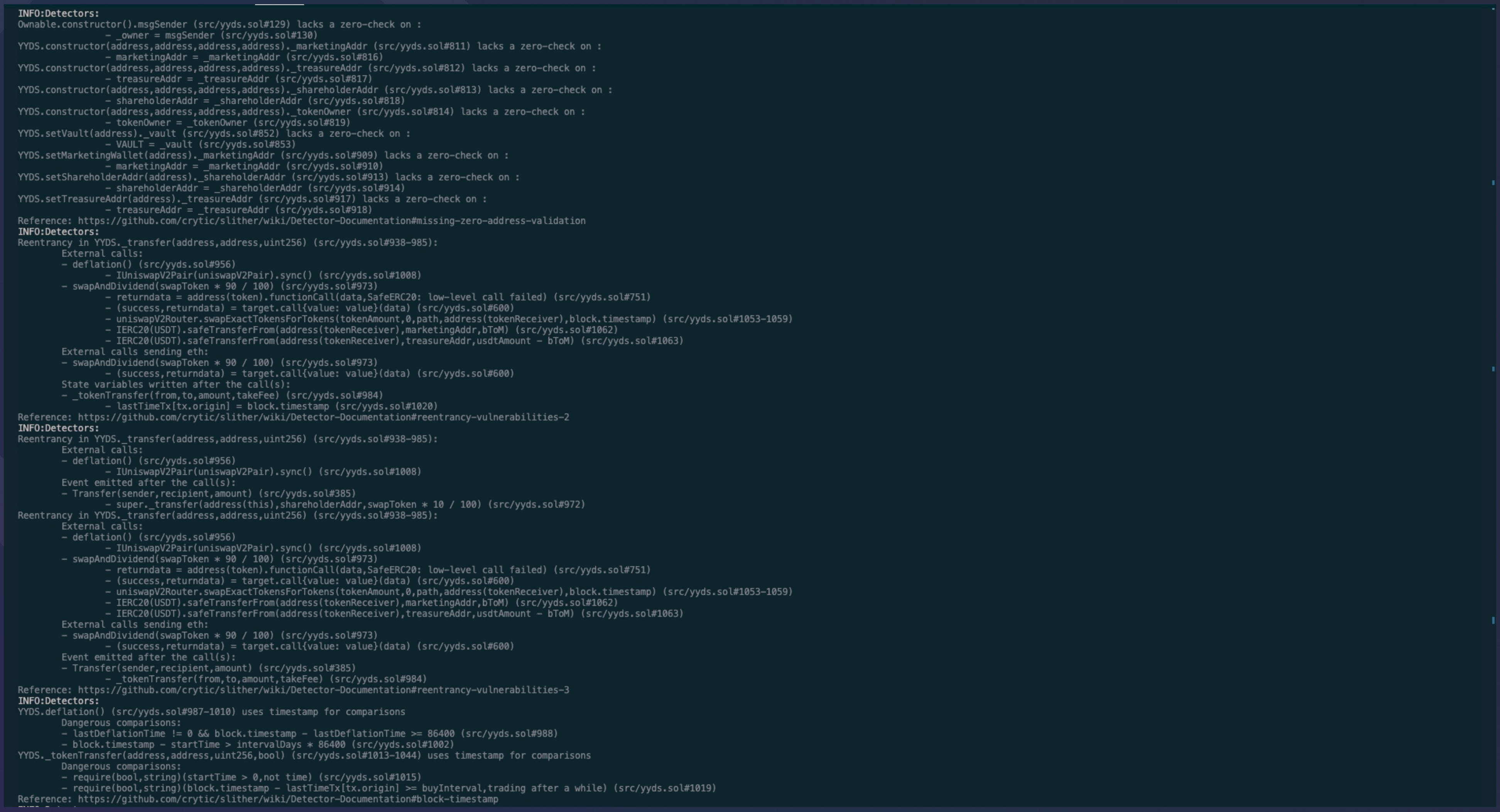
✗ Should not inflate tokenPrice()

Automated Tests

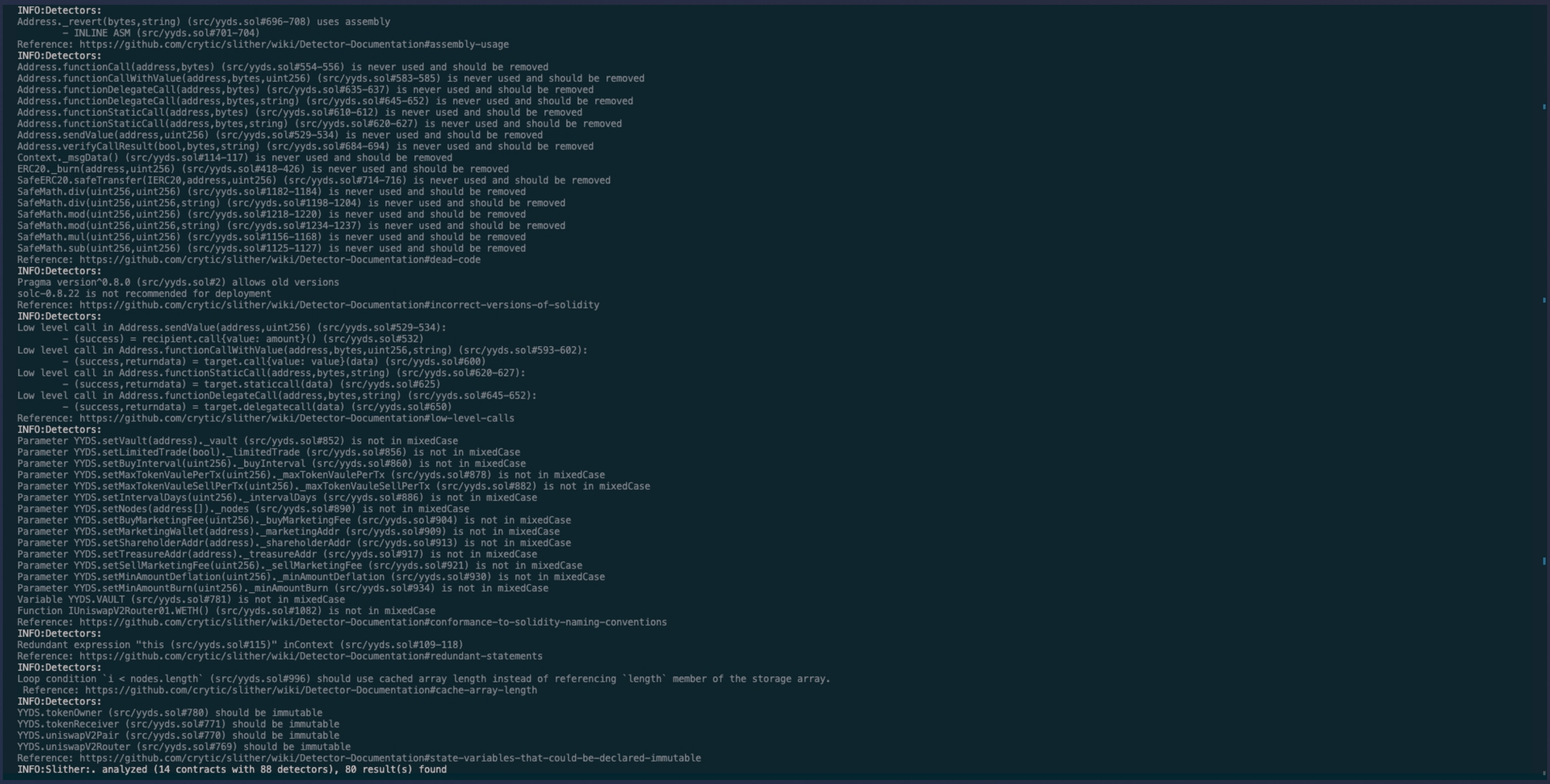
No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.



```
INFO:Detectors:
YYDS.swapAndDividend(uint256) (src/yyds.sol#1046-1064) uses arbitrary from in transferFrom: IERC20(USDT).safeTransferFrom(address(tokenReceiver),marketingAddr,bToM) (src/yyds.sol#1062)
YYDS.swapAndDividend(uint256) (src/yyds.sol#1046-1064) uses arbitrary from in transferFrom: IERC20(USDT).safeTransferFrom(address(tokenReceiver),treasureAddr,usdtAmount - bToM) (src/yyds.sol#1063)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#arbitrary-from-in-transferfrom
INFO:Detectors:
Reentrancy in YYDS._transfer(address,address,uint256) (src/yyds.sol#938-985):
        External calls:
        - deflation() (src/yyds.sol#956)
                - IUniswapV2Pair(uniswapV2Pair).sync() (src/yyds.sol#1008)
        - swapAndDividend(swapToken * 90 / 100) (src/yyds.sol#973)
                - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (src/yyds.sol#751)
                - (success,returndata) = target.call{value: value}(data) (src/yyds.sol#600)
                - uniswapV2Router.swapExactTokensForTokens(tokenAmount,0,path,address(tokenReceiver),block.timestamp) (src/yyds.sol#1053-1059)
                - IERC20(USDT).safeTransferFrom(address(tokenReceiver),marketingAddr,bToM) (src/yyds.sol#1062)
                - IERC20(USDT).safeTransferFrom(address(tokenReceiver),treasureAddr,usdtAmount - bToM) (src/yyds.sol#1063)
        External calls sending eth:
        - swapAndDividend(swapToken * 90 / 100) (src/yyds.sol#973)
                - (success,returndata) = target.call{value: value}(data) (src/yyds.sol#600)
        State variables written after the call(s):
        - _tokenTransfer(from,to,amount,takeFee) (src/yyds.sol#984)
                - _balances[sender] = _balances[sender].sub(amount,ERC20: transfer amount exceeds balance) (src/yyds.sol#383)
                - _balances[recipient] = _balances[recipient].add(amount) (src/yyds.sol#384)
        ERC20._balances (src/yyds.sol#199) can be used in cross function reentrancies:
        - ERC20._mint(address,uint256) (src/yyds.sol#397-405)
        - ERC20._transfer(address,address,uint256) (src/yyds.sol#373-386)
        - ERC20.balanceOf(address) (src/yyds.sol#264-266)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities
INFO:Detectors:
YYDS._transfer(address,address,uint256) (src/yyds.sol#938-985) performs a multiplication on the result of a division:
        - swapToken = numTokenValueSellToSwap * 1e18 / price (src/yyds.sol#964)
        - super._transfer(address(this),shareholderAddr,swapToken * 10 / 100) (src/yyds.sol#972)
YYDS._transfer(address,address,uint256) (src/yyds.sol#938-985) performs a multiplication on the result of a division:
        - swapToken = numTokenValueSellToSwap * 1e18 / price (src/yyds.sol#964)
        - swapAndDividend(swapToken * 90 / 100) (src/yyds.sol#973)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
YYDS._transfer(address,address,uint256) (src/yyds.sol#938-985) uses a dangerous strict equality:
        - price == 0 (src/yyds.sol#961)
YYDS.tokenPrice() (src/yyds.sol#1066-1069) uses a dangerous strict equality:
        - balanceOf(uniswapV2Pair) == 0 (src/yyds.sol#1067)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
Reentrancy in YYDS._transfer(address,address,uint256) (src/yyds.sol#938-985):
        External calls:
        - deflation() (src/yyds.sol#956)
                - IUniswapV2Pair(uniswapV2Pair).sync() (src/yyds.sol#1008)
        State variables written after the call(s):
        - super._transfer(address(this),shareholderAddr,swapToken * 10 / 100) (src/yyds.sol#972)
                - _balances[sender] = _balances[sender].sub(amount,ERC20: transfer amount exceeds balance) (src/yyds.sol#383)
                - _balances[recipient] = _balances[recipient].add(amount) (src/yyds.sol#384)
        ERC20._balances (src/yyds.sol#199) can be used in cross function reentrancies:
        - ERC20._mint(address,uint256) (src/yyds.sol#397-405)
        - ERC20._transfer(address,address,uint256) (src/yyds.sol#373-386)
        - ERC20.balanceOf(address) (src/yyds.sol#264-266)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
YYDS._tokenTransfer(address,address,uint256,bool) (src/yyds.sol#1013-1044) uses tx.origin for authorization: require(bool,string)(block.timestamp - lastTimeTx[tx.origin] >= buyInterval,trading after a while) (src/yyds.sol#1019)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-usage-of-txorigin
INFO:Detectors:
YYDS._tokenTransfer(address,address,uint256,bool).feeToThis (src/yyds.sol#1016) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
INFO:Detectors:
YYDS.swapAndDividend(uint256) (src/yyds.sol#1046-1064) ignores return value by uniswapV2Router.swapExactTokensForTokens(tokenAmount,0,path,address(tokenReceiver),block.timestamp) (src/yyds.sol#1053-1059)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
YYDS.setBuyInterval(uint256) (src/yyds.sol#860-862) should emit an event for:
        - buyInterval = _buyInterval (src/yyds.sol#861)
YYDS.setMaxTokenVaulePerTx(uint256) (src/yyds.sol#878-880) should emit an event for:
        - maxTokenVaulePerTx = _maxTokenVaulePerTx (src/yyds.sol#879)
YYDS.setMaxTokenVauleSellPerTx(uint256) (src/yyds.sol#882-884) should emit an event for:
        - maxTokenVauleSellPerTx = _maxTokenVauleSellPerTx (src/yyds.sol#883)
YYDS.setIntervalDays(uint256) (src/yyds.sol#886-888) should emit an event for:
        - intervalDays = _intervalDays (src/yyds.sol#887)
YYDS.setBuyMarketingFee(uint256) (src/yyds.sol#904-907) should emit an event for:
        - buyMarketingFee = _buyMarketingFee (src/yyds.sol#906)
```



```
INFO:Detectors:
Ownable.constructor().msgSender (src/yyds.sol#129) lacks a zero-check on :
                - _owner = msgSender (src/yyds.sol#130)
YYDS.constructor(address,address,address,address)._marketingAddr (src/yyds.sol#811) lacks a zero-check on :
                - marketingAddr = _marketingAddr (src/yyds.sol#816)
YYDS.constructor(address,address,address,address)._treasureAddr (src/yyds.sol#812) lacks a zero-check on :
                - treasureAddr = _treasureAddr (src/yyds.sol#817)
YYDS.constructor(address,address,address,address)._shareholderAddr (src/yyds.sol#813) lacks a zero-check on :
                - shareholderAddr = _shareholderAddr (src/yyds.sol#818)
YYDS.constructor(address,address,address,address)._tokenOwner (src/yyds.sol#814) lacks a zero-check on :
                - tokenOwner = _tokenOwner (src/yyds.sol#819)
YYDS.setVault(address)._vault (src/yyds.sol#852) lacks a zero-check on :
                - VAULT = _vault (src/yyds.sol#853)
YYDS.setMarketingWallet(address)._marketingAddr (src/yyds.sol#909) lacks a zero-check on :
                - marketingAddr = _marketingAddr (src/yyds.sol#910)
YYDS.setShareholderAddr(address)._shareholderAddr (src/yyds.sol#913) lacks a zero-check on :
                - shareholderAddr = _shareholderAddr (src/yyds.sol#914)
YYDS.setTreasureAddr(address)._treasureAddr (src/yyds.sol#917) lacks a zero-check on :
                - treasureAddr = _treasureAddr (src/yyds.sol#918)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Reentrancy in YYDS._transfer(address,address,uint256) (src/yyds.sol#938-985):
        External calls:
        - deflation() (src/yyds.sol#956)
                - IUniswapV2Pair(uniswapV2Pair).sync() (src/yyds.sol#1008)
        - swapAndDividend(swapToken * 90 / 100) (src/yyds.sol#973)
                - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (src/yyds.sol#751)
                - (success,returndata) = target.call{value: value}(data) (src/yyds.sol#600)
                - uniswapV2Router.swapExactTokensForTokens(tokenAmount,0,path,address(tokenReceiver),block.timestamp) (src/yyds.sol#1053-1059)
                - IERC20(USDT).safeTransferFrom(address(tokenReceiver),marketingAddr,bToM) (src/yyds.sol#1062)
                - IERC20(USDT).safeTransferFrom(address(tokenReceiver),treasureAddr,usdtAmount - bToM) (src/yyds.sol#1063)
        External calls sending eth:
        - swapAndDividend(swapToken * 90 / 100) (src/yyds.sol#973)
                - (success,returndata) = target.call{value: value}(data) (src/yyds.sol#600)
        State variables written after the call(s):
        - _tokenTransfer(from,to,amount,takeFee) (src/yyds.sol#984)
                - lastTimeTx[tx.origin] = block.timestamp (src/yyds.sol#1020)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
Reentrancy in YYDS._transfer(address,address,uint256) (src/yyds.sol#938-985):
        External calls:
        - deflation() (src/yyds.sol#956)
                - IUniswapV2Pair(uniswapV2Pair).sync() (src/yyds.sol#1008)
        Event emitted after the call(s):
        - Transfer(sender,recipient,amount) (src/yyds.sol#385)
                - super._transfer(address(this),shareholderAddr,swapToken * 10 / 100) (src/yyds.sol#972)
Reentrancy in YYDS._transfer(address,address,uint256) (src/yyds.sol#938-985):
        External calls:
        - deflation() (src/yyds.sol#956)
                - IUniswapV2Pair(uniswapV2Pair).sync() (src/yyds.sol#1008)
        - swapAndDividend(swapToken * 90 / 100) (src/yyds.sol#973)
                - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (src/yyds.sol#751)
                - (success,returndata) = target.call{value: value}(data) (src/yyds.sol#600)
                - uniswapV2Router.swapExactTokensForTokens(tokenAmount,0,path,address(tokenReceiver),block.timestamp) (src/yyds.sol#1053-1059)
                - IERC20(USDT).safeTransferFrom(address(tokenReceiver),marketingAddr,bToM) (src/yyds.sol#1062)
                - IERC20(USDT).safeTransferFrom(address(tokenReceiver),treasureAddr,usdtAmount - bToM) (src/yyds.sol#1063)
        External calls sending eth:
        - swapAndDividend(swapToken * 90 / 100) (src/yyds.sol#973)
                - (success,returndata) = target.call{value: value}(data) (src/yyds.sol#600)
        Event emitted after the call(s):
        - Transfer(sender,recipient,amount) (src/yyds.sol#385)
                - _tokenTransfer(from,to,amount,takeFee) (src/yyds.sol#984)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
YYDS.deflation() (src/yyds.sol#987-1010) uses timestamp for comparisons
        Dangerous comparisons:
        - lastDeflationTime != 0 && block.timestamp - lastDeflationTime >= 86400 (src/yyds.sol#988)
        - block.timestamp - startTime > intervalDays * 86400 (src/yyds.sol#1002)
YYDS._tokenTransfer(address,address,uint256,bool) (src/yyds.sol#1013-1044) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(startTime > 0,not time) (src/yyds.sol#1015)
        - require(bool,string)(block.timestamp - lastTimeTx[tx.origin] >= buyInterval,trading after a while) (src/yyds.sol#1019)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
```



```
INFO:Detectors:
Address._revert(bytes,string) (src/yyds.sol#696-708) uses assembly
        - INLINE ASM (src/yyds.sol#701-704)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Address.functionCall(address,bytes) (src/yyds.sol#554-556) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (src/yyds.sol#583-585) is never used and should be removed
Address.functionDelegateCall(address,bytes) (src/yyds.sol#635-637) is never used and should be removed
Address.functionDelegateCall(address,bytes,string) (src/yyds.sol#645-652) is never used and should be removed
Address.functionStaticCall(address,bytes) (src/yyds.sol#610-612) is never used and should be removed
Address.functionStaticCall(address,bytes,string) (src/yyds.sol#620-627) is never used and should be removed
Address.sendValue(address,uint256) (src/yyds.sol#529-534) is never used and should be removed
Address.verifyCallResult(bool,bytes,string) (src/yyds.sol#684-694) is never used and should be removed
Context._msgData() (src/yyds.sol#114-117) is never used and should be removed
ERC20._burn(address,uint256) (src/yyds.sol#418-426) is never used and should be removed
SafeERC20.safeTransfer(IERC20,address,uint256) (src/yyds.sol#714-716) is never used and should be removed
SafeMath.div(uint256,uint256) (src/yyds.sol#1182-1184) is never used and should be removed
SafeMath.div(uint256,uint256,string) (src/yyds.sol#1198-1204) is never used and should be removed
SafeMath.mod(uint256,uint256) (src/yyds.sol#1218-1220) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (src/yyds.sol#1234-1237) is never used and should be removed
SafeMath.mul(uint256,uint256) (src/yyds.sol#1156-1168) is never used and should be removed
SafeMath.sub(uint256,uint256) (src/yyds.sol#1125-1127) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version^0.8.0 (src/yyds.sol#2) allows old versions
solc-0.8.22 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in Address.sendValue(address,uint256) (src/yyds.sol#529-534):
        - (success) = recipient.call{value: amount}() (src/yyds.sol#532)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (src/yyds.sol#593-602):
        - (success,returndata) = target.call{value: value}(data) (src/yyds.sol#600)
Low level call in Address.functionStaticCall(address,bytes,string) (src/yyds.sol#620-627):
        - (success,returndata) = target.staticcall(data) (src/yyds.sol#625)
Low level call in Address.functionDelegateCall(address,bytes,string) (src/yyds.sol#645-652):
        - (success,returndata) = target.delegatecall(data) (src/yyds.sol#650)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Parameter YYDS.setVault(address)._vault (src/yyds.sol#852) is not in mixedCase
Parameter YYDS.setLimitedTrade(bool)._limitedTrade (src/yyds.sol#856) is not in mixedCase
Parameter YYDS.setBuyInterval(uint256)._buyInterval (src/yyds.sol#860) is not in mixedCase
Parameter YYDS.setMaxTokenVaulePerTx(uint256)._maxTokenVaulePerTx (src/yyds.sol#878) is not in mixedCase
Parameter YYDS.setMaxTokenVauleSellPerTx(uint256)._maxTokenVauleSellPerTx (src/yyds.sol#882) is not in mixedCase
Parameter YYDS.setIntervalDays(uint256)._intervalDays (src/yyds.sol#886) is not in mixedCase
Parameter YYDS.setNodes(address[])._nodes (src/yyds.sol#890) is not in mixedCase
Parameter YYDS.setBuyMarketingFee(uint256)._buyMarketingFee (src/yyds.sol#904) is not in mixedCase
Parameter YYDS.setMarketingWallet(address)._marketingAddr (src/yyds.sol#909) is not in mixedCase
Parameter YYDS.setShareholderAddr(address)._shareholderAddr (src/yyds.sol#913) is not in mixedCase
Parameter YYDS.setTreasureAddr(address)._treasureAddr (src/yyds.sol#917) is not in mixedCase
Parameter YYDS.setSellMarketingFee(uint256)._sellMarketingFee (src/yyds.sol#921) is not in mixedCase
Parameter YYDS.setMinAmountDeflation(uint256)._minAmountDeflation (src/yyds.sol#930) is not in mixedCase
Parameter YYDS.setMinAmountBurn(uint256)._minAmountBurn (src/yyds.sol#934) is not in mixedCase
Variable YYDS.VAULT (src/yyds.sol#781) is not in mixedCase
Function IUniswapV2Router01.WETH() (src/yyds.sol#1082) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Redundant expression "this (src/yyds.sol#115)" inContext (src/yyds.sol#109-118)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Detectors:
Loop condition `i < nodes.length` (src/yyds.sol#996) should use cached array length instead of referencing `length` member of the storage array.
 Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#cache-array-length
INFO:Detectors:
YYDS.tokenOwner (src/yyds.sol#780) should be immutable
YYDS.tokenReceiver (src/yyds.sol#771) should be immutable
YYDS.uniswapV2Pair (src/yyds.sol#770) should be immutable
YYDS.uniswapV2Router (src/yyds.sol#769) should be immutable
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-immutable
INFO:Slither:. analyzed (14 contracts with 88 detectors), 80 result(s) found
```

Closing Summary

Some issues of High, Medium, Low and informational severity were found, Some suggestions and best practices are also provided in order to improve the code quality and security posture.

Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in YYDS smart contracts. However, it is important to understand that no security audit can guarantee complete protection against all possible security threats. QuillAudits audit reports are based on the information provided to us at the time of the audit, and we cannot guarantee the accuracy or completeness of this information. Additionally, the security landscape is constantly evolving, and new security threats may emerge after the audit has been completed.

Therefore, it is recommended that multiple audits and bug bounty programs be conducted to ensure the ongoing security of YYDS smart contracts. One audit is not enough to guarantee complete protection against all possible security threats. It is important to implement proper risk management strategies and stay vigilant in monitoring your smart contracts for potential security risks.

QuillAudits cannot be held liable for any security breaches or losses that may occur subsequent to and despite using our audit services.. It is the responsibility of the YYDS to implement the recommendations provided in our audit reports and to take appropriate steps to mitigate potential security risks.

About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem.

850+
Audits Completed

$30B
Secured

$30B
Lines of Code Audited

Follow Our Journey

Audit Report
November, 2023

For

YYDS

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com