QuillAudits

Audit Report
November, 2023

For

Watt Trade

Table of Content

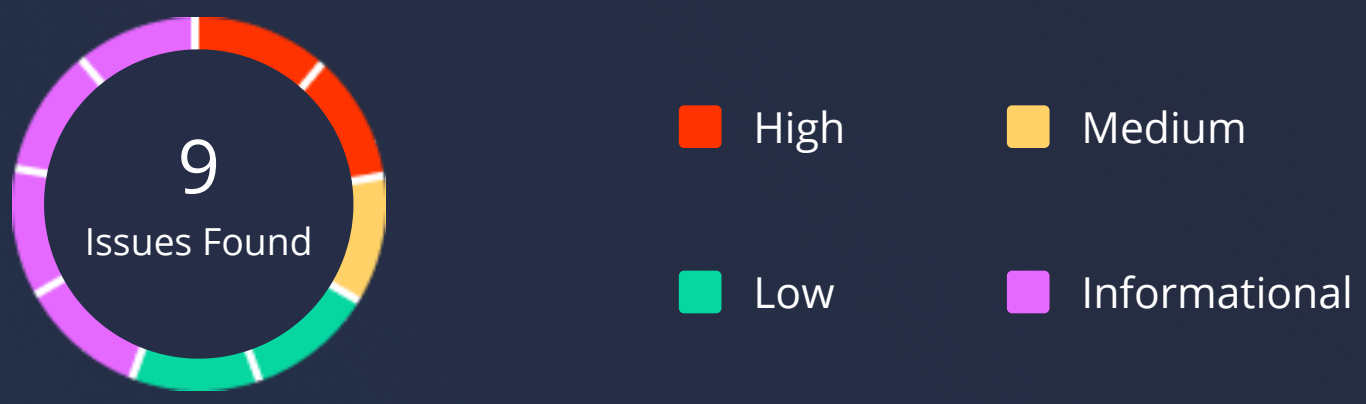
Table of Content

Executive Summary

Project Name	Watt2Trade
Project URL	*https://watt2trade.com/*
Overview	Watt2Trade Escrow Account is a smart contract tailored around the electricity market that employs the spread options mechanism. Traders can place a put or call spread option order with the aim of a payoff derived from the differences in prices. Traders can also engage in future orders whose match system between sellers and buyers is based on a first-in-first-out basis. The contract allows for cancellation of any kind of option. ElectricityPrices contract allows the contract admin to update the prices of electricity based on the market, year, month, and day. These data get fetched in the Escrow Account during withdrawals. The StakingVault contract creates a platform for users to stake their Energy coin tokens from which they benefit from the kilowatt reduced fee from the Escrow contract. Users can remove their energy coins from the contract as well.
Audit Scope	*https://github.com/Watt-2-Trade/dex-smartcontracts/tree/main/contracts*
Contracts in Scope	contracts/Common.sol contracts/StakingVault.sol contracts/EscrowAccount.sol contracts/ElectricityPrices.sol
Commit Hash	50338be7b27cc73cf60bdf882d61b7d25f083612
Language	Solidity
Blockchain	Polygon
Method	Manual Testing, Automated Tests, Functional Testing
Review 1	17th October 2023 - 5th November 2023

Updated Code Received	5th November 2023
Review 2	7th November 2023 - 8th November 2023
Fixed In	*https://github.com/Watt-2-Trade/dex-smartcontracts/commit/9d4cda0860ce74d13e20d50bf77ac36b713cba37*

Number of Security Issues per Severity



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

Checked Vulnerabilities

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Byte array
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Redundant fallback function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- 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-20 token 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

Remix IDE, Truffle, Solhint, Mythril, Slither, Solidity Statistic 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.

A. Common Issues

High Severity Issues

No issues were found.

Medium Severity Issues

No issues were found.

Low Severity Issues

A.1 Update all state changes before invoking external functions

```
function removeCoins(string calldata market↑, uint quantity↑) external {
  require(
    stakingBalancesPerMarket[msg.sender][market↑] >= quantity↑,
    "stakingbalance too low"
  );
  require(
    stakingConsumption[msg.sender][market↑][
      _mapLiquidationDate(block.timestamp)
    ] <= stakingBalancesPerMarket[msg.sender][market↑] - quantity↑,
    "already consumed too many coins today"
  );

  IERC20(ENERGY_COIN_ADDRESS).transfer(
    msg.sender,
    quantity↑
  );

  totalStakingBalances[msg.sender] -= quantity↑;
  stakingBalancesPerMarket[msg.sender][market↑] -= quantity↑;
  emit CoinsRemoved(msg.sender, market↑, quantity↑);
}
```

Description

In the removeCoins function from the StakingVault contract, an external call was made to the Energy Coin contract before state variables were updated. This exposes the contract to the possibility of a reentrancy. This negates the principle of the check-effect interaction that poses the need for all state updates before an external call. This is also common in the withdrawOptionOrder and withdraw.

StakingVault:

- removeCoins

A.1 Update all state changes before invoking external functions

EscrowAccount:

- withdrawOptionOrder
- withdrawFutureOrder

Remediation

Follow the check-effect interaction to resolve this issue. Invoke the transfer function from ENERGY_COIN address after the state variables have been updated. For the withdraw functions from the EscrowAccount, create a local address variable to copy the address of the trader, delete the values from the mapping and then make external calls.

Status

Resolved

A.2 Permanent unprotected admin address

Description

Hardcoding the admin address set to constant and also not a multisig creates a point of target to attackers and leaves the contract permanently unprotected if compromised. This will make it impossible to carry out a swift incident response process to curtail the attack. This is common across all contracts under audit scope.

Remediation

Use multisig.

Status

Resolved

Watt2Trade Team's Comment:

We have setup a safe wallet for production (matic:0xaCf7bcFca9A83E4ca2a514e48f77B614d6f698CF). There is no safe for Mumbai, so we'll use it for prod deploy only.

Informational Issues

A.3 Functions certain to fail when called by unpermitted address should be marked as payable to save gas

Description

There are some privileged functions in all the contracts under scope that allow for either admin or whitelisted addresses to call, else it fails. Since these functions are certain to fail when triggered when unpermitted addresses, they can be marked as payable functions. Payable functions are cheaper when called and can save over 50 gas or more.

StakingVault:

- consumeCoins
- whitelistContractAddress
- changeEnergyCoinAddress

ElectricityPrices:

- storeElectricityPrices

EscrowAccount:

- updateFees
- updateCapAndfloor
- adjustClosingTime
- adjustMaximums
- adjustContractAddresses

Remediation

To derive a gas-optimized function, mark these functions as payable.

Status

Resolved

A.4 Missing check for zero address

Description

Updating state changes that pertains to addresses are a critical action especially avoiding setting any to the null address. For instance, on changing the ENERGY_COIN address in the staking vault contract to a zero address, this would disrupt the contract flow and cause a denial of service.

StakingVault:

- consumeCoins

EscrowAccount:

- adjustContractAddresses

Remediation

Add zero address check in these functions to prevent setting critical state variables to a null address.

Status

Resolved

Reference

https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

A.5 Comparison of values to a boolean constant

Description

Some mappings were used in the contract to map addresses to a boolean. This is to help know when a value is true or false. However, because these are the only two possible outcomes, it is recommended to use the mapping to serve as the condition within the require statement. W2TContractAddress[msg.sender] is expected to return a true while ! W2TContractAddress[msg.sender] for false.

StakingVault:

- onlyW2T modifier

EscrowAccount:

- placeOptionSellOrder
- placeOptionBuyOrder

A.5 Comparison of values to a boolean constant

https://github.com/Watt-2-Trade/dex-smartcontracts/blob/50338be7b27cc73cf60bdf882d61b7d25f083612/contracts/EscrowAccount.sol#L334

https://github.com/Watt-2-Trade/dex-smartcontracts/blob/50338be7b27cc73cf60bdf882d61b7d25f083612/contracts/EscrowAccount.sol#L418

https://github.com/Watt-2-Trade/dex-smartcontracts/blob/50338be7b27cc73cf60bdf882d61b7d25f083612/contracts/EscrowAccount.sol#L498

https://github.com/Watt-2-Trade/dex-smartcontracts/blob/50338be7b27cc73cf60bdf882d61b7d25f083612/contracts/EscrowAccount.sol#L519

https://github.com/Watt-2-Trade/dex-smartcontracts/blob/50338be7b27cc73cf60bdf882d61b7d25f083612/contracts/EscrowAccount.sol#L557

https://github.com/Watt-2-Trade/dex-smartcontracts/blob/50338be7b27cc73cf60bdf882d61b7d25f083612/contracts/EscrowAccount.sol#L604

https://github.com/Watt-2-Trade/dex-smartcontracts/blob/50338be7b27cc73cf60bdf882d61b7d25f083612/contracts/EscrowAccount.sol#L712

https://github.com/Watt-2-Trade/dex-smartcontracts/blob/50338be7b27cc73cf60bdf882d61b7d25f083612/contracts/EscrowAccount.sol#L900

https://github.com/Watt-2-Trade/dex-smartcontracts/blob/50338be7b27cc73cf60bdf882d61b7d25f083612/contracts/EscrowAccount.sol#L968

https://github.com/Watt-2-Trade/dex-smartcontracts/blob/50338be7b27cc73cf60bdf882d61b7d25f083612/contracts/EscrowAccount.sol#L1012

Remediation

These values are ordinarily expected to return a boolean value. Use these values without making an explicit comparison to a boolean constant.

Status

Resolved

Reference

https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality

B. Contract - EscrowAccount

High Severity Issues

B.1 Not updating on withdrawal opens opportunity to draining funds

Description

The withdrawOptionOrder() function doesn't delete the sell order on withdraw of sell trade as stated here: https://github.com/Watt-2-Trade/dex-smartcontracts/blob/main/contracts/EscrowAccount.sol#L707

However, it doesn't include any update to the withdrawal. Trader of the optionSellOrder can execute this action for an infinite amount to drain out funds.

Remediation

Track changes made on withdrawal and ensure they are updated before trader receives the funds i.e order.quantityMatched.

Status

Resolved

B.2 Costly DOS for futureOrder traders due to cancellations array loop execution in withdrawFutureOrder()

Description

When a trader calls withdrawFutureOrder(), it iterates through the FutureCancellations array of the futureOrder to ensure the trader is eligible by first-come-first-serve approach. The issue arises from the cancellations array being a user manipulated array and also never gets decreased. Iterating through such an array where users can decide how large the array can grow could cause a revert due to out of gas error, meaning traders attempting to withdraw their futureOrder would fail causing DOS.
https://github.com/Watt-2-Trade/dex-smartcontracts/blob/main/contracts/EscrowAccount.sol#L725

Remediation

Consider reimplementing cancellation and withdrawal mechanisms for futureOrders in a way that doesn't rely on this approach.

Status

Resolved

Medium Severity Issues

No issues were found.

Low Severity Issues

No issues were found.

Informational Issues

B.3 Redundant code creation

```
// stores a new fee generation
function updateFees(
  uint newReducedFee,
  uint newFullFee,
  int newPlacementFee,
  uint newCancellationFeePercentage
) external onlyAdmin {
  uint newGen = CURRENT_FEE_GEN += 1;

  FEE_CONSTANTS[newGen].REDUCED_TRADING_FEE_KW = newReducedFee;
  FEE_CONSTANTS[newGen].FULL_TRADING_FEE_KW = newFullFee;
  FEE_CONSTANTS[newGen]
    .CANCELLATION_FEE_PERCENTAGE = newCancellationFeePercentage;
  FEE_CONSTANTS[newGen].PLACEMENT_FEE_KW = newPlacementFee;

  CURRENT_FEE_GEN = newGen;

  emit FeesChanged(
    newReducedFee,
    newFullFee,
    newPlacementFee,
    newCancellationFeePercentage
  );
}
```

```
// stores a new cap generation
function updateCapAndFloor(
  int newCapPrice,
  int newFloorPrice
) external onlyAdmin {
  uint newGen = CURRENT_CAP_GEN += 1;

  MARKET_CAP_CONSTANTS[newGen].CAP_PRICE_MARKET_KW = newCapPrice;
  MARKET_CAP_CONSTANTS[newGen].FLOOR_PRICE_MARKET_KW = newFloorPrice;

  CURRENT_CAP_GEN = newGen;

  emit CapsChanged(newCapPrice, newFloorPrice);
}
```

B.3 Redundant code creation

Description

Unnecessary code implemented on update functions can be removed to save gas. In the preceding lines before the issue pointed out, a new local variable was created called newGen. The newGen variable equals the summation of the CURRENT_FEE_GEN and 1. After updating all mappings that pertains to the fee, the state variable CURRENT_FEE_GEN was afterwards set to newGen. For an optimized code, it is okay to update the CURRENT_FEE_GEN.

https://github.com/Watt-2-Trade/dex-smartcontracts/blob/main/contracts/EscrowAccount.sol#L1042

https://github.com/Watt-2-Trade/dex-smartcontracts/blob/main/contracts/EscrowAccount.sol#L1062

Status

Resolved

C. Contract - StakingVault

High Severity Issues

No issues were found.

Medium Severity Issues

C.1 Lack of unwhitelist function on sensitive permission lead to unavertable DOS privilege

Description

Admin calling `StakingVault::whitelistContractAddress()` on any address allows that address to call the `StakingVault::consumeCoins()` function, this also gives that address the ability to DOS any user from removing their coins. However, there is NO undo function on this operation, meaning if this happens it would be permanent and unavertable.

Remediation

Implement an undo mechanism to avert this critical privilege from addresses previously whitelisted.

Status

Resolved

Low Severity Issues

No issues were found.

Informational Issues

No issues were found.

Functional Tests

Some of the tests performed are mentioned below:

EscrowAccount

- ✓ Should allow some buyers to cancel option order before the liquidation time
- ✓ Should revert when the liquidation time sets for an order exceed the maximum year
- ✓ Should revert when withdrawal functions are invoked but electricity prices are unavailable
- ✓ Should successfully call withdraw as a buyer when there are no seller option in the pool
- ✓ Should withdraw higher amount if traders have some staked amount in the stake vault
- ✓ Should revert when withdraw is called but the escrow account is not whitelisted to call the consumeCoins function yet
- ✓ Should revert when withdraw function is called but there are numerous canceled future orders - ploy to make the length of cancellation high
- ✓ Should verify that future order buyers get matched on a first-in-first-out basis

StakingVault

- ✓ Energy coin token holders can stake successfully on approving the stake contract
- ✓ Only staked users can remove their token (apply CEI on the removeFunction)
- ✓ Users can query for their consumable coins for a market
- ✓ Users staking consumption increases when they call withdraw from Escrow contract
- ✓ Only whitelisted addresses can invoke the consumeCoins function

ElectricityPrices

- ✓ Only admin can store electricity prices
- ✓ Should fetch the information of electricity prices and its status availability
- ✓ Should revert when non-admin invokes the setoreElectricityPrices function

Automated Tests

```
StakingVault.onlyW2T() (StakingVault.sol#30-36) compares to a boolean constant:
        -require(bool,string)(W2TContractAddress[msg.sender] == true,admin-only function) (StakingVault.sol#31-34)
EscrowAccount.placeOptionSellOrder(EscrowAccount.OptionSellOrderInput) (EscrowAccount.sol#231-316) compares to a boolean constant:
        -require(bool,string)(optionSellOrders[input.orderId].exists == false,orderid already exists) (EscrowAccount.sol#259-262)
EscrowAccount.placeOptionBuyOrder(EscrowAccount.OptionBuyOrderInput) (EscrowAccount.sol#319-387) compares to a boolean constant:
        -require(bool,string)(optionBuyOrders[input.ownOrderId].exists == false,orderid already exists) (EscrowAccount.sol#333-336)
EscrowAccount.placeOptionBuyOrder(EscrowAccount.OptionBuyOrderInput) (EscrowAccount.sol#319-387) compares to a boolean constant:
        -require(bool,string)(optionSellOrders[input.sellOrderId].exists == true,sell-order doesn't exist) (EscrowAccount.sol#320-323)
EscrowAccount.placeFutureOrder(EscrowAccount.FutureOrderInput) (EscrowAccount.sol#390-490) compares to a boolean constant:
        -require(bool,string)(futureOrders[input.orderId].exists == false,orderid already exists) (EscrowAccount.sol#417-420)
EscrowAccount.cancelOptionOrder(uint256,IWatt2Trade.TradingSide) (EscrowAccount.sol#492-554) compares to a boolean constant:
        -require(bool,string)(optionBuyOrders[orderId].exists == true,order doesn't exist) (EscrowAccount.sol#498)
EscrowAccount.cancelOptionOrder(uint256,IWatt2Trade.TradingSide) (EscrowAccount.sol#492-554) compares to a boolean constant:
        -require(bool,string)(optionSellOrders[orderId].exists == true,order doesn't exist) (EscrowAccount.sol#519)
EscrowAccount.cancelFutureOrder(uint256) (EscrowAccount.sol#556-600) compares to a boolean constant:
        -require(bool,string)(futureOrders[orderId].exists == true,order doesn't exist) (EscrowAccount.sol#557)
EscrowAccount.withdrawOptionOrder(uint256,IWatt2Trade.TradingSide) (EscrowAccount.sol#602-709) compares to a boolean constant:
        -require(bool,string)(optionSellOrders[orderId].exists == true,order doesn't exist) (EscrowAccount.sol#658)
EscrowAccount.withdrawOptionOrder(uint256,IWatt2Trade.TradingSide) (EscrowAccount.sol#602-709) compares to a boolean constant:
        -require(bool,string)(optionBuyOrders[orderId].exists == true,order doesn't exist) (EscrowAccount.sol#604)
EscrowAccount.withdrawFutureOrder(uint256) (EscrowAccount.sol#711-791) compares to a boolean constant:
        -require(bool,string)(futureOrders[orderId].exists == true,order doesn't exist) (EscrowAccount.sol#712)
EscrowAccount._calculateSpreadOption(IWatt2Trade.PriceType,IWatt2Trade.TradingSide,IWatt2Trade.OptionType,int256,int256,int256) (EscrowAccount
.sol#886-952) compares to a boolean constant:
        -require(bool,string)(priceAvailable == true,price is not available yet) (EscrowAccount.sol#900)
EscrowAccount._calculateSpreadFuture(IWatt2Trade.PriceType,IWatt2Trade.TradingSide,int256,int256) (EscrowAccount.sol#955-984) compares to a bo
olean constant:
        -require(bool,string)(priceAvailable == true,price is not available yet) (EscrowAccount.sol#968)
EscrowAccount._determineGenerations(uint256) (EscrowAccount.sol#1008-1022) compares to a boolean constant:
        -GENERATIONS_PER_DAY[liquidationDateTimestamp].exists == false (EscrowAccount.sol#1013)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality
```

```
Reentrancy in StakingVault.removeCoins(string,uint256) (StakingVault.sol#54-74):
        External calls:
        - IERC20(ENERGY_COIN_ADDRESS).transfer(msg.sender,quantity) (StakingVault.sol#66-69)
        State variables written after the call(s):
        - stakingBalancesPerMarket[msg.sender][market] -= quantity (StakingVault.sol#72)
        StakingVault.stakingBalancesPerMarket (StakingVault.sol#17) can be used in cross function reentrancies:
        - StakingVault.consumeCoins(address,string,uint256,uint256) (StakingVault.sol#76-94)
        - StakingVault.queryConsumableCoins(string) (StakingVault.sol#96-104)
        - StakingVault.removeCoins(string,uint256) (StakingVault.sol#54-74)
        - StakingVault.stakeCoins(string,uint256) (StakingVault.sol#42-52)
        - StakingVault.stakingBalancesPerMarket (StakingVault.sol#17)
```

```
StakingVault.changeEnergyCoinAddress(address).newAddress (StakingVault.sol#118) lacks a zero-check on :
                - ENERGY_COIN_ADDRESS = newAddress (StakingVault.sol#119)
EscrowAccount.adjustContractAddresses(address,address,address).newUSDC (EscrowAccount.sol#1087) lacks a zero-check on :
                - USDC_ADDRESS = newUSDC (EscrowAccount.sol#1091)
EscrowAccount.adjustContractAddresses(address,address,address).newElec (EscrowAccount.sol#1088) lacks a zero-check on :
                - ELECTRICITY_ADDRESS = newElec (EscrowAccount.sol#1092)
EscrowAccount.adjustContractAddresses(address,address,address).newStake (EscrowAccount.sol#1089) lacks a zero-check on :
                - STAKING_VAULT_ADDRESS = newStake (EscrowAccount.sol#1093)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
```

```
EscrowAccount.adjustClosingTime(uint256) (EscrowAccount.sol#1069-1071) should emit an event for:
        - CLOSING_TIME_HOUR_MARKET_UTC = newClosingTime (EscrowAccount.sol#1070)
EscrowAccount.adjustMaximums(uint256,int256,int256) (EscrowAccount.sol#1074-1082) should emit an event for:
        - MAX_YEARS_PLACEMENT = newMaximumYears (EscrowAccount.sol#1079)
        - MAX_QUANTITY = newMaximumQuantity (EscrowAccount.sol#1080)
        - MAX_PRIME = newMaximumPrime (EscrowAccount.sol#1081)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic
```

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.

Closing Summary

In this report, we have considered the security of the Watt2Trade. We performed our audit according to the procedure described above.

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 Watt2Trade 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 Watt2Trade 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 Watt2Trade 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

Watt Trade

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com