QuillAudits

JANUARY 2023

QUILL TEST

Security Audit

PRESENTED BY

Prosper Onah

Contents

Executive Summary

TYPES
Token/Stake, Auditing

METHODS
Manual Review, Static Analysis, Remix

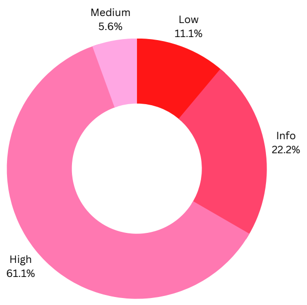
LANGUAGE
Solidity

TIMELINE
Delivered on 02/02/2023

REPOSITORY
https://github.com/Quillhash/Audit_mocks/blob/main/QuillTest.sol

Commit Hash
a80ec627d5152a39f674e5d0cb965b86df1fc678

Vulnerability Summary



18	0	18
Total Findings	Resolved	Unresolved

🟥	11	High	High risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
🟨	1	Medium	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform
🟦	2	Low	Minor risks do not compromise the overall integrity of the project, but they may be less efficient than other solutions.
🟦	4	Informational	Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code

Audit Goals and Focus

Verification of details

This will verify that every implementation on the contract follows the standard requirement of in smart contract development.

Verification of behavior

This will verify that the smart contract does not have any behavior, explicit or implicit that is not captured in any standard of smart contract development.
This audit will also verify that the contract does not violate the original intended behavior.

Smart Contract Best Practices

This audit evaluate whether the codebase follows the current established best practices for smart contract development.

Code Correctness

This audit will evaluate whether the code does what it is intended to do.

Code Quality

This audit will evaluate whether the code has been written in a way that ensures readability and maintainability.

Security

This audit will look for any exploitable security vulnerabilities, or other potential threats to stake holders.

Testing and testability

This audit will examine how easily tested the code is, and review how thoroughly tested the code is.

Audit Scope / Vulnerabilities Checked

The list of vulnerability checks conducted on the token contract includes but not limited to;

- Centralization of power.
- Timestamp Dependence.
- Exception Disorder.
- Compiler version not fixed.
- Address hardcoded.
- Divide before multiply.
- Integer overflow/underflow.
- Dangerous strict equalities.
- Missing Zero Address Validation.
- Revert/require functions.
- Using block.timestamp.
- Using block.number.
- Reentrancy.
- Access controls
- Arithmetic Issues (integers Overflow/Underflow).
- Unchecked return value for low level call.
- Unsafe external calls
- Business logic contradicting the specification
- Short Address Attack
- Unknown Vulnerability.

Automated Testing and Verification

I used automated testing techniques to enhance coverage of certain areas of the token contract.

● Slither, a Solidity static analysis framework. Slither can statically verify algebraic relationships between Solidity variables. I used Slither to detect invalid or inconsistent usage of the contracts' APIs across the entire codebase.

● Echidna, a smart contract fuzzer. Echidna can rapidly test security properties via malicious, coverage-guided test case generation. I used Echidna to test the expected system properties of the token contract and its libraries.

● Foundry is a smart contract development toolchain. Forge supports property-based testing.
Property-based testing is a way of testing general behaviors as opposed to isolated scenarios.
With large amounts of random input data, called "fuzz", in order to find bugs or vulnerabilities. The random input data generated by a fuzzer can be designed to exercise specific parts of a smart contract's code, such as error-handling routines, in ways that are difficult or impossible to achieve through manual testing.

While automated testing methods can enhance manual security evaluation, they cannot replace it completely. Each approach has its own limitations, for example, Slither may detect security features that do not hold up when translated into Solidity code, and Foundry limitations which include difficulties in accurately identifying all potential security properties, generating relevant test cases, or fully replicating the behavior of a contract. To mitigate these risks I generate 50,000 test cases per property with Echidna.

I evaluated 16 security properties across 2 main contracts. In the process, I formalized and tested a variety of properties, from high-level ones to very specific and low-level ones in basic libraries like SafeMath.sol and Address.sol

Regarding property coverage, the core of the contract, consisting of the token contract and its libraries, received substantial coverage.

The token contract contains the main business logic. It is the entry point for essential operations such as transfers, approvals, and staking platform. I identified security properties for each contract used to implement the token contract. Each property listed is valid regardless of the state, initialized or not, of the Exchange contract.

File: [contracts/Ownable.sol]

Property	Approach	Result
Owner can renounce ownership	Echidna	Failed
Owner can transfer ownership	Echidna	pass

Automated Testing and Verification

File: [contracts/QuillTest.sol]

Property	Approach	Result
Total supply must not exceed initial supply	Echidna	Failed
Assert that balance of owner reduce after token burn	Echidna	Failed
Only Owner can claim stuck token	Echidna	Pass
Only Owner can claim stuck ETH	Echidna	Pass
Transfer only when trade is enabled	Echidna	Pass
Trade when blacklist is enabled	Echidna	Failed
Transfer amount must not exceed balance	Echidna	Pass
Stop trade when amount is greater than Max wallet	Echidna	Pass
Owner can exclude from fee	Echidna	pass
Owner can update buy and sell	Echidna	pass
Owner can enable and disable swap	Echidna	pass
Owner can set swap token at an amount	Echidna	pass
Owner can enable maximum wallet limit	Echidna	pass
Owner can set maximum wallet amount	Echidna	pass
Owner can set maximum wallet amount to be total supply	Echidna	pass
Owner can exclude an account from max wallet	Echidna	pass
Owner can include an account to max wallet	Echidna	pass

Automated Testing and Verification

File: [contracts/QuillTest.sol]

Property	Approach	Result
Owner can enable maximum transaction limit	Echidna	pass
Owner can exclude an account from maximum transaction limit	Echidna	pass
Owner can set maximum transaction limit	Echidna	pass
Transfer zero amount and ensure user balance never change	Echidna	Failed
Can't Transfer max when trade is enabled	Echidna	Pass
Traders can stake and withdraw arbitrary token	Echidna	Pass
Fee always debited when trader withdraw stake	Echidna	Pass
Preserve staker previous stake amount on next stake	Echidna	Failed
Staker can empty the contract funds via multi claim	Echidna	Pass
Staker can withdraw rewards from contract	Echidna	Pass
Staker can redeem rewards from contract	Echidna	Pass

Summary of Findings

ID	TITLE	SEVERITY	STATUS
QUI-01	Floating pragma	Informational	Unresolved
QUI-02	Missing event emission	Informational	Unresolved
QUI-03	Public to External visibility	Informational	Unresolved
QUI-04	Unused Variable Declared	Informational	Unresolved
QUI-05	Ownership Change Risk	High	Unresolved
QUI-06	Unsafe Calculation	High	Unresolved
QUI-07	Unclear MarketingWallet Address	High	Unresolved
QUI-08	Spending Limit Restrictions	High	Unresolved
QUI-09	Owner Spending Limit Restrictions	High	Unresolved
QUI-10	Faulty Check Blacklist Function	High	Unresolved
QUI-11	Wrong max wallet Amount Validation	High	Unresolved
QUI-12	Wrong max transactionAmount Validation Check	High	Unresolved
QUI-13	Staking Vulnerability	High	Unresolved
QUI-14	Unused calculateFee Function	Medium	Unresolved
QUI-15	Malicious token withdrawal	High	Unresolved
QUI-16	Reentrancy attack on claim reward	High	Unresolved
QUI-17	Unused Event	Low	Unresolved
QUI-18	Save Deployment Fee	Low	Unresolved

Findings

ID	TITLE	SEVERITY	STATUS
QUI-01	Floating Pragma	Information	Unresolved

Unlock Pragma

Description

The contract makes use of the floating-point pragma 0.8.0. Contracts should be deployed using the same compiler version and flags that were used during the testing process. The use of a floating pragma increases the risk of encountering compatibility issues with different versions of the compiler, as well as with future upgrades to the compiler. This can result in unintended behaviour or vulnerabilities in the contract. Locking the pragma helps ensure that contracts are not unintentionally deployed using another pragma.

```
pragma solidity ^0.8.17;
```

Recommendations

Avoid the use of a floating pragma is to specify a fixed pragma version that corresponds to the version of the compiler being used. This ensures that the contract is compiled using a known and tested version of the compiler, reducing the risk of compatibility issues and vulnerabilities. Additionally, it is good practice to regularly upgrade the fixed pragma version to take advantage of bug fixes and security upgrades in the compiler.

```
pragma solidity 0.8.17;
```

Findings

ID	TITLE	SEVERITY	STATUS
QUI-02	Missing event emission	information	Unresolved

Missing event emission
File: [contracts/QuillTest.sol]

Description

write functions that change the state of the contract are required to emit an event, in other for off-chain use case.

Recommendations

It is recommended to add an event in the following functions: claimRewards, changeRewardToken, withdraw, stake, excludeFromMaxTransactionLimit, setMaxTransactionAmounts, setEnableMaxTransactionLimit, excludeFromMaxWallet, setMaxWalletAmount, setEnableMaxWalletLimit, excludeFromFees, updateBuyFees, updateSellFees, enableTrading, setBlackListEnabled, addBlacklist, setSwapEnabled, setSwapTokensAtAmount, function to notify off-chain clients of any state changes made in the contract. This can be achieved by using the emit keyword in the function to trigger an event and pass along relevant information to off-chain clients.

Findings

ID	TITLE	SEVERITY	STATUS
QUI-03	Public to External visibility	information	Unresolved

Public to External visibility
File: [contracts/QuillTest.sol]

Description

Public functions that do not need perform an operation within the state of the contract are not gas effective compare to declaring them as an external functions.

Recommendations

It is recommended to change the visibility of functions that do not need perform an operation within the state of the contract and just need to be called directly by end-users to external, as it help to reduce gas and also improve code readability.
the following functions needs to set to external:
setBlackListEnabled, addBlacklist, claimRewards

ID	TITLE	SEVERITY	STATUS
QUI-04	Unused Variable Declared	Informational	Unresolved

unused variable declared
File(s): [contracts/QuillTest.sol]

Description

The contract contains a custom error named "zeroAddress()" which is intended to ensure that the zero address is not used in a specific function, and a mapping of "balances", and "maxFee". However, this function is declared but never used in the code.

```
error zeroAddress();

mapping(address => uint256) public balances;

uint256 private maxFee;
```

The unused custom error and mapping may create confusion for future developers as it suggests that it is intended to be used in the code, but it is not. It may also indicate that a function is intended to use this custom error, and mapping is intended to store some user value, but not used, which could impact the security, performance, or functionality of the contract.

Recommendations

Remove the unused custom error "zeroAddress()" and mapping "balances" from the code to avoid confusion and maintain a clean codebase if it not meant to be use.

Findings

ID	TITLE	SEVERITY	STATUS
QUI-05	Ownership Change Risk	High	Unresolved

RenounceOwnership doesn't change ownership state of the contract
File(s): [contracts/Ownable.sol]

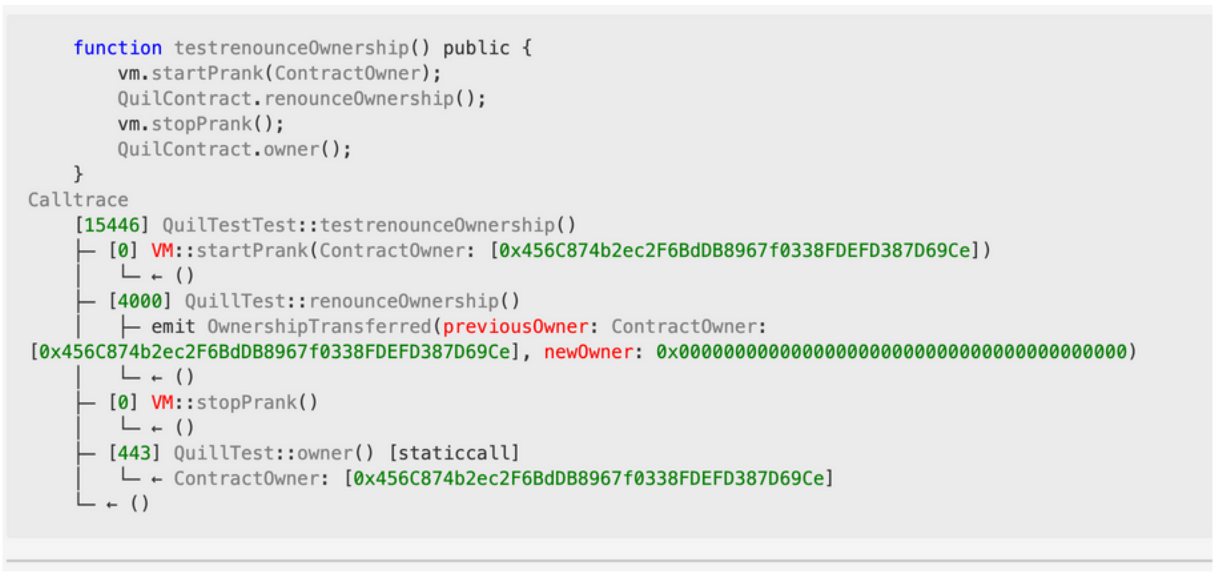
Description

The function below

```
function renounceOwnership() public virtual onlyOwner {
    emit OwnershipTransferred(_owner, address(0));
}
```

The intended purpose is to change the state of the contract and transfer the ownership of the contract to address(0) and notify other contracts or external entities of the change in ownership. However, the current implementation of the function only emits an event to notify of the change in ownership and doesn't change the state of the contract and the ownership variable.
This means that the contract's ownership variable will not be updated to address(0) and the contract will continue to recognize the current owner as the owner.

POC



```
    function testrenounceOwnership() public {
        vm.startPrank(ContractOwner);
        QuilContract.renounceOwnership();
        vm.stopPrank();
        QuilContract.owner();
    }
Calltrace
    [15446] QuilTestTest::testrenounceOwnership()
    ├─ [0] VM::startPrank(ContractOwner: [0x456C874b2ec2F6BdDB8967f0338FDEFD387D69Ce])
    │   └─ ← ()
    ├─ [4000] QuillTest::renounceOwnership()
    │   ├─ emit OwnershipTransferred(previousOwner: ContractOwner:
[0x456C874b2ec2F6BdDB8967f0338FDEFD387D69Ce], newOwner: 0x0000000000000000000000000000000000000000)
    │   └─ ← ()
    ├─ [0] VM::stopPrank()
    │   └─ ← ()
    ├─ [443] QuillTest::owner() [staticcall]
    │   └─ ← ContractOwner: [0x456C874b2ec2F6BdDB8967f0338FDEFD387D69Ce]
    └─ ← ()
```

Recommendations

Just as the function name implies Renouncing ownership should leave the contract without an owner, thereby removing any functionality that is only available to the owner.

1. First recommendation: it’s important to use open-source libraries like OpenZeppelin to ensure that your contracts are secure and reliable in other to ensure it intended purpose.
2. Kindly update renounceOwnership function_transferOwnership within it function instead of event emission

```
function renounceOwnership() public virtual onlyOwner {
      _transferOwnership(address(0));
  }
```

ID	TITLE	SEVERITY	STATUS
QUI-06	Unsafe Calculation	High	Unresolved

Unsafe calculation in safeMath
File: [SafeMath.sol]

Description

During the Returns of subtraction for two unsigned integers in other to ensure that Subtraction doesn't overflow, the function to subtract value b from a checks when b == 0 and return ~uint256(0);

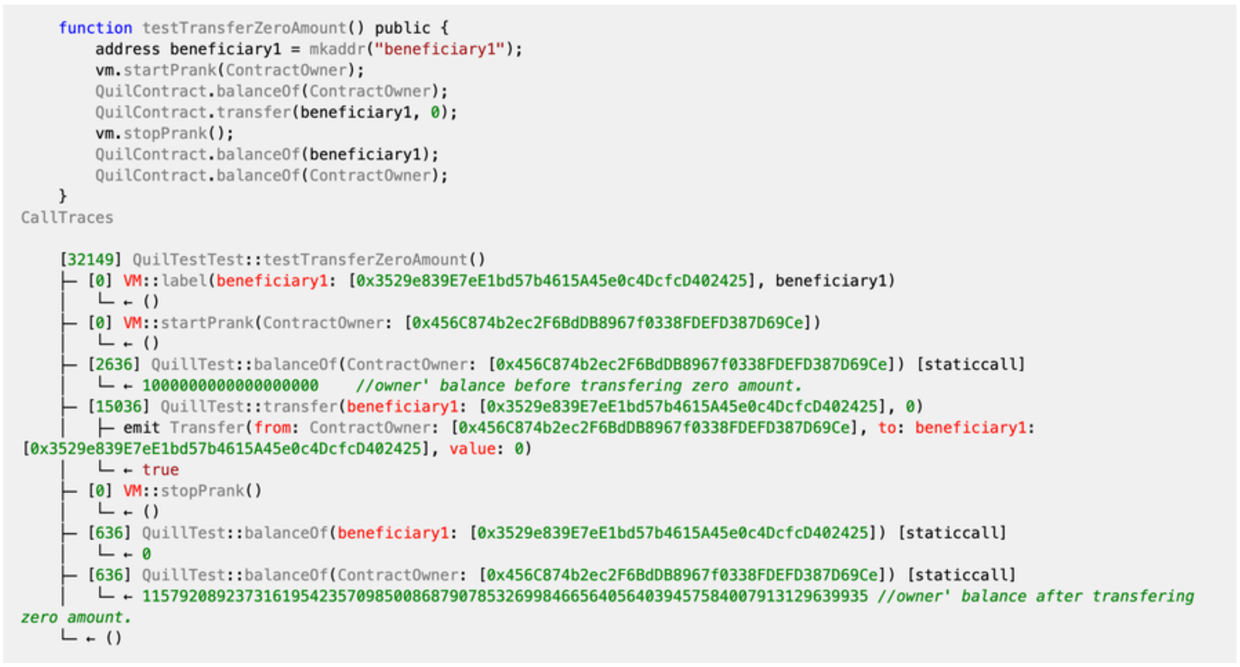
```
function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
    if(b == 0) {
        return ~uint256(0);
    }

    require(b <= a, errorMessage);
    uint256 c = a - b;

    return c;
}
```

An attacker could output the maximum value of uint256 115792089237316195423570985008687907853269984665640564039457584007913129639935 with the above code by passing zero as the second argument in the "sub" function. Since the first check in the internal function is

```
  if(b == 0) {
      return ~uint256(0);
  }
```

the function will return the maximum value of uint256 when b is zero.

POC

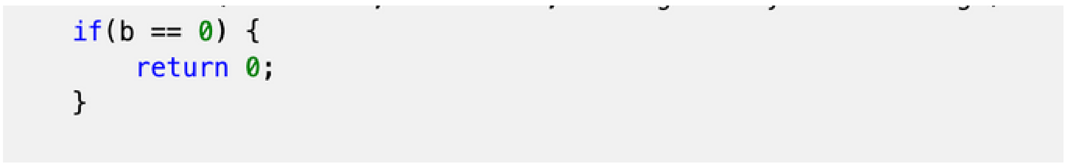


```
    function testTransferZeroAmount() public {
        address beneficiary1 = mkaddr("beneficiary1");
        vm.startPrank(ContractOwner);
        QuilContract.balanceOf(ContractOwner);
        QuilContract.transfer(beneficiary1, 0);
        vm.stopPrank();
        QuilContract.balanceOf(beneficiary1);
        QuilContract.balanceOf(ContractOwner);
    }
CallTraces

    [32149] QuilTestTest::testTransferZeroAmount()
    ├─ [0] VM::label(beneficiary1: [0x3529e839E7eE1bd57b4615A45e0c4DcfcD402425], beneficiary1)
    │   └─ ← ()
    ├─ [0] VM::startPrank(ContractOwner: [0x456C874b2ec2F6BdDB8967f0338FDEFD387D69Ce])
    │   └─ ← ()
    ├─ [2636] QuillTest::balanceOf(ContractOwner: [0x456C874b2ec2F6BdDB8967f0338FDEFD387D69Ce]) [staticcall]
    │   └─ ← 1000000000000000000     //owner' balance before transfering zero amount.
    ├─ [15036] QuillTest::transfer(beneficiary1: [0x3529e839E7eE1bd57b4615A45e0c4DcfcD402425], 0)
    │   ├─ emit Transfer(from: ContractOwner: [0x456C874b2ec2F6BdDB8967f0338FDEFD387D69Ce], to: beneficiary1:
[0x3529e839E7eE1bd57b4615A45e0c4DcfcD402425], value: 0)
    │   └─ ← true
    ├─ [0] VM::stopPrank()
    │   └─ ← ()
    ├─ [636] QuillTest::balanceOf(beneficiary1: [0x3529e839E7eE1bd57b4615A45e0c4DcfcD402425]) [staticcall]
    │   └─ ← 0
    ├─ [636] QuillTest::balanceOf(ContractOwner: [0x456C874b2ec2F6BdDB8967f0338FDEFD387D69Ce]) [staticcall]
    │   └─ ← 115792089237316195423570985008687907853269984665640564039457584007913129639935 //owner' balance after transfering
zero amount.
    └─ ← ()
```

Recommendation

To avoid such an attack,

1. First recommendation: it's important to use open-source libraries like OpenZeppelin to ensure that your contracts are secure and reliable in other to ensure it intended purpose. kindly import OpenZeppelin SafeMath.
2. remove the check for zero value of the second argument or change the return value to a more appropriate value such as zero, or another way to prevent such attack is to validate the input parameters before passing them to the function, this can be done by using the assert() or require() functions, to check that the input parameters are within the expected range.



```
    if(b == 0) {
        return 0;
    }
```

List of affected functions:

Kindly Note that the current implementation affected the following:

1. When using the transfer function: attacker can initiate a 0 transfer and due to the usage of the above _balances[sender] = _balances[sender].sub(amount, "ERC20: transfer amount exceeds balance"); sender balance will be updated with 115792089237316195423570985008687907853269984665640564039457584007913129639935.
2. owner can use the burn function to update it balance to uint256 max.
3. An attacker could use the approval function by just owning the smallest value of the token, then approve twice it other account to amount to uint256 max.

ID	TITLE	SEVERITY	STATUS
QUI-07	Unclear MarketingWallet Address	High	Unresolved

Casting MarketingWallet from bytes20 to uint160
File(s): [contracts/QuillTest.sol]

Description

The process used in casting an address in a specific way to assign a specific value to the variable "marketingWallet". The address is being cast from several types in order to assign a specific value to it, and Transferring ether to a randomly generated address can lead to the protocol stucking ether forever because no access to the private key will be provided when a random address is generated

```
address constant marketingWallet = address(uint160(bytes20("Your address")));
```

the marketingWallet uses an unknown string in a byte array to represent the address, it is not clear if an hexadecimal representation will be used during deployment.

Recommendations

The code is using a byte array to define the address, which can be a security risk if the byte array is not properly generated. Kindly ensure when computing this to an address, the generated address is control controlled by the marketing wallet. This can be mitigated by using a secure method of generating the byte array, such as using a cryptographic library to generate a random byte array.
Or better still use the following to ensure the right address is used for maketing wallet.

```
 // 0x012 should be repalced with marking wallet address
address constant marketingWallet = address(0x012..);
```

ID	TITLE	SEVERITY	STATUS
QUI-08	Spending Limit Restrictions	High	Unresolved

Restriction of maximum spending limit from dead addresses
File(s): [contracts/QuillTest.sol]

Description

Removing the restriction of maximum spending limit from dead addresses (address(0xdead)) could potentially allow an attacker to perform a denial of service attack by repeatedly sending transactions from a dead address,

```
constructor () ERC20("QuillTest", "QT") {
    ..............//
    ..............//
    _isExcludedFromMaxTxLimit[address(0xdead)] = true;
    ..............//
    ..............//
}
```

potentially exhausting the contract’s resources or causing it to malfunction.

Recommendations

Consider using the burn function to burn token instead

ID	TITLE	SEVERITY	STATUS
QUI-09	Owner Spending Limit Restrictions	High	Unresolved

Excluded the owner from maximum spending limit
File(s): [contracts/QuillTest.sol]

Description

the smart contract has a restriction of maximum spending limit for security reasons but it also excluded the owner from this restriction. This can be a significant security risk as it could lead to financial loss for the smart contract's users and stakeholders, as this could lead to potential conflicts of interest if the owner misuses their power for personal gain.

```
constructor () ERC20("QuillTest", "QT") {
    ..............//
    ..............//
    _isExcludedFromMaxTxLimit[address(owner())] = true;
    ..............//
    ..............//
}
```

Recommendations

It is recommended to re-evaluate the decision of excluding the owner from the maximum transaction limit and consider removing this exception if it poses a security risk, or it highly recommended to mitigate the risk and transfer the ownership of the contract to a multi-sig wallet. This would add an extra layer of security to the contract by requiring multiple parties to sign off on any transactions that exceed the maximum spending limit

ID	TITLE	SEVERITY	STATUS
QUI-10	Faulty Check Blacklist Function	High	Unresolved

Check Blacklist doesn’t check if an address is blacklisted
File(s): [contracts/QuillTest.sol]

Description

isBlacklisted function. When the blacklistEnabled flag is true, the function

```
function isBlacklisted(address _account) internal view returns (bool) {
    if(blacklistEnabled){
        {_isBlackListed[_account] == true;} return true;
    }else{
        return false;
    }
}
```

checks whether the address passed as an argument is blacklisted by looking up the _isBlackListed mapping. However, the way the check is written, it will always return true regardless of whether the address is blacklisted or not. The issue is that the check "_isBlackListed[_account] == true"; is in a code block, which doesn’t do anything and the line return true; is outside of this block, so it will always be executed.

POC

```
 function testBlackList() public {
    address BlackListedAddress = mkaddr("BlackListedAddress");
    address WhitelistAddress = mkaddr("WhiteListAddress");
    vm.prank(ContractOwner);
    QuilContract.addBlacklist(BlackListedAddress, true); //BlackListing an address
    //Adding a function (checkBlackList) to the contract to help check the status of addresses
    QuilContract.checkBlackList(BlackListedAddress);
    QuilContract.checkBlackList(WhitelistAddress);
}

[41436] QuilTestTest::testBlackList()
├─ [0] VM::label(BlackListedAddress: [0xFAe7907D6A7093DCFC68e2B0c4BDA77D43496515], BlackListedAddress)
│   └─ ← ()
├─ [0] VM::label(WhiteListAddress: [0xE2E3B172E505371a83a13B7236676d59454F020D], WhiteListAddress)
│   └─ ← ()
├─ [0] VM::prank(ContractOwner: [0x456C874b2ec2F6BdDB8967f0338FDEFD387D69Ce])
│   └─ ← ()
├─ [24826] QuillTest::addBlacklist(BlackListedAddress: [0xFAe7907D6A7093DCFC68e2B0c4BDA77D43496515], true)
│   └─ ← ()
├─ [2633] QuillTest::checkBlackList(BlackListedAddress: [0xFAe7907D6A7093DCFC68e2B0c4BDA77D43496515])
│   └─ ← false
├─ [633] QuillTest::checkBlackList(WhiteListAddress: [0xE2E3B172E505371a83a13B7236676d59454F020D])
│   └─ ← false
└─ ← ()
```

Recommendations

The check should be moved outside of the block and the return true; statement should be inside of the block and only be executed if the check is true.

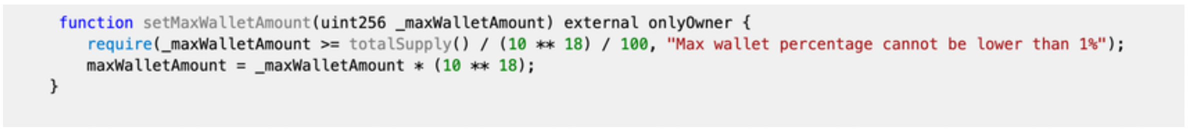
```
function isBlacklisted(address _account) internal view returns (bool) {
   return blacklistEnabled && _isBlackListed[_account];
}
```

ID	TITLE	SEVERITY	STATUS
QUI-11	Wrong max Wallet Amount Validation	High	Unresolved

SetMaxWalletAmount() wrong validation
File(s): [contracts/QuillTest.sol]

Description

The require statement in the setMaxWalletAmount()

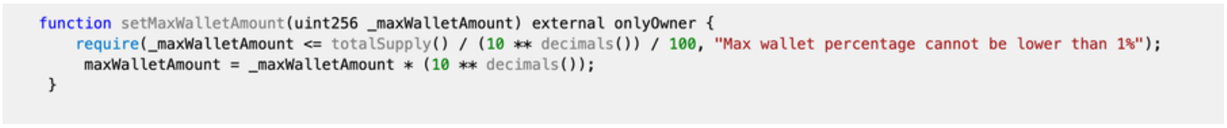


```
function setMaxWalletAmount(uint256 _maxWalletAmount) external onlyOwner {
    require(_maxWalletAmount >= totalSupply() / (10 ** 18) / 100, "Max wallet percentage cannot be lower than 1%");
    maxWalletAmount = _maxWalletAmount * (10 ** 18);
}
```

checks if the input value _maxWalletAmount is greater than or equal to 1% of the total supply. However, the calculation for 1% of the total supply is incorrect. the current calculation of totalSupply() / (10 ** 18) / 100, which is incorrect as it is dividing the total supply by 10^18 and then by 100, which results in 0. As a result, the check will always evaluate to true, regardless of the input value. This means that the owner could set a very high maximum wallet amount, potentially leading to a concentration of tokens in a single wallet and a potential loss of decentralization

Recommendations

To fix this issue, the comparison operator should be changed in the require statement to ensure that the maximum wallet amount is not greater than 1% of the total supply. Also the decimal of the token is set as 9 and _maxWalletAmount is been multiply by (10 ** 18) which will result in an unexpected value.



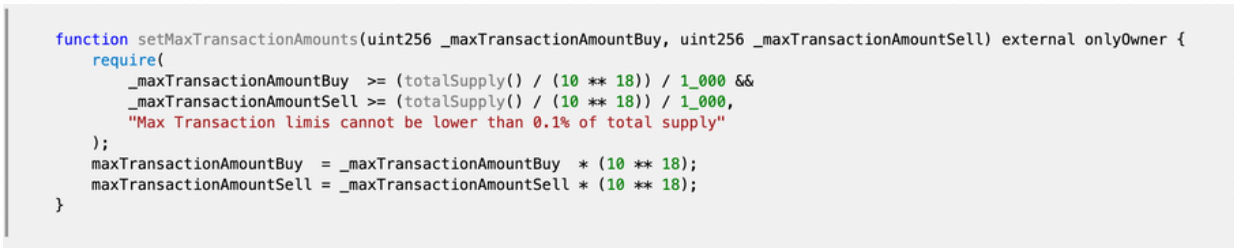
```
function setMaxWalletAmount(uint256 _maxWalletAmount) external onlyOwner {
    require(_maxWalletAmount <= totalSupply() / (10 ** decimals()) / 100, "Max wallet percentage cannot be lower than 1%");
    maxWalletAmount = _maxWalletAmount * (10 ** decimals());
}
```

ID	TITLE	SEVERITY	STATUS
QUI-12	Wrong transaction Amount Validation Check	High	Unresolved

Set maximum transaction amounts wrong validation check
File(s): [contracts/QuillTest.sol]

Description

The require statement in the setMaxTransactionAmounts function checks if the input value _maxTransactionAmountBuy, and _maxTransactionAmountSell is greater than or equal to 0.1% of the total supply.

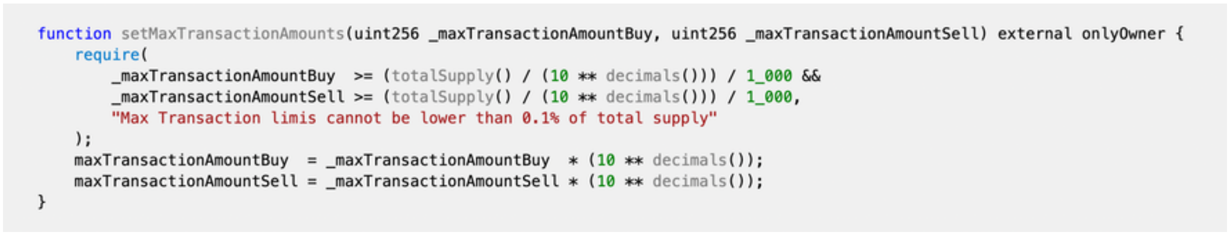


```
function setMaxTransactionAmounts(uint256 _maxTransactionAmountBuy, uint256 _maxTransactionAmountSell) external onlyOwner {
    require(
        _maxTransactionAmountBuy  >= (totalSupply() / (10 ** 18)) / 1_000 &&
        _maxTransactionAmountSell >= (totalSupply() / (10 ** 18)) / 1_000,
        "Max Transaction limis cannot be lower than 0.1% of total supply"
    );
    maxTransactionAmountBuy  = _maxTransactionAmountBuy  * (10 ** 18);
    maxTransactionAmountSell = _maxTransactionAmountSell * (10 ** 18);
}
```

However, the calculation for 0.1% of the total supply is incorrect, the current calculation of (totalSupply() / (10 ** 18)) / 1_000, which is incorrect as it is dividing the total supply by 10^18 and then by 1_000, which results in 0. As a result, the check will always evaluate to true, regardless of the input value. This means that the owner could set a very high maximum wallet amount, potentially leading to a concentration of tokens in a single wallet and a potential loss of decentralization.

Recommendations

The check on the requirement of the maximum limit should be corrected, and the expression should be modified to check the actual maximum limit for both _maxTransactionAmountBuy, and _maxTransactionAmountSell, rather than always returning 0.



```
function setMaxTransactionAmounts(uint256 _maxTransactionAmountBuy, uint256 _maxTransactionAmountSell) external onlyOwner {
    require(
        _maxTransactionAmountBuy  >= (totalSupply() / (10 ** decimals())) / 1_000 &&
        _maxTransactionAmountSell >= (totalSupply() / (10 ** decimals())) / 1_000,
        "Max Transaction limis cannot be lower than 0.1% of total supply"
    );
    maxTransactionAmountBuy  = _maxTransactionAmountBuy  * (10 ** decimals());
    maxTransactionAmountSell = _maxTransactionAmountSell * (10 ** decimals());
}
```

ID	TITLE	SEVERITY	STATUS
QUI-13	Staking Vulnerability	High	Unresolved

staking vulnerability
File(s): [contracts/QuillTest.sol]

Description

The function

```
function stake(uint256 amount, IERC20 token) external {
    if(amount == 0) revert zeroAmount();
    UserInfo memory user = UserInfo(amount, 0, 0, block.number, 0);
    userInfo[msg.sender] = user;
    IERC20(token).transferFrom(msg.sender, address(this), amount);
}
```

appears to be used for staking a certain amount of an unspecific token to the contract, with the following potential security risk:

- The function does not check if the passed token address is actually a valid required token address, so an attacker can deploy an arbitrary token and stake any any amount.
- An attempt for a user to stake multiple time will always override previous stake amount.
- An arbitrary amount of token can be stake.
- Another security risk is related to the use of block.number in the userInfo struct. Using the block number in this way can be problematic because it can be manipulated by the miner who mines the block. This could allow the miner to change the block number to any number they desire which could have unintended consequences.

Recommendations

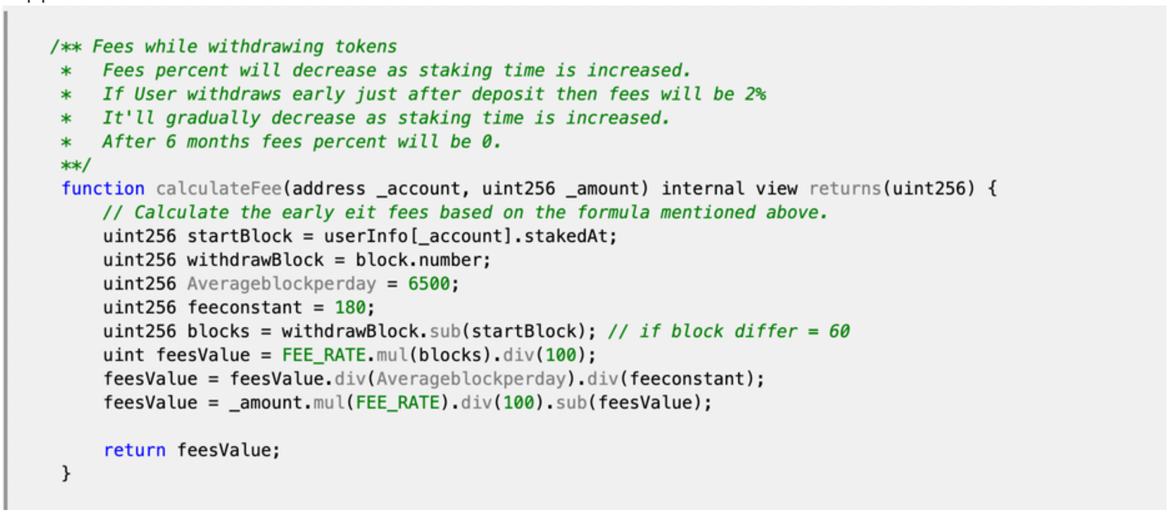
```
// mapping that store the states of tokens require
 // this assume all token price are treated the same during reward
 mapping(address => bool) private requireTokens;
 mapping(address => mapping(address => UserInfo)) private userInfo;
 // event to be emitted any time a token changes status
 event TokenStatus(address token, bool status, uint256 time);

 // custom error to revert when invalid token is provided
 error invalidToken();

 /// @notice setStakeToken is set to be used only by the owner to change the current stake of a token that will be use to stake
 /// @param _token address of the required token address that can be use to stake
 /// @param status state to change the status of the `_token` into
 function setStakeToken(address _token, bool status) external onlyOwner {
     require( requireTokens[_token] != status, "token is already set to that state");
     requireTokens[_token] = status;
     emit TokenStatus(_token, status, block.timestamp);
 }

 function stake(uint256 amount, IERC20 token) external {
     // revert if 0 amount is provided
     if(amount == 0) revert zeroAmount();
     // revert when invalid token address is provided
     if(!requireTokens[address(token)]) revert invalidToken();
     // get user current staked amount
     uint256 currentStaked = userInfo[msg.sender][address(token)].stakedAmount + amount;
     UserInfo memory user = UserInfo(currentStaked, 0, 0, block.timestamp, 0);
     userInfo[msg.sender][address(token)] = user;
     IERC20(token).transferFrom(msg.sender, address(this), amount);
 }

 /// @dev Returns the current state of tokenToCheck.
 /// will return true is it required for stake
 function checkTokenStatus(address tokenToCheck) external view returns(bool) {
     return requireTokens[tokenToCheck];
 }
```

ID	TITLE	SEVERITY	STATUS
QUI-14	Unused calculateFee Function	Medium	Unresolved

The unused return type of calculating fee
File(s): [contracts/QuillTest.sol]

Description

It appears that the function



```
/** Fees while withdrawing tokens
 *   Fees percent will decrease as staking time is increased.
 *   If User withdraws early just after deposit then fees will be 2%
 *   It'll gradually decrease as staking time is increased.
 *   After 6 months fees percent will be 0.
 **/
 function calculateFee(address _account, uint256 _amount) internal view returns(uint256) {
     // Calculate the early eit fees based on the formula mentioned above.
     uint256 startBlock = userInfo[_account].stakedAt;
     uint256 withdrawBlock = block.number;
     uint256 Averageblockperday = 6500;
     uint256 feeconstant = 180;
     uint256 blocks = withdrawBlock.sub(startBlock); // if block differ = 60
     uint feesValue = FEE_RATE.mul(blocks).div(100);
     feesValue = feesValue.div(Averageblockperday).div(feeconstant);
     feesValue = _amount.mul(FEE_RATE).div(100).sub(feesValue);

     return feesValue;
 }
```

is declared in the contract but is never used. The function takes in an address and an amount and returns a uint256, but the returned value is never used. This function calculates the fee for withdrawing tokens, with the fee percent decreasing as the staking time increases. However, this calculation is never applied to any withdrawals in the code.
This could indicate a potential issue in the contract as it appears that the fee calculation is not being applied to any withdrawals

Recommendations

It is recommended to review the contract to ensure that the fee calculation is being applied correctly and that the contract is functioning as intended.

ID	TITLE	SEVERITY	STATUS
QUI-15	Malicious token withdrawal	High	Unresolved

Withdraw rewards function vulnerable
File(s): [contracts/QuillTest.sol]

Description

```
function withdraw(IERC20 token, uint256 _amt) external {
    UserInfo storage user = userInfo[msg.sender];
    user.stakedAmount = user.stakedAmount - _amt;
    calculateFee(msg.sender, _amt);
    uint256 rewards = user.pendingRewardAmount;
    user.rewardAmount = rewards;

    uint256 fee = _amt.mul(100).div(FEE_RATE);

    user.pendingRewardAmount = 0;
    user.stakedAt = 0;

    IERC20(rewardToken).transfer(msg.sender, rewards);
    IERC20(token).transfer(msg.sender, _amt - fee);
}
```

An attacker may be able to withdraw any token from the contract by calling the function with a malicious token address. This is because the function doesn't check the address of the token being withdrawn against a whitelist or known set of allowed tokens, which could be used to prevent unauthorized tokens from being withdrawn. The return value calculateFee(msg.sender, _amt) is never used within the function

Recommendations

An attacker withdrawing any token from the contract is to check the address of the token being withdrawn against the token user deposited.

```
function withdraw(IERC20 token, uint256 _amt) external {
    UserInfo storage user = userInfo[msg.sender];
    user.stakedAmount = user.stakedAmount - _amt;
    calculateFee(msg.sender, _amt);
    uint256 rewards = user.pendingRewardAmount;
    user.rewardAmount = rewards;

    uint256 fee = _amt.mul(100).div(FEE_RATE);

    user.pendingRewardAmount = 0;
    user.stakedAt = 0;

    IERC20(rewardToken).transfer(msg.sender, rewards);
    IERC20(token).transfer(msg.sender, _amt - fee);
}
```

ID	TITLE	SEVERITY	STATUS
QUI-16	Reentrancy attack on claim reward	High	Unresolved

Claim Rewards function vulnerable
File(s): [contracts/QuillTest.sol]

Description

```
function claimRewards(address _account) public returns(uint256) {
    uint256 reward;
    UserInfo storage user = userInfo[_account];
    uint256 stakedAt = user.stakedAt;
    reward = userInfo[_account].stakedAmount.mul(block.number - stakedAt).div(100000);
    user.claimedAt = block.number;

    user.pendingRewardAmount = user.pendingRewardAmount.add(reward);

    return reward;
}
```

The reward calculation is based on the difference between the current block number and the block number when the user's stake was created. This means that an attacker could call this function multiple times with a short delay between calls to increase their rewards.
There is no check to ensure that the user's stake has not expired, so an attacker could call this function to claim rewards even after the stake has expired.
If an attacker can repeatedly call the function with the same stakedAt value before the block number has updated, they can cause the reward variable to be computed with the same block number difference each time, resulting in an accumulation of rewards that is not intended by the smart contract's design.
This is because the function doesn't check if the claim rewards already been claimed or not and also doesn't have any restriction on the frequency of the function call.

Recommendations

In other to mitigate this attack, a few recommendations would be:
Implement a delay between calls to the claimRewards function to ensure that the block number has updated before the function can be called again.
Use a timestamp instead of block number to calculate the rewards, as timestamps are less susceptible to manipulation.

```
function claimRewards(address _account, address token) public returns(uint256) {
    uint256 reward;
    UserInfo storage user = userInfo[_account];
    // get user last claim time
    uint256 lastClaimTime = user.claimedAt;
    // verify the time different between claim time
    if (block.timestamp < lastClaimTime + 7 days) revert Patience();
    reward = userInfo[_account][token].stakedAmount.mul(block.timestamp - lastClaimTime).div(100000);
    user.claimedAt = block.timestamp;

    user.pendingRewardAmount = user.pendingRewardAmount.add(reward);

    return reward;
}
```

ID	TITLE	SEVERITY	STATUS
QUI-17	Unused Event	Low	Unresolved

Unused declared event
File(s): [contracts/QuillTest.sol]

Description

The smart contract has an event called ExcludedFromMaxTransactionLimit, MaxTransactionLimitStateChanged, and MaxTransactionLimitAmountChanged, that is declared but never used in the code.

```
event ExcludedFromMaxTransactionLimit(address indexed account, bool isExcluded);
event MaxTransactionLimitStateChanged(bool maxTransactionLimit);
event MaxTransactionLimitAmountChanged(uint256 maxTransactionAmountBuy, uint256 maxTransactionAmountSell);
```

This event is intended to track when an account is excluded from the maximum transaction limit, when maximum transaction limit state changes, and Maximum Transaction Limit Amount for buy and sell changes, but it is never triggered. The unused event does not affect the functionality of the smart contract, but it does add unnecessary complexity to the code and increases the deployment cost.

Recommendations

it should be implemented in the appropriate function, such as the excludeFromMaxTransactionLimit(), setEnableMaxTransactionLimit(), and setMaxTransactionAmounts().

ID	TITLE	SEVERITY	STATUS
QUI-18	Save Deployment Fee	Low	Unresolved

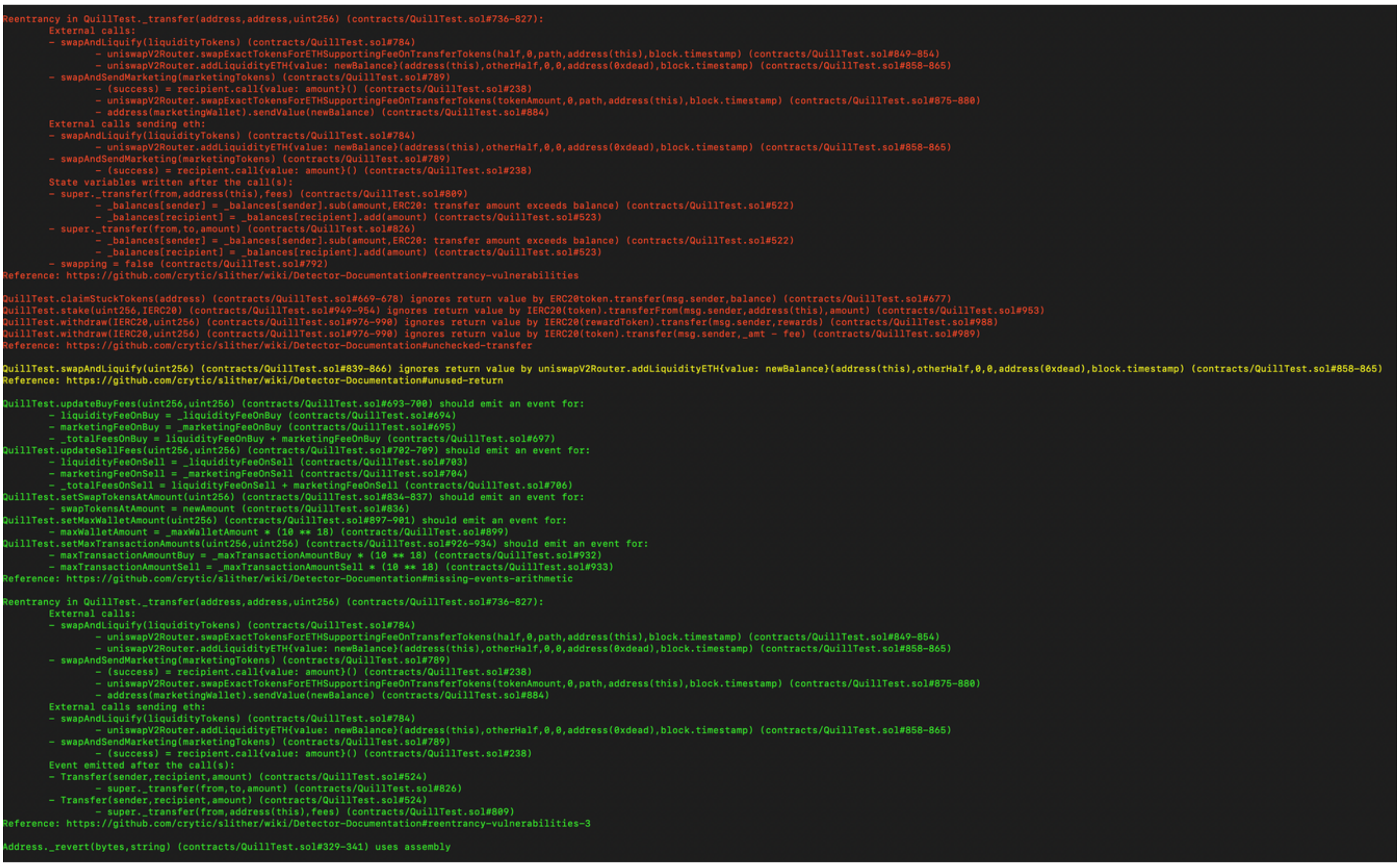
Save deployment fee
File(s): [contracts/QuillTest.sol]

Description

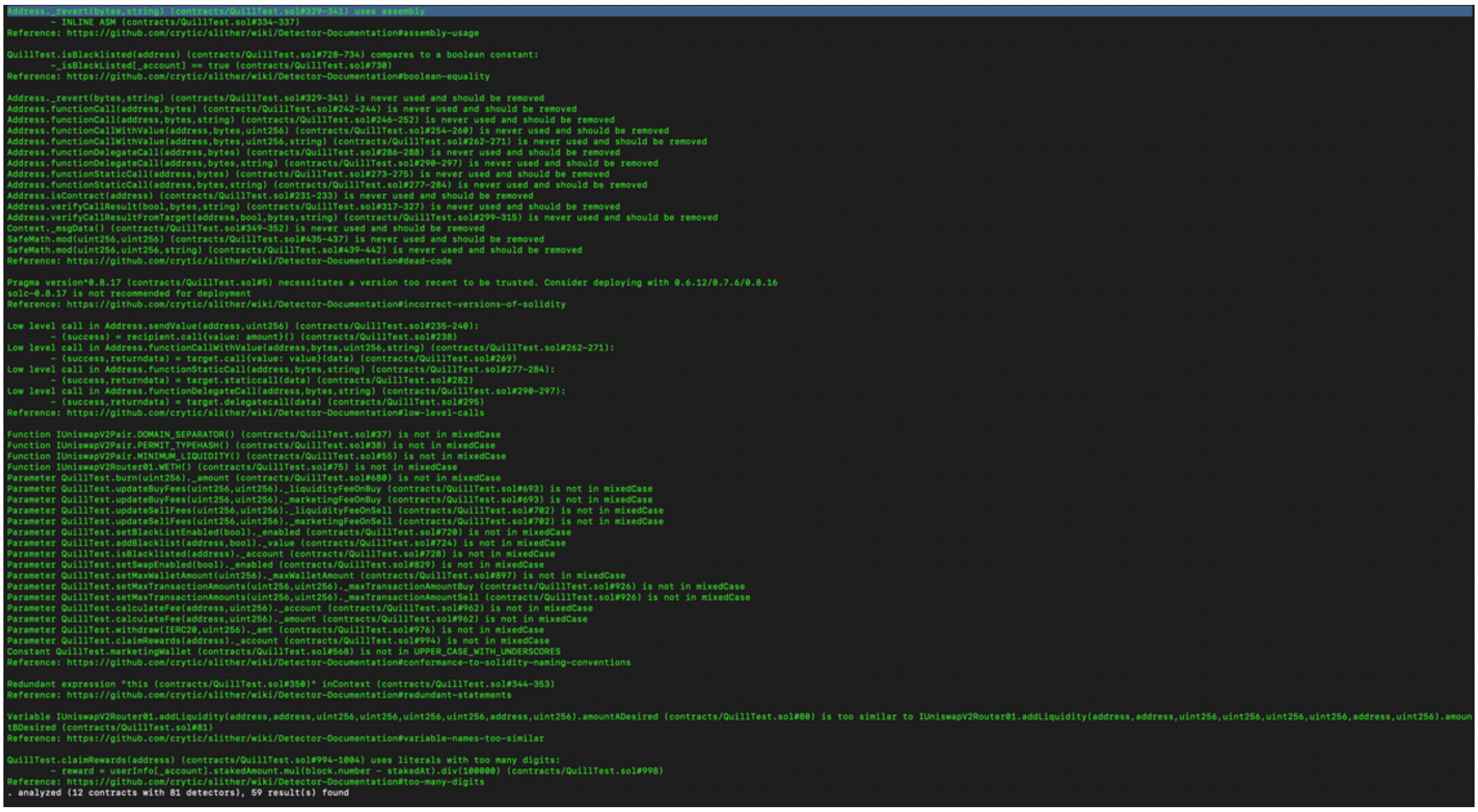
Remove any unnecessary state variables and constructor code that are not being used. In this case, the contract only has two state variables "tradingEnabled", and "swapEnabled" that are set to "false" in the constructor, This will also reduce deployment cost.

```
constructor () ERC20("QuillTest", "QT") {
      ..............//
      ..............//
      tradingEnabled = false;
      swapEnabled = false;
      ..............//
      ..............//
  }
```

Automated Tests (Slither)

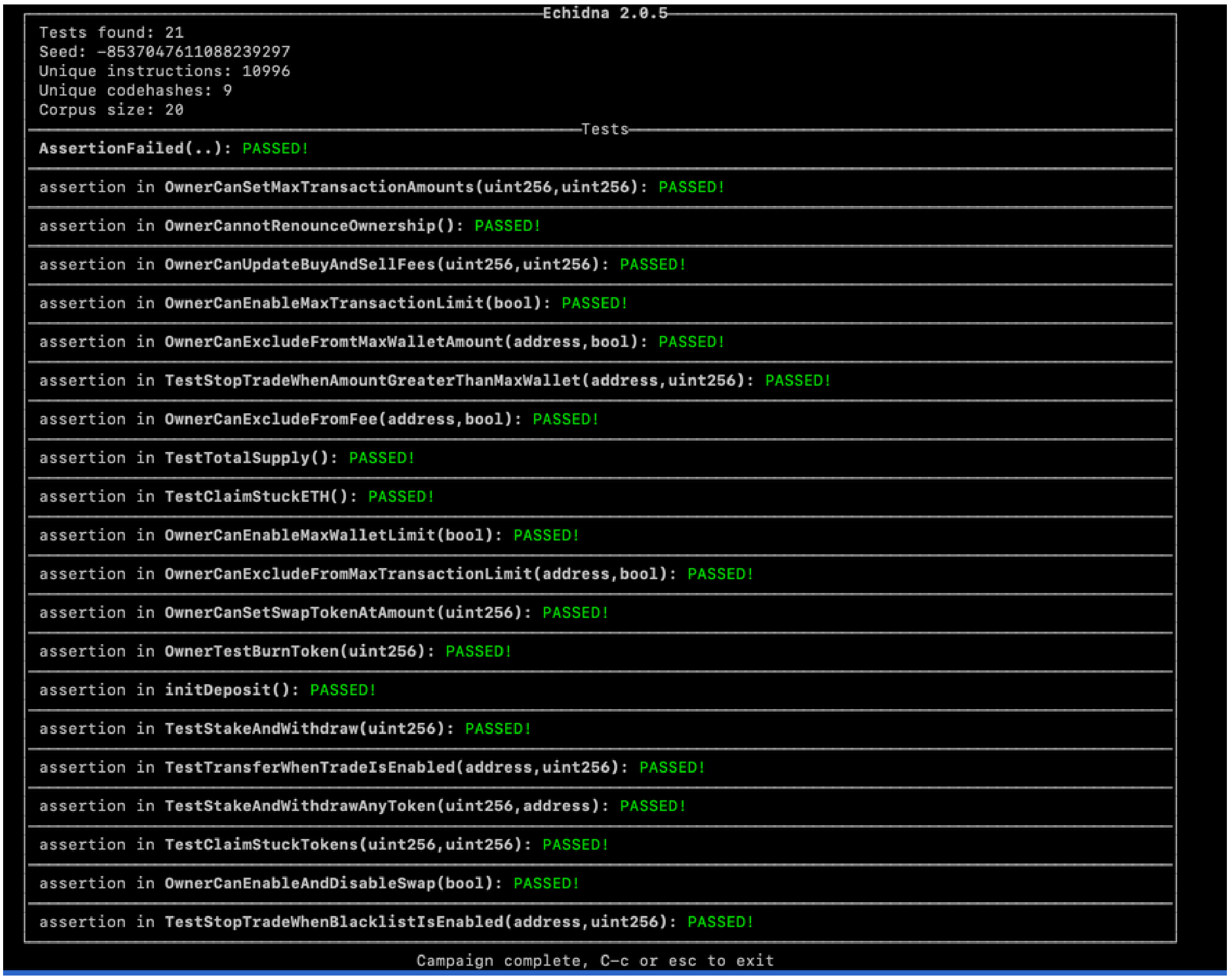


```
Reentrancy in QuillTest._transfer(address,address,uint256) (contracts/QuillTest.sol#736-827):
        External calls:
        - swapAndLiquify(liquidityTokens) (contracts/QuillTest.sol#784)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(half,0,path,address(this),block.timestamp) (contracts/QuillTest.sol#849-854)
                - uniswapV2Router.addLiquidityETH{value: newBalance}(address(this),otherHalf,0,0,address(0xdead),block.timestamp) (contracts/QuillTest.sol#858-865)
        - swapAndSendMarketing(marketingTokens) (contracts/QuillTest.sol#789)
                - (success) = recipient.call{value: amount}() (contracts/QuillTest.sol#238)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (contracts/QuillTest.sol#875-880)
                - address(marketingWallet).sendValue(newBalance) (contracts/QuillTest.sol#884)
        External calls sending eth:
        - swapAndLiquify(liquidityTokens) (contracts/QuillTest.sol#784)
                - uniswapV2Router.addLiquidityETH{value: newBalance}(address(this),otherHalf,0,0,address(0xdead),block.timestamp) (contracts/QuillTest.sol#858-865)
        - swapAndSendMarketing(marketingTokens) (contracts/QuillTest.sol#789)
                - (success) = recipient.call{value: amount}() (contracts/QuillTest.sol#238)
        State variables written after the call(s):
        - super._transfer(from,address(this),fees) (contracts/QuillTest.sol#809)
                - _balances[sender] = _balances[sender].sub(amount,ERC20: transfer amount exceeds balance) (contracts/QuillTest.sol#522)
                - _balances[recipient] = _balances[recipient].add(amount) (contracts/QuillTest.sol#523)
        - super._transfer(from,to,amount) (contracts/QuillTest.sol#826)
                - _balances[sender] = _balances[sender].sub(amount,ERC20: transfer amount exceeds balance) (contracts/QuillTest.sol#522)
                - _balances[recipient] = _balances[recipient].add(amount) (contracts/QuillTest.sol#523)
        - swapping = false (contracts/QuillTest.sol#792)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities

QuillTest.claimStuckTokens(address) (contracts/QuillTest.sol#669-678) ignores return value by ERC20token.transfer(msg.sender,balance) (contracts/QuillTest.sol#677)
QuillTest.stake(uint256,IERC20) (contracts/QuillTest.sol#949-954) ignores return value by IERC20(token).transferFrom(msg.sender,address(this),amount) (contracts/QuillTest.sol#953)
QuillTest.withdraw(IERC20,uint256) (contracts/QuillTest.sol#976-990) ignores return value by IERC20(rewardToken).transfer(msg.sender,rewards) (contracts/QuillTest.sol#988)
QuillTest.withdraw(IERC20,uint256) (contracts/QuillTest.sol#976-990) ignores return value by IERC20(token).transfer(msg.sender,_amt - fee) (contracts/QuillTest.sol#989)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer

QuillTest.swapAndLiquify(uint256) (contracts/QuillTest.sol#839-866) ignores return value by uniswapV2Router.addLiquidityETH{value: newBalance}(address(this),otherHalf,0,0,address(0xdead),block.timestamp) (contracts/QuillTest.sol#858-865)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

QuillTest.updateBuyFees(uint256,uint256) (contracts/QuillTest.sol#693-700) should emit an event for:
        - liquidityFeeOnBuy = _liquidityFeeOnBuy (contracts/QuillTest.sol#694)
        - marketingFeeOnBuy = _marketingFeeOnBuy (contracts/QuillTest.sol#695)
        - _totalFeesOnBuy = liquidityFeeOnBuy + marketingFeeOnBuy (contracts/QuillTest.sol#697)
QuillTest.updateSellFees(uint256,uint256) (contracts/QuillTest.sol#702-709) should emit an event for:
        - liquidityFeeOnSell = _liquidityFeeOnSell (contracts/QuillTest.sol#703)
        - marketingFeeOnSell = _marketingFeeOnSell (contracts/QuillTest.sol#704)
        - _totalFeesOnSell = liquidityFeeOnSell + marketingFeeOnSell (contracts/QuillTest.sol#706)
QuillTest.setSwapTokensAtAmount(uint256) (contracts/QuillTest.sol#834-837) should emit an event for:
        - swapTokensAtAmount = newAmount (contracts/QuillTest.sol#836)
QuillTest.setMaxWalletAmount(uint256) (contracts/QuillTest.sol#897-901) should emit an event for:
        - maxWalletAmount = _maxWalletAmount * (10 ** 18) (contracts/QuillTest.sol#899)
QuillTest.setMaxTransactionAmounts(uint256,uint256) (contracts/QuillTest.sol#926-934) should emit an event for:
        - maxTransactionAmountBuy = _maxTransactionAmountBuy * (10 ** 18) (contracts/QuillTest.sol#932)
        - maxTransactionAmountSell = _maxTransactionAmountSell * (10 ** 18) (contracts/QuillTest.sol#933)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic

Reentrancy in QuillTest._transfer(address,address,uint256) (contracts/QuillTest.sol#736-827):
        External calls:
        - swapAndLiquify(liquidityTokens) (contracts/QuillTest.sol#784)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(half,0,path,address(this),block.timestamp) (contracts/QuillTest.sol#849-854)
                - uniswapV2Router.addLiquidityETH{value: newBalance}(address(this),otherHalf,0,0,address(0xdead),block.timestamp) (contracts/QuillTest.sol#858-865)
        - swapAndSendMarketing(marketingTokens) (contracts/QuillTest.sol#789)
                - (success) = recipient.call{value: amount}() (contracts/QuillTest.sol#238)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (contracts/QuillTest.sol#875-880)
                - address(marketingWallet).sendValue(newBalance) (contracts/QuillTest.sol#884)
        External calls sending eth:
        - swapAndLiquify(liquidityTokens) (contracts/QuillTest.sol#784)
                - uniswapV2Router.addLiquidityETH{value: newBalance}(address(this),otherHalf,0,0,address(0xdead),block.timestamp) (contracts/QuillTest.sol#858-865)
        - swapAndSendMarketing(marketingTokens) (contracts/QuillTest.sol#789)
                - (success) = recipient.call{value: amount}() (contracts/QuillTest.sol#238)
        Event emitted after the call(s):
        - Transfer(sender,recipient,amount) (contracts/QuillTest.sol#524)
                - super._transfer(from,to,amount) (contracts/QuillTest.sol#826)
        - Transfer(sender,recipient,amount) (contracts/QuillTest.sol#524)
                - super._transfer(from,address(this),fees) (contracts/QuillTest.sol#809)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

Address._revert(bytes,string) (contracts/QuillTest.sol#329-341) uses assembly
```



```
Address._revert(bytes,string) (contracts/QuillTest.sol#329-341) uses assembly
        - INLINE ASM (contracts/QuillTest.sol#334-337)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

QuillTest.isBlacklisted(address) (contracts/QuillTest.sol#728-734) compares to a boolean constant:
        -_isBlackListed[_account] == true (contracts/QuillTest.sol#730)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality

Address._revert(bytes,string) (contracts/QuillTest.sol#329-341) is never used and should be removed
Address.functionCall(address,bytes) (contracts/QuillTest.sol#242-244) is never used and should be removed
Address.functionCall(address,bytes,string) (contracts/QuillTest.sol#246-252) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (contracts/QuillTest.sol#254-260) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256,string) (contracts/QuillTest.sol#262-271) is never used and should be removed
Address.functionDelegateCall(address,bytes) (contracts/QuillTest.sol#286-288) is never used and should be removed
Address.functionDelegateCall(address,bytes,string) (contracts/QuillTest.sol#290-297) is never used and should be removed
Address.functionStaticCall(address,bytes) (contracts/QuillTest.sol#273-275) is never used and should be removed
Address.functionStaticCall(address,bytes,string) (contracts/QuillTest.sol#277-284) is never used and should be removed
Address.isContract(address) (contracts/QuillTest.sol#231-233) is never used and should be removed
Address.verifyCallResult(bool,bytes,string) (contracts/QuillTest.sol#317-327) is never used and should be removed
Address.verifyCallResultFromTarget(address,bool,bytes,string) (contracts/QuillTest.sol#299-315) is never used and should be removed
Context._msgData() (contracts/QuillTest.sol#349-352) is never used and should be removed
SafeMath.mod(uint256,uint256) (contracts/QuillTest.sol#435-437) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (contracts/QuillTest.sol#439-442) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.8.17 (contracts/QuillTest.sol#5) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.16
solc-0.8.17 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in Address.sendValue(address,uint256) (contracts/QuillTest.sol#235-240):
        - (success) = recipient.call{value: amount}() (contracts/QuillTest.sol#238)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (contracts/QuillTest.sol#262-271):
        - (success,returndata) = target.call{value: value}(data) (contracts/QuillTest.sol#269)
Low level call in Address.functionStaticCall(address,bytes,string) (contracts/QuillTest.sol#277-284):
        - (success,returndata) = target.staticcall(data) (contracts/QuillTest.sol#282)
Low level call in Address.functionDelegateCall(address,bytes,string) (contracts/QuillTest.sol#290-297):
        - (success,returndata) = target.delegatecall(data) (contracts/QuillTest.sol#295)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

Function IUniswapV2Pair.DOMAIN_SEPARATOR() (contracts/QuillTest.sol#37) is not in mixedCase
Function IUniswapV2Pair.PERMIT_TYPEHASH() (contracts/QuillTest.sol#38) is not in mixedCase
Function IUniswapV2Pair.MINIMUM_LIQUIDITY() (contracts/QuillTest.sol#55) is not in mixedCase
Function IUniswapV2Router01.WETH() (contracts/QuillTest.sol#75) is not in mixedCase
Parameter QuillTest.burn(uint256)._amount (contracts/QuillTest.sol#680) is not in mixedCase
Parameter QuillTest.updateBuyFees(uint256,uint256)._liquidityFeeOnBuy (contracts/QuillTest.sol#693) is not in mixedCase
Parameter QuillTest.updateBuyFees(uint256,uint256)._marketingFeeOnBuy (contracts/QuillTest.sol#693) is not in mixedCase
Parameter QuillTest.updateSellFees(uint256,uint256)._liquidityFeeOnSell (contracts/QuillTest.sol#702) is not in mixedCase
Parameter QuillTest.updateSellFees(uint256,uint256)._marketingFeeOnSell (contracts/QuillTest.sol#702) is not in mixedCase
Parameter QuillTest.setBlackListEnabled(bool)._enabled (contracts/QuillTest.sol#720) is not in mixedCase
Parameter QuillTest.addBlacklist(address,bool)._value (contracts/QuillTest.sol#724) is not in mixedCase
Parameter QuillTest.isBlacklisted(address)._account (contracts/QuillTest.sol#728) is not in mixedCase
Parameter QuillTest.setSwapEnabled(bool)._enabled (contracts/QuillTest.sol#829) is not in mixedCase
Parameter QuillTest.setMaxWalletAmount(uint256)._maxWalletAmount (contracts/QuillTest.sol#897) is not in mixedCase
Parameter QuillTest.setMaxTransactionAmounts(uint256,uint256)._maxTransactionAmountBuy (contracts/QuillTest.sol#926) is not in mixedCase
Parameter QuillTest.setMaxTransactionAmounts(uint256,uint256)._maxTransactionAmountSell (contracts/QuillTest.sol#926) is not in mixedCase
Parameter QuillTest.calculateFee(address,uint256)._account (contracts/QuillTest.sol#962) is not in mixedCase
Parameter QuillTest.calculateFee(address,uint256)._amount (contracts/QuillTest.sol#962) is not in mixedCase
Parameter QuillTest.withdraw(IERC20,uint256)._amt (contracts/QuillTest.sol#976) is not in mixedCase
Parameter QuillTest.claimRewards(address)._account (contracts/QuillTest.sol#994) is not in mixedCase
Constant QuillTest.marketingWallet (contracts/QuillTest.sol#568) is not in UPPER_CASE_WITH_UNDERSCORES
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Redundant expression "this (contracts/QuillTest.sol#350)" inContext (contracts/QuillTest.sol#344-353)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements

Variable IUniswapV2Router01.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amountADesired (contracts/QuillTest.sol#80) is too similar to IUniswapV2Router01.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amoun
tBDesired (contracts/QuillTest.sol#81)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-too-similar

QuillTest.claimRewards(address) (contracts/QuillTest.sol#994-1004) uses literals with too many digits:
        - reward = userInfo[_account].stakedAmount.mul(block.number - stakedAt).div(100000) (contracts/QuillTest.sol#998)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
. analyzed (12 contracts with 81 detectors), 59 result(s) found
```

Echidna test results



```
Echidna 2.0.5
Tests found: 21
Seed: -8537047611088239297
Unique instructions: 10996
Unique codehashes: 9
Corpus size: 20
Tests
AssertionFailed(..): PASSED!
assertion in OwnerCanSetMaxTransactionAmounts(uint256,uint256): PASSED!
assertion in OwnerCannotRenounceOwnership(): PASSED!
assertion in OwnerCanUpdateBuyAndSellFees(uint256,uint256): PASSED!
assertion in OwnerCanEnableMaxTransactionLimit(bool): PASSED!
assertion in OwnerCanExcludeFromtMaxWalletAmount(address,bool): PASSED!
assertion in TestStopTradeWhenAmountGreaterThanMaxWallet(address,uint256): PASSED!
assertion in OwnerCanExcludeFromFee(address,bool): PASSED!
assertion in TestTotalSupply(): PASSED!
assertion in TestClaimStuckETH(): PASSED!
assertion in OwnerCanEnableMaxWalletLimit(bool): PASSED!
assertion in OwnerCanExcludeFromMaxTransactionLimit(address,bool): PASSED!
assertion in OwnerCanSetSwapTokenAtAmount(uint256): PASSED!
assertion in OwnerTestBurnToken(uint256): PASSED!
assertion in initDeposit(): PASSED!
assertion in TestStakeAndWithdraw(uint256): PASSED!
assertion in TestTransferWhenTradeIsEnabled(address,uint256): PASSED!
assertion in TestStakeAndWithdrawAnyToken(uint256,address): PASSED!
assertion in TestClaimStuckTokens(uint256,uint256): PASSED!
assertion in OwnerCanEnableAndDisableSwap(bool): PASSED!
assertion in TestStopTradeWhenBlacklistIsEnabled(address,uint256): PASSED!
Campaign complete, C-c or esc to exit
```

Automated Tests Results

Some false positive errors were reported by the tool. All the other issues have been categorized above according to their level of severity. Attached to this is a detailed POC.

About Prosper

Prosper Onah is a smart contract auditor. As a smart contract auditor, he is responsible for conducting thorough assessments of smart contracts to identify potential security vulnerabilities and ensure that the contract operates as intended. This includes reviewing the code, testing its functionality, and analyzing its architecture to identify potential risks and vulnerabilities. With expertise in the field, I provides valuable insights and recommendations to improve the security and overall functionality of smart contracts.