DividendToken

v0. 0.8.16+commit.7dd6d404

v0.8.16

✦Low-Risk	✦Medium-Risk	✦High-Risk
Low-risk code	Medium-risk code	High-risk code

Contract Address

0x52f0802af4E396b998783D51CAD0751a0befFE7f

[Disclaimer]

AuditBlock is not liable for any financial losses incurred due to its services. The information provided in this contract audit should not be considered financial advice. Please conduct your research to make informed decisions.

Types of Severities

High

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

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.

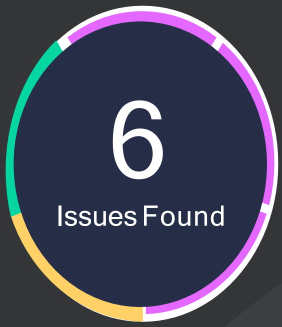
Low

Low-level severity issues can cause minor impact and or 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 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.

Name	DividendToken
Method	Manual Review, Functional Testing, Automated Testing etc.
Scope of Audit	The scope of this audit was to analyze the contract codebase for quality, security, and correctness.
Audit Team	AuditBlock



🟥 High

🟨 Medium

🟪 Low

🟩 Informational

	High	Medium	Low	Informational
Open Issues	4	1	0	0
Acknowledged Issues	0	2	0	0
Partially Resolved Issues	0	0	0	0
Resolved Issues	0	0	0	0

ID	File Name	Audit Score
10016	DividendToken.sol	40%

Smart Contract Weakness Classification (SWC) Vulnerabilities for Attacks

- ✗ Re-entrancy
- ✓ Timestamp Dependence
- ✓ Gas Limit and Loops
- ✓ Exception Disorder
- ✓ Gasless Send
- ✓ Use of tx.origin
- ✗ Compiler version not fixed
- ✗ Address hardcoded
- ✗ Divide before multiply
- ✓ Integer overflow/underflow
- ✗ Dangerous strict equalities

- ✓ Tautology or contradiction
- ✓ Missing Zero Address Validation
- ✗ Return values of low-level calls
- ✓ Revert/require functions
- ✓ Private modifier
- ✓ Using block.timestamp
- ✗ Multiple Sends
- ✓ Using SHA3
- ✓ Using suicide
- ✓ Using throw
- ✓ Using inline assembly

Techniques and Methods

The overall quality of code.

- Use of best practices.
- Code documentation and comments match logic and expected behavior.
- Token distribution and calculations are as per the intended behavior mentioned in the whitepaper.
- implementation of ERC-20 token standards.
- Efficient use of gas.
- Code is safe from re-entrance 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

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, and 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.

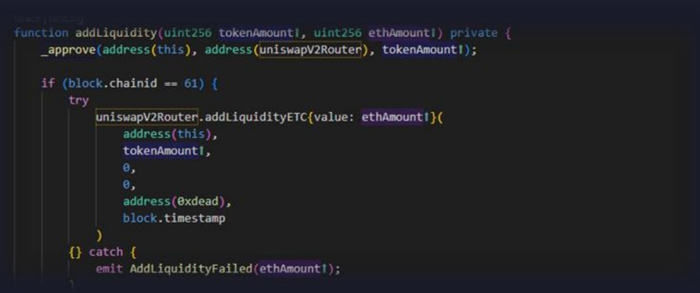
Tools and Platforms Used for Audit

Remix IDE, Truffle, Truffle Team, Solhint, Mythril, Slither, Solidity statistic analysis.

Phase 1

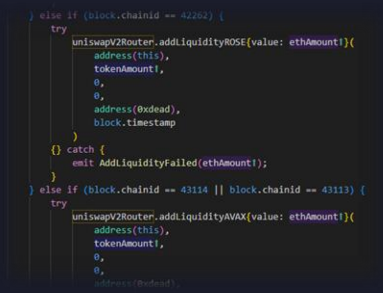
High Severity Issues

1. Dangerous calls.



```
function addLiquidity(uint256 tokenAmount, uint256 ethAmount) private {
    _approve(address(this), address(uniswapV2Router), tokenAmount);

    if (block.chainid == 61) {
        try
            uniswapV2Router.addLiquidityETC{value: ethAmount}(
                address(this),
                tokenAmount,
                0,
                0,
                address(0xdead),
                block.timestamp
            )
        {} catch {
            emit AddLiquidityFailed(ethAmount);
```



```
} else if (block.chainid == 42262) {
    try
        uniswapV2Router.addLiquidityROSE{value: ethAmount}(
            address(this),
            tokenAmount,
            0,
            0,
            address(0xdead),
            block.timestamp
        )
    {} catch {
        emit AddLiquidityFailed(ethAmount);
    }
} else if (block.chainid == 43114 || block.chainid == 43113) {
    try
        uniswapV2Router.addLiquidityAVAX{value: ethAmount}(
            address(this),
            tokenAmount,
            0,
            0,
            address(0xdead),
```

Description

It is a high-severity issue. Our auditor found Dangerous calls inside the [addLiquidity] line of code (#1897-1953). We acknowledge that this function uses an Unprotected call to a function sending Ether to an arbitrary address.

Recommendation

It is important to note that. You can double-check your function usability and how it's working with different behavior. We recommend that you Ensure that an arbitrary user cannot withdraw unauthorized funds.

Status

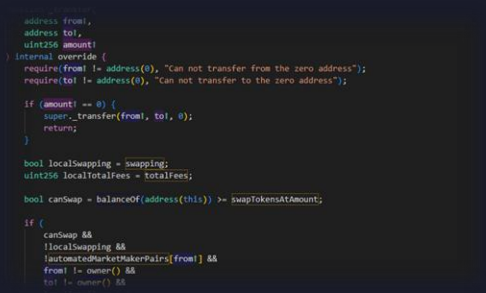
High

2. Reentrancy [Reentrancy in DividendToken._transfer]

Description

It is a high-severity issue. During an audit, a reentrancy bug was identified within the following methods:

- _transfer
- addLiquidityETC
- swapExactTokensForETCSupportingFeeOnTransferTokens
- addLiquidityROSE
- distributeCAKEDividends
- addLiquidityAVAX
- super._transfer
- ERC20._mint
- ERC20._transfe



```
    address from,
    address to,
    uint256 amount
) internal override {
    require(from != address(0), "Can not transfer from the zero address");
    require(to != address(0), "Can not transfer to the zero address");

    if (amount == 0) {
        super._transfer(from, to, 0);
        return;
    }

    bool localSwapping = swapping;
    uint256 localTotalFees = totalFees;

    bool canSwap = balanceOf(address(this)) >= swapTokensAtAmount;

    if (
        canSwap &&
        !localSwapping &&
        !automatedMarketMakerPairs[from] &&
        from != owner() &&
        to != owner() &&
```

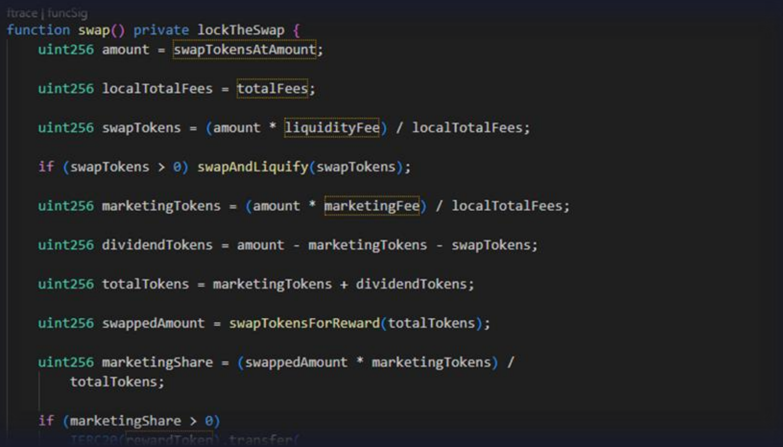
We acknowledge the reentrancy vulnerability within these functions and are reviewing the existing detection mechanisms.

Recommendation

It is important to note that. You can double-check your function usability and how it's working with different behavior. We recommend To avoid re-entrancy, you can use the Checks-Effects-Interactions pattern.
Status

High

3. Unchecked transfer



```
ftrace | funcSig
function swap() private lockTheSwap {
    uint256 amount = swapTokensAtAmount;

    uint256 localTotalFees = totalFees;

    uint256 swapTokens = (amount * liquidityFee) / localTotalFees;

    if (swapTokens > 0) swapAndLiquify(swapTokens);

    uint256 marketingTokens = (amount * marketingFee) / localTotalFees;

    uint256 dividendTokens = amount - marketingTokens - swapTokens;

    uint256 totalTokens = marketingTokens + dividendTokens;

    uint256 swappedAmount = swapTokensForReward(totalTokens);

    uint256 marketingShare = (swappedAmount * marketingTokens) /
        totalTokens;

    if (marketingShare > 0)
        IERC20(rewardToken).transfer(
```

Description

It is a high-severity issue. Our auditor found an Unchecked transfer inside the [swap] method. line of code (#1786-1813). We acknowledge that this function uses a The return value of an external transfer/transferFrom call is not checked

Recommendation

It is important to note that. You can double-check your function usability and how it’s working with different behavior. We recommend that to Use SafeERC20 or ensure that the transfer/transferFrom return value is checked.

Status

High

Medium Severity Issues

divide-before-multiply, Unused-return, dangerous-strict-equalities

Description

This is a medium-severity issue. Our auditor found these issues multiple times in different methods across several lines of code (#1786-1813, #1815-1830, #1651-1668). We acknowledge the need for a code review.

Recommendation

It's important to note that you can double-check your function's usability and how it's working with different behaviors. We recommend using OpenZeppelin contracts.

Status

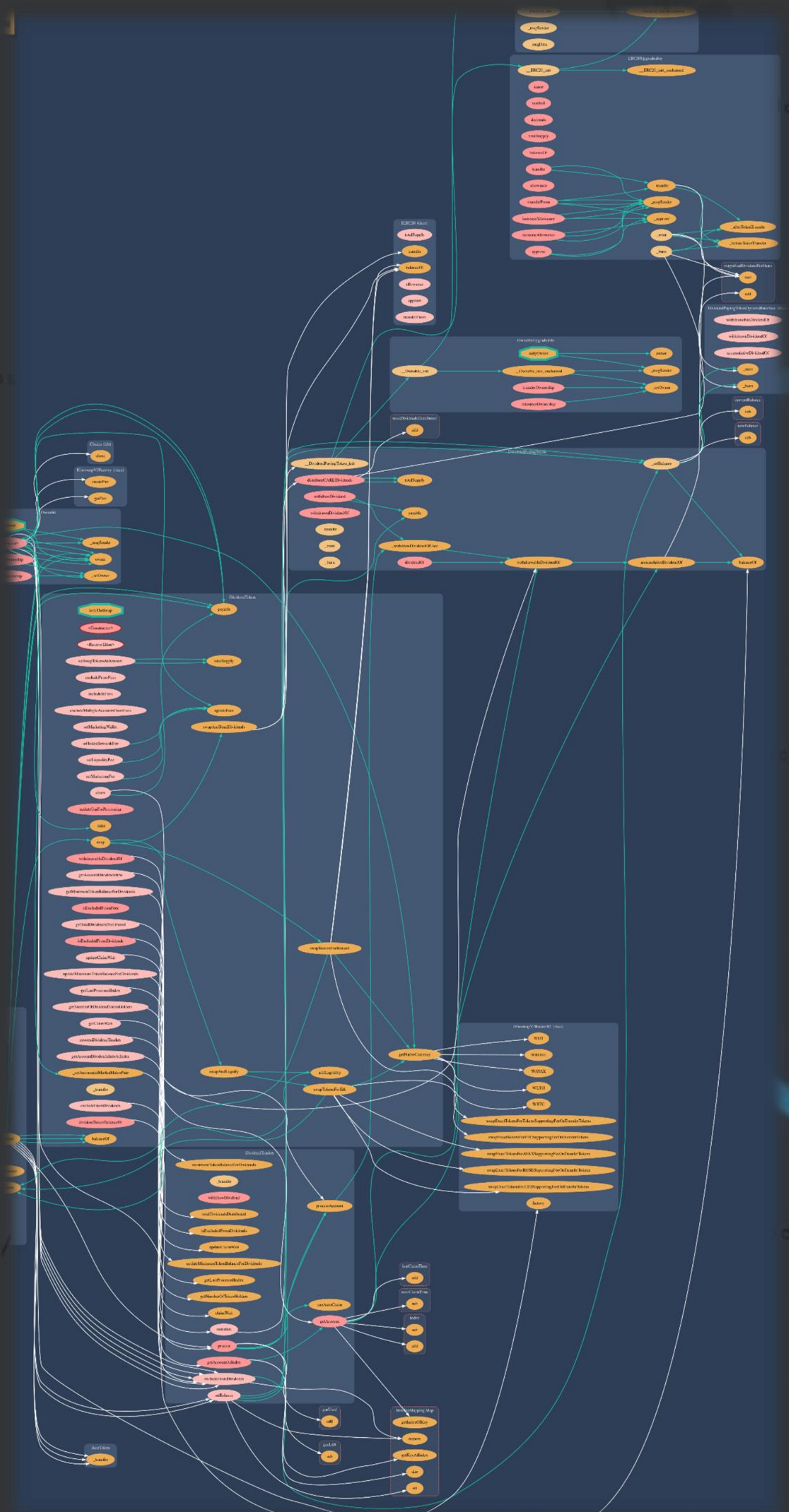
Medium

Low Severity Issues

No issues found

Informational Severity Issues

No issues found



10016 - DividendToken

Phase 2

```
DividendToken.addLiquidity(uint256,uint256) (contracts/DividendToken.sol#1897-1953) sends eth
to arbitrary user
	Dangerous calls:
	- uniswapV2Router.addLiquidityETC{value:
ethAmount}(address(this),tokenAmount,0,0,address(0xdead),block.timestamp)
(contracts/DividendToken.sol#1901-1912)
	- uniswapV2Router.addLiquidityROSE{value:
ethAmount}(address(this),tokenAmount,0,0,address(0xdead),block.timestamp)
(contracts/DividendToken.sol#1914-1925)
	- uniswapV2Router.addLiquidityAVAX{value:
ethAmount}(address(this),tokenAmount,0,0,address(0xdead),block.timestamp)
(contracts/DividendToken.sol#1927-1938)
	- uniswapV2Router.addLiquidityETH{value:
ethAmount}(address(this),tokenAmount,0,0,address(0xdead),block.timestamp)
(contracts/DividendToken.sol#1940-1951)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-
to-arbitrary-destinations


Reentrancy in DividendToken._transfer(address,address,uint256)
(contracts/DividendToken.sol#1717-1784):
	External calls:
	- swap() (contracts/DividendToken.sol#1743)
		- uniswapV2Router.addLiquidityETC{value:
ethAmount}(address(this),tokenAmount,0,0,address(0xdead),block.timestamp)
(contracts/DividendToken.sol#1901-1912)
		- success = localRewardToken.transfer(address(dividendTracker),dividends)
(contracts/DividendToken.sol#1993-1996)
		-
uniswapV2Router.swapExactTokensForETCSupportingFeeOnTransferTokens(tokenAmount,0,path
,address(this),block.timestamp) (contracts/DividendToken.sol#1841-1852)
		- uniswapV2Router.addLiquidityROSE{value:  (contracts/DividendToken.sol#1940-1951)
	State variables written after the call(s):
	- super._transfer(from,address(this),fees) (contracts/DividendToken.sol#1756)
		- _balances[sender] = senderBalance - amount (contracts/DividendToken.sol#1087)
		- _balances[recipient] += amount (contracts/DividendToken.sol#1089)
	ERC20._balances (contracts/DividendToken.sol#966) can be used in cross function
reentrancies:
	- ERC20._mint(address,uint256) (contracts/DividendToken.sol#1094-1100)
	- ERC20._transfer(address,address,uint256) (contracts/DividendToken.sol#1073-1092)
	- ERC20.balanceOf(address) (contracts/DividendToken.sol#997-1001)
	- super._transfer(from,to,amount) (contracts/DividendToken.sol#1759)
		- _balances[sender] = senderBalance - amount (contracts/DividendToken.sol#1087)
		- _balances[recipient] += amount (contracts/DividendToken.sol#1089)
	ERC20._balances (contracts/DividendToken.sol#966) can be used in cross function
reentrancies:
	- ERC20._mint(address,uint256) (contracts/DividendToken.sol#1094-1100)
	- ERC20._transfer(address,address,uint256) (contracts/DividendToken.sol#1073-1092)
	- ERC20.balanceOf(address) (contracts/DividendToken.sol#997-1001)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities
```

```
DividendToken.swap() (contracts/DividendToken.sol#1786-1813) ignores return value by
IERC20(rewardToken).transfer(marketingWalletAddress,marketingShare)
(contracts/DividendToken.sol#1807-1810)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer

DividendToken.swap() (contracts/DividendToken.sol#1786-1813) performs a multiplication on the
result of a division:
	- marketingTokens = (amount * marketingFee) / localTotalFees
(contracts/DividendToken.sol#1795)
	- marketingShare = (swappedAmount * marketingTokens) / totalTokens
(contracts/DividendToken.sol#1803-1804)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply


DividendToken.swapAndLiquify(uint256) (contracts/DividendToken.sol#1815-1830) uses a
dangerous strict equality:
	- newBalance == 0 (contracts/DividendToken.sol#1825)
DividendToken.swapAndSendDividends() (contracts/DividendToken.sol#1986-2002) uses a
dangerous strict equality:
	- dividends == 0 (contracts/DividendToken.sol#1991)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities

Reentrancy in DividendTracker.process(uint256) (contracts/DividendToken.sol#852-897):
	External calls:
	- processAccount(address(account),true) (contracts/DividendToken.sol#878)
		- success = IERC20(rewardToken).transfer(user,_withdrawableDividend)
(contracts/DividendToken.sol#508-511)
	State variables written after the call(s):
	- lastProcessedIndex = _lastProcessedIndex (contracts/DividendToken.sol#894)
	DividendTracker.lastProcessedIndex (contracts/DividendToken.sol#669) can be used in cross
function reentrancies:
	- DividendTracker.getAccount(address) (contracts/DividendToken.sol#751-800)
	- DividendTracker.getLastProcessedIndex() (contracts/DividendToken.sol#743-745)
	- DividendTracker.lastProcessedIndex (contracts/DividendToken.sol#669)
	- DividendTracker.process(uint256) (contracts/DividendToken.sol#852-897)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-
1
DividendToken.addLiquidity(uint256,uint256) (contracts/DividendToken.sol#1897-1953) ignores
return value by uniswapV2Router.addLiquidityETC{value:
ethAmount}(address(this),tokenAmount,0,0,address(0xdead),block.timestamp)
(contracts/DividendToken.sol#1901-1912)
DividendToken.addLiquidity(uint256,uint256) (contracts/DividendToken.sol#1897-1953) ignores
return value by uniswapV2Router.addLiquidityROSE{value:
ethAmount}(address(this),tokenAmount,0,0,address(0xdead),block.timestamp)
(contracts/DividendToken.sol#1914-1925)
DividendToken.addLiquidity(uint256,uint256) (contracts/DividendToken.sol#1897-1953) ignores
return value by uniswapV2Router.addLiquidityAVAX{value:
ethAmount}(address(this),tokenAmount,0,0,address(0xdead),block.timestamp)
(contracts/DividendToken.sol#1927-1938)
DividendToken.addLiquidity(uint256,uint256) (contracts/DividendToken.sol#1897-1953) ignores
return value by uniswapV2Router.addLiquidityETH{value:
ethAmount}(address(this),tokenAmount,0,0,address(0xdead),block.timestamp)
(contracts/DividendToken.sol#1940-1951)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
```

```
DividendPayingToken.__DividendPayingToken_init(address,string,string)._name
(contracts/DividendToken.sol#474) shadows:
        - ERC20Upgradeable._name (contracts/DividendToken.sol#188) (state variable)
DividendPayingToken.__DividendPayingToken_init(address,string,string)._symbol
(contracts/DividendToken.sol#475) shadows:
        - ERC20Upgradeable._symbol (contracts/DividendToken.sol#189) (state variable)
DividendPayingToken.dividendOf(address)._owner (contracts/DividendToken.sol#526) shadows:
        - OwnableUpgradeable._owner (contracts/DividendToken.sol#381) (state variable)
DividendPayingToken.withdrawableDividendOf(address)._owner (contracts/DividendToken.sol#531)
shadows:
        - OwnableUpgradeable._owner (contracts/DividendToken.sol#381) (state variable)
DividendPayingToken.withdrawnDividendOf(address)._owner (contracts/DividendToken.sol#537)
shadows:
        - OwnableUpgradeable._owner (contracts/DividendToken.sol#381) (state variable)
DividendPayingToken.accumulativeDividendOf(address)._owner (contracts/DividendToken.sol#543)
shadows:
        - OwnableUpgradeable._owner (contracts/DividendToken.sol#381) (state variable)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

DividendPayingToken._withdrawDividendOfUser(address) (contracts/DividendToken.sol#499-524) has
external calls inside a loop: success = IERC20(rewardToken).transfer(user,_withdrawableDividend)
(contracts/DividendToken.sol#508-511)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop

cts/DividendToken.sol#1901-1912)
                - uniswapV2Router.addLiquidityROSE{value:
ethAmount}(address(this),tokenAmount,0,0,address(0xdead),block.timestamp)
(contracts/DividendToken.sol#1914-1925)
                - uniswapV2Router.addLiquidityAVAX{value:
ethAmount}(address(this),tokenAmount,0,0,address(0xdead),block.timestamp)
(contracts/DividendToken.sol#1927-1938)
                - uniswapV2Router.addLiquidityETH{value:
ethAmount}(address(this),tokenAmount,0,0,address(0xdead),block.timestamp)
(contracts/DividendToken.sol#1940-1951)
        State variables written after the call(s):
        - addLiquidity(otherHalf,newBalance) (contracts/DividendToken.sol#1827)
                - _allowances[owner][spender] = amount (contracts/DividendToken.sol#1123)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2

DividendTracker.getAccount(address) (contracts/DividendToken.sol#751-800) uses timestamp for
comparisons
        Dangerous comparisons:
        - nextClaimTime > block.timestamp (contracts/DividendToken.sol#797-799)
DividendTracker.canAutoClaim(uint256) (contracts/DividendToken.sol#827-833) uses timestamp for
comparisons
        Dangerous comparisons:
        - lastClaimTime > block.timestamp (contracts/DividendToken.sol#828)
        - block.timestamp.sub(lastClaimTime) >= claimWait (contracts/DividendToken.sol#832)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

DividendToken.dividendTracker (contracts/DividendToken.sol#1330) should be immutable
DividendToken.rewardToken (contracts/DividendToken.sol#1332) should be immutable
DividendToken.uniswapV2Router (contracts/DividendToken.sol#1326) should be immutable
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-immutable
```

```
 warning  Found more than One contract per file. 25 contracts found!            one-contract-per-file
 32:13  warning  Use Custom Errors instead of require statements                 custom-errors
 43:13  warning  Use Custom Errors instead of require statements                 custom-errors
 56:9   warning  Provide an error message for require                          reason-string
 56:9   warning  Use Custom Errors instead of require statements                 custom-errors
 57:9   warning  Provide an error message for require                          reason-string
 57:9   warning  Use Custom Errors instead of require statements                 custom-errors
 62:9   warning  Provide an error message for require                          reason-string
 62:9   warning  Use Custom Errors instead of require statements                 custom-errors
 69:9   warning  Provide an error message for require                          reason-string
 69:9   warning  Use Custom Errors instead of require statements                 custom-errors
 75:9   warning  Provide an error message for require                          reason-string
 75:9   warning  Use Custom Errors instead of require statements                 custom-errors
 80:9   warning  Provide an error message for require                          reason-string
 80:9   warning  Use Custom Errors instead of require statements                 custom-errors
 88:9   warning  Provide an error message for require                          reason-string
 88:9   warning  Use Custom Errors instead of require statements                 custom-errors
 99:9   warning  Error message for require is too long: 46 counted / 32 allowed         reason-string
 99:9   warning  Use Custom Errors instead of require statements                 custom-errors
 119:5   warning  Function name must be in mixedCase                             func-name-mixedcase
 123:5   warning  Function name must be in mixedCase                             func-name-mixedcase
 123:62  warning  Code contains empty blocks                                   no-empty-blocks
 191:5   warning  Function name must be in mixedCase                             func-name-mixedcase
 199:5   warning  Function name must be in mixedCase                             func-name-mixedcase
 260:9   warning  Error message for require is too long: 40 counted / 32 allowed         reason-string
 260:9   warning  Use Custom Errors instead of require statements                 custom-errors
 288:9   warning  Error message for require is too long: 37 counted / 32 allowed         reason-string
 288:9   warning  Use Custom Errors instead of require statements                 custom-errors
 304:9   warning  Error message for require is too long: 37 counted / 32 allowed         reason-string
 304:9   warning  Use Custom Errors instead of require statements                 custom-errors
 305:9   warning  Error message for require is too long: 35 counted / 32 allowed         reason-string
 305:9   warning  Use Custom Errors instead of require statements                 custom-errors
 310:9   warning  Error message for require is too long: 38 counted / 32 allowed         reason-string
 310:9   warning  Use Custom Errors instead of require statements                 custom-errors
 325:9   warning  Use Custom Errors instead of require statements                 custom-errors
 337:9   warning  Error message for require is too long: 33 counted / 32 allowed         reason-string
 337:9   warning  Use Custom Errors instead of require statements                 custom-errors
 342:9   warning  Error message for require is too long: 34 counted / 32 allowed         reason-string
 342:9   warning  Use Custom Errors instead of require statements                 custom-errors
 358:9   warning  Error message for require is too long: 36 counted / 32 allowed         reason-string
 358:9   warning  Use Custom Errors instead of require statements                 custom-errors
 359:9   warning  Error message for require is too long: 34 counted / 32 allowed         reason-string
 359:9   warning  Use Custom Errors instead of require statements                 custom-errors
 369:24  warning  Code contains empty blocks                                   no-empty-blocks
 375:24  warning  Code contains empty blocks                                   no-empty-blocks
 388:5   warning  Function name must be in mixedCase                             func-name-mixedcase
 393:5   warning  Function name must be in mixedCase                             func-name-mixedcase
 402:9   warning  Use Custom Errors instead of require statements                 custom-errors
 411:9   warning  Error message for require is too long: 38 counted / 32 allowed         reason-string
 411:9   warning  Use Custom Errors instead of require statements                 custom-errors
 463:5   warning  Constant name must be in capitalized SNAKE_CASE                      const-name-snakecase
 472:5   warning  Function name must be in mixedCase                             func-name-mixedcase
 483:9   warning  Provide an error message for require                          reason-string
 483:9   warning  Use Custom Errors instead of require statements                 custom-errors
 514:17  warning  Possible reentrancy vulnerabilities. Avoid state changes after transfer  reentrancy
 558:9   warning  Provide an error message for require                          reason-string
 558:9   warning  Use Custom Errors instead of require statements                 custom-errors
 701:9   warning  Error message for require is too long: 41 counted / 32 allowed         reason-string
 701:9   warning  Use Custom Errors instead of require statements                 custom-errors
 705:9   warning  Error message for require is too long: 45 counted / 32 allowed         reason-string
 705:9   warning  Use Custom Errors instead of require statements                 custom-errors
 709:9   warning  Provide an error message for require                          reason-string
 709:9   warning  Use Custom Errors instead of require statements                 custom-errors
 725:9   warning  Error message for require is too long: 39 counted / 32 allowed         reason-string
 725:9   warning  Use Custom Errors instead of require statements                 custom-errors
 729:9   warning  Error message for require is too long: 44 counted / 32 allowed         reason-string
 729:9   warning  Use Custom Errors instead of require statements                 custom-errors
1410:9   warning  Error message for require is too long: 53 counted / 32 allowed         reason-string
 1936:13  warning  Code contains empty blocks                                   no-empty-blocks
 1949:13  warning  Code contains empty blocks                                   no-empty-blocks
```

Closing Summary

In this report, we have considered the security of this Dividend Token. We performed our audit according to the procedure described above.

Several issues were identified during the audit process, and their severity levels have been classified. Recommendations and best practices have also been provided to enhance code quality and security posture. The team has acknowledged all identified issues.

Disclaimer

AuditBlock does not provide security warranties, investment advice, or endorsements of any platform. This audit does not guarantee the security or correctness of the audited smart contracts. The statements made in this document should not be interpreted as investment or legal advice. The authors are not liable for any decisions made based on the information in this document. Securing smart contracts is an ongoing process. A single audit is not sufficient. We recommend that the platform's development team implement a bug bounty program to encourage further analysis of the smart contract by other third parties

AuditBlock

AuditBlock is a blockchain security company that provides professional services and solutions for securing blockchain projects. They specialize in smart contract audits on various blockchains and offer a range of services

30+
Audits Completed

$10K
Secured

100K
Lines of Code Audited

https://t.me/AuditBlock

https://github.com/AuditBlock

https://twitter.com/0AuditBlock