QuillAudits

Audit Report
December, 2022

For

Blockchain Australia®

NEO VIRTUAL TECH

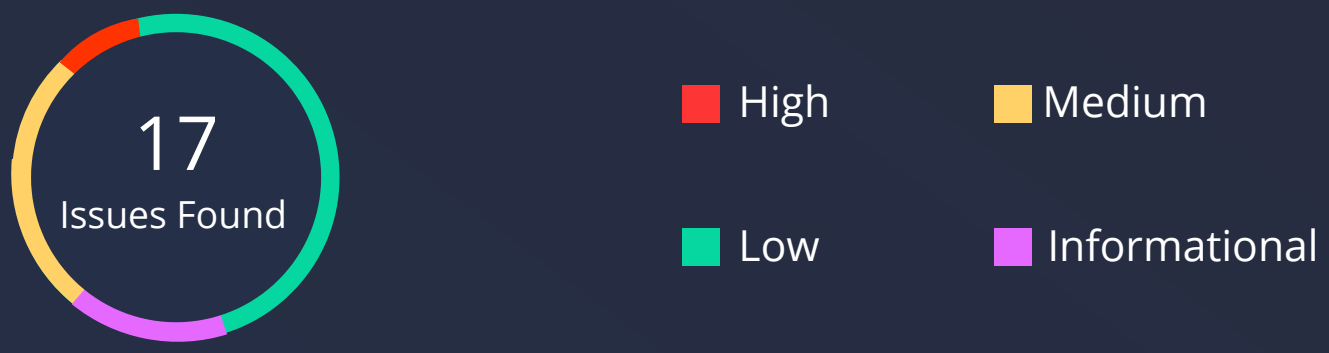
NEO LEGAL

Table of Content

Executive Summary

Project Name	DecryptMarketplace
Overview	DecryptMarketplace is an NFT marketplace that allows for the trading of ERC721 and ERC1155 tokens. This marketplace features four different tokens; SimpleERC721, ExtendedERC721, SimpleERC1155, and ExtendedERC1155. The differences that exist between the Simple and Extended tokens are the characteristics of a Presale feature present in the Extended tokens. The marketplace utilizes the Openzeppelin UUPSUpgradable.sol, Initializable.sol, OwnableUpgradable.sol for upgradability of DecryptMarketplace and access control features, respectively. It also integrated ERC2981 features to handle the distribution of royalties among involved parties in the marketplace and also EIP712Upgradable for cryptography; used in the recognition of the signatures of sellers and buyers.
Timeline	September 6, 2022 - 9th November, 2022
Method	Manual Review, Functional Testing, Automated Testing, etc.
Scope of Audit	The scope of this audit was to analyse DecryptMarketplace.sol, CustomERC721.sol, PreSale721.sol, SimpleERC721.sol, SimpleERC721Deployer.sol, ExtendedERC721.sol, ExtendedERC721Deployer.sol, CustomERC1155.sol, PreSale1155.sol, SimpleERC1155.sol, SimpleERC1155Deployer.sol, ExtendedERC1155.sol, ExtendedERC1155Deployer.sol, CreatorV1, and Royalty
Commit Hash	*https://github.com/Blockchain-australia/DecryptMarketPlaceContracts/commit/6fb3123d154e3a8a0ffe360d2b3452f9a3770ca3*
Fixed In	6fb3123d154e3a8a0ffe360d2b3452f9a3770ca3

Executive Summary



	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	3	2
Partially Resolved Issues	0	0	0	0
Resolved Issues	1	5	4	2

Executive Summary

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.

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.

Checked Vulnerabilities

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

Techniques and Methods

Throughout the audit of smart contract, 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 analysed 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 analysed, 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 behaviour 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.

Manual Testing

A. Common Issues

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

A.1 Missing events emission for significant actions

Contracts Affected: DecryptMarketplace.sol, PreSale721.sol, SimpleERC721Deployer.sol, ExtendedERC721Deployer.sol, RoyaltyDistribution.sol, PreSale1155.sol, SimpleERC1155Deployer, ExtendedERC1155Deployer.sol

Description

Whenever certain significant privileged actions are performed within the contract, it is recommended to emit an event about it. When these events are thoroughly emitted in contracts, it makes it easier to query events on-chain. Critical actions such as setting the creators of a token when deployed, reservation of tokens for an address, setting admin and super admin of the marketplace, adding addresses to whitelist are important hence must be emitted.
In DecryptMarketplace.sol:
- setSuperAdminShare
- setAdminAddress
In PreSale721.sol:
- addToWhitelist
- setSpecialPriceForToken
- reserveToken
- setPreSalePaymentToken
In SimpleERC721Deployer.sol:
- setCreator
In ExtendedERC721Deployer.sol:
- setCreator
In PreSale1155.sol:
- addToWhitelist
- setSpecialPriceForToken
- setPreSalePaymentToken

In SimpleERC1155Deployer.sol:
- setCreator
In ExtendedERC1155Deployer.sol:
- setCreator
In RoyaltyDistribution.sol:
- setGlobalRoyalty
- setRoyaltyReceiver
- setTokenRoyalty

Remediation

Consider emitting an event whenever certain significant changes are made in the contracts which needs to be notified or noted.

Status

Resolved

A.2 Centralization

Description

The contract(s) contains some functions like setRoyaltyLimit, setMarketplaceFee, setSuperAdminShare, restrictTokenToThisMarketplace in marketplace contract where owner has control and can set fees upto certain limits and can even restrict some tokens to the the marketplace contract only, if it is using custom token contract like CustomERC721 and CustomERC1155.

Recommendation

The multisig wallet can be used to use decentralized way for these owner activities.

Status

Acknowledged

Decrypt team's comment: We'll explore and set up a multisign wallet.

Informational Issues

A.3 Ensure default royalty is always added.

Description

Ensure that default royalty is always getting added in the token contract. In the case where tokenCollaboratorsRoyaltyShare for any token id is set (using tokenContract:setTokenRoyaltyDistribution()) and it is expected that marketplace contract's transferCoins() should distribute the shares to these addresses. but on L693 if tokenContract.getDefaultRoyaltyDistribution().length is 0 then the “if” block won't execute where the nested “if else” blocks are there to ensure that royalty distribution is happening. So here “if” on L698 should execute as if "getTokenRoyaltyDistribution(payment.tokenId).length" is greater than 0. which is may not execute in the case where tokenContract.getDefaultRoyaltyDistribution().length is 0.

Recommendation

Review the code and ensure that defaultCollaboratorsRoyaltyShare in contracts is getting set.

Status

Acknowledged

Decrypt team’s comment: We are making sure on the application level, setting the default royalty while deploying the contract.

A.4 Floating pragma

Description

Contracts are using floating pragma (pragma solidity ^0.8.0) Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly. Using floating pragma does not ensure that the contracts will be deployed with the same version. It is possible that the most recent compiler version get selected while deploying contract which has higher chances of having bugs in it.

Recommendation

Remove floating pragma and use specific compiler version with which contracts have been tested.

Status

Resolved

B. Contract - DecryptMarketplace.sol

High Severity Issues

B.1 Reentrancy in prePurchase:

Description

prePurchase() is getting used to purchase tokens in the presale event. while sending ERC721 or ERC1155, transferTokens() function is getting used first on L485,L514 and then countTokensBought() getting called to update the state of NFT contract which uses tokensBoughtDuringEvent and preSaleEventInfo mappings to check if token is available for buyer. In this case reentrant call can be made via transferTokens() which uses lazyMint() and/or safeTransferFrom() to send the NFT to to msg.sender address.

Exploit scenario:
1) Attacker creates a malicious contract having code to make calls to Marketplace:prePurchase().

2) prePurchase() uses transferTokens() function to send NFT to msg.sender (which is attacker contract in this case).

3) transferTokens() which uses lazyMint() (lazymint uses _safeMint) and/or safeTransferFrom() to send the NFT to to msg.sender address.

4) in openzeppelin ERC721 , both _safeMint and safeTransferFrom functions use _checkOnERC721Received() to check if receiver supports ERC721 tokens. in this case the _checkOnERC721Received() is going to get called on attacker contract which contains malicious _checkOnERC721Received() which reenters into the transaction flow and then calls prePurchase() to buy NFTs.

5) prePurchase() again transfers NFTs to attacker contract as countTokensBought() is not yet executed and getTokenInfo() still returns availableForBuyer to true even while attacker can exceed the tokensBoughtDuringEvent and maxTokensPerWallet limits or any other state.

Note : Also,this type of scenario can occur when purchasing ERC1155 as the transferTokens() function is using lazyMint function which uses OZ ERC1155 implementation's _mint() and which uses _doSafeTransferAcceptanceCheck() to check if receiver contract supports erc1155 by calling onERC1155Received() on that address which opens path for reentrancy where onERC1155Received() can contain malicious code.

```
484
485         transferTokens(tokenAddress↑, tokenId↑, ownerAddress↑, msg.sender, quantity↑, true);
486
487         PaymentInfo memory payment = PaymentInfo(
488             ownerAddress↑,
489             msg.sender,
490             tokenAddress↑,
491             tokenId↑,
492             tokenPrice,
493             paymentToken
494         );
495
496         transferCoins(payment, true);
497
498         IPreSale721(tokenAddress↑).countTokensBought(eventId↑, msg.sender);
499
```

```
514         transferTokens(tokenAddress↑, tokenId↑, ownerAddress↑, msg.sender, quantity↑, false);
515
516         PaymentInfo memory payment = PaymentInfo(
517             ownerAddress↑,
518             msg.sender,
519             tokenAddress↑,
520             tokenId↑,
521             quantity↑ * tokenPrice,
522             paymentToken
523         );
524
525         transferCoins(payment, true);
526
527         IPreSale1155(tokenAddress↑).countTokensBought(msg.sender, tokenId↑, quantity↑, eventId↑);
```

Remediation

Follow Checks Effects Interaction pattern, Call countTokensBought() before sending tokens using transferTokens() on L485 and L514.

Status

Resolved

Medium Severity Issues

B.2 Possible frontrunning

Description

Frontrunning can happen in the case of AUCTION where there are two users one with higher and one with lower bid for the token(s) that seller wants to sell, now user with higher bid only should be able to claim the auction logically but user/buyer with lower bid claims the auction before other one, leaving no chance to the user who had higher bid.

The require check on L310 in else block allows buyer address (address other than seller address) to call the completeOrder() in the case of auction.

Remediation

looking at the above case where one “interested buyer with low bid value can frontrun (or simply buy before) the buyer with high bid”, consider not allowing any other address other than seller to call the function in the case of auction (by modifying a require() on L310).
In this case seller can accept the order by looking at the offers made by buyers.

Status

Resolved

B.3 Possible frontrunning 2

Description

cancelOrder() takes two parameters user order and user signature, in this case if someone wants to cancel the order created by them they need to pass the created order and signature to cancelOrder() function in this case this transaction can be frontrunned by the the attacker that can then use the order signature to buy some tokens according to order that seller wanted to cancel. It can also happen in the condition where buyer wanted to cancel the order sign but malicious seller frontruns the transaction and sells the tokens to buyer.

Remediation

User should only sign a signature after making sure the order they are signing. Looking at smart contract logic it is difficult to suggest a specific mechanism to follow. for e.g In the scenario of commit and reveal scheme it is needed to stop some functionality of contract while commit and reveal phase which is not possible here looking at marketplace's usecase. We encourage development team to research on more safe way to use signatures

Status

Resolved

B.4 Cancel the used order/signature

Description

In some cases orderIsCancelledOrCompleted is not getting set to true for both buyer and seller, e.g on L346 seller order hash will get set to true for that address only if listing type is FLOOR_PRICE_BID and on L352 buyer order hash gets set to true for that buyer user only if msg.sender is _sellerOrder.user. In the case when listing type is AUCTION and when msg.sender is buyer address then code on L347 if block won't execute and code on L352 if block won't execute. which may not set order hash for the specific address to true in orderIsCancelledOrCompleted. In the case of AUCTION where signatures from both the sides are needed and used in this function, it is necessary to set them completed by setting orderIsCancelledOrCompleted to true for the buyer order hash for buyer address and seller order hash for seller address.

Falling to set any used order/order hash to true may increase the chances of using used orders and signature for that order again maliciously.

Remediation

Set orderIsCancelledOrCompleted value to true for user address to order hash (orderIsCancelledOrCompleted[user address][order hash]) for every used signature in completeOrder(), which should be then used to check that the orderhash is not used already.

Status

Resolved

B.5 Condition is prone to unexpected results:

Description

checkOrdersCompatibility() contains require checks to check if token addresses are same.

The require check on L1029 checks that for "_buyerOrder.tokenAddress == _sellerOrder.tokenAddress" or "_buyerOrder.listingType == ListingType.FLOOR_PRICE_BID" this means condition can be passed if listing type is FLOOR_PRICE_BID and token addresses in both the orders are not same, which can be maliciously used by seller to sell token on other token address which can be different than what buyer wanted to buy.

Remediation

Review the code logic and remove “_buyerOrder.listingType == ListingType.FLOOR_PRICE_BID” condition.

Status

Resolved

B.6 Use modifier like nonReentrant

Description

Marketplace contract is using some functions to send tokens, native tokens using call() and overall contains many external calls. some important public/external completeOrder,prePurchase functions where users can provide inputs as token addresses, some used contract addresses can have hooks which can be used by attacker to reenter if some malicious users would be involved in the process e.g reentering while completeOrder() calls transferTokens() to send NFTs to msg.sender and msg.sender reenters to call completeOrder, In the case where seller order hash doesn’t gets set to true (if amount left is not 0) in orderIsCancelledOrCompleted for ERC1155 sale it doesnt cancels the signature which can give chance to reenter.
Looking at these cases we recommend to use modifiers like OpenZeppelin nonReentrant to protect from possible reentrancies.

Remediation

Use modifier like OpenZeppelin nonReentrant on completeOrder() and prePurchase() for extra security

Status

Resolved

Low severity Issues

B.7 Deadline/expiry should be always checked

Description

If "msg.sender == _buyerOrder.user" then it is gettting checked that "block.timestamp > _sellerOrder.deadline" on L299 which checks for buyer user only claim after auction ends. but it does not ensures the expiry of _sellerOrder.deadline. There can be the case where the seller wants their order should be expired after some time.

The severity of this issue depends on usecase.

Remediation

Check can be added to ensure that the current timestamp is not more than “_sellerOrder.deadline + some time” when buyer would be claiming after the auction. (in this case buyer needs to claim after deadline and before “deadline + some time limit” otherwise buyer may not able to claim because of this check)

Status

Acknowledged

Decrypt team’s comment: Deadline will always be checked before calling the smart contract at dApp level. Also their is a way for seller to cancel his/her order.

B.8 Missing check for zero address

Description

There are multiple functions in the contract which are missing zero address validation. Adding a zero address check is necessary because, a zero address is something to which if any funds or tokens are transferred, it can not be retrieved back. it is recommended to add a check for zero address.
- initialize
-setAdminAddress

Remediation

Consider adding a require statement that validates input against zero address to mitigate the same.

Status

Resolved

B.9 If seller already owns some quantity of ERC1155 tokens and if he wants to sell more quantity then functions mint all new tokens to buyer

Description

transferTokens and transferBundle functions lazy mints all the ERC1155 tokens quantity instead of transferring the amount that are current tokens if the token quantity they wanted to transfer is greater than the balance of seller address

Example (In the case of transferToken):
If address A holds 5 ERC1155 tokens of ID 6 (Here A is owner of this ERC1155 contract which tokens he wants to sell), And wants to sell 10 tokens. Then he creates the order with quantity of 10 tokens.

The buyer B comes to buy 10 tokens from A, while transferring the tokens the code will check if “shouldLazyMint” returns true on line 587 and will lazy mint the tokens to buyer if “shouldLazyMint” returns true. The shouldLazyMint will return true if owner address is owner of the token contract and owner address holds less quantity/balance for the token ID (ID 6 in this case) that he wants to sell.

In this case it will mint new 10 tokens instead of transferring 5 tokens of ID 6 that owner holds and minting only 5 new tokens

Remediation

there should be a logic that will check how many tokens needs to be lazy minted and other tokens that seller holds will get transferred instead of lazy minting all the token quantity.

Status

Resolved

B.10 Care needs to be taken with ERC20 tokens.

Description

It can be seen from the test files that this project is using oz erc20 implementation which reverts on failure. but there can be some situations where contract's functionality may not work as expected e.g when payment token doesnt reverts on failure and returns false in these cases for these type situations libraries like OZ SafeERC20 can be used to wrap erc20 operations.
In the case of token which deduct fees on transfer the actual amount that the smart contract will transfer won't get transferred to destination addresses as there would be fees on the transferred amount for these type of situations we recommend limiting the uses of specific type of erc20 tokens for which contract is written and to perform more unit testing for these type of cases.

Recommendation

Review the code and follow recommendations mentioned in description.

Status

Resolved

Informational Issues

No issues found

C. Contract - CustomERC721.sol

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

No issues found

D. Contract - PreSale721.sol

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

No issues found

Informational Issues

No issues found

E. Contract - SimpleERC721.sol

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

No issues found

Informational Issues

No issues found

F. Contract - SimpleERC721Deployer.sol

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

No issues found

Informational Issues

No issues found

G. Contract - ExtendedERC721.sol

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

No issues found

Informational Issues

No issues found

H. Contract - ExtendedERC721Deployer.sol

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

No issues found

Informational Issues

No issues found

I. Contract - CustomERC1155.sol

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

No issues found

Informational Issues

No issues found

J. Contract - PreSale1155.sol

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

No issues found

Informational Issues

No issues found

K. Contract - SimpleERC1155.sol

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

No issues found

Informational Issues

No issues found

L. Contract - SimpleERC1155Deployer.sol

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

No issues found

Informational Issues

No issues found

M. Contract - ExtendedERC1155.sol

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

No issues found

Informational Issues

No issues found

N. Contract - ExtendedERC1155Deployer.sol

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

No issues found

Informational Issues

No issues found

O. Contract - CreatorV1.sol

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

O.1 Anyone can create token contracts

Description

There are no limitations/restrictions on CreatorV1's deploySimpleERC721, deployExtendedERC721, deploySimpleERC1155, deployExtendedERC1155. while specification document shared by Decrypt team shows the same (we have decided to keep it as low severity) we highlight this again that anyone can use deployment feature of CreatorV1 as there are no access control restrictions on deploySimpleERC721, deployExtendedERC721, deploySimpleERC1155, deployExtendedERC1155.

Remediation

Consider reviewing the business logic

Status

Acknowledged

Informational Issues

No issues found

P. Contract - RoyaltyDistribution.sol

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

No issues found

Informational Issues

P.1 Redundant function

Description

Redundant method getTokenRoyaltyDistribution() for getting tokenCollaboratorsRoyaltyShare values. tokenCollaboratorsRoyaltyShare mapping is public mapping and because of which the compiler creates a getter method for this. Which can be used to get values instead of declaring new method

Remediation

Consider reviewing the code and remove the redundant function

Status

Resolved

Auditor's comment: tokenCollaboratorsRoyaltyShare mapping scope changed to private and getTokenRoyaltyDistribution() is used as getter.

P.2 setTokenRoyaltyDistribution deletes previous distributions

Description

setTokenRoyaltyDistribution() is getting used to set shares to collaborators for the tokenid , it is necessary to note that calling setTokenRoyaltyDistribution() deletes the previously set shares for addresses for that token id.

Status

Acknowledged

Functional Testing

- ✓ Should revert when non-owner tries to upgrade contract
- ✓ Should revert when the function call is not called through delegateCall
- ✓ Should revert when start time exceeds end time.
- ✓ Should revert if createPresaleEvent is called by non-owner
- ✓ Signature signer is getting checked for the order if someone uses signed signature for
- ✓ order from the other user.
- ✓ prePurchase() Should follow Checks Effects Interactions
- ✓ Cancel used order signatures

Automated Tests

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

Closing Summary

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

Some issues of High, Mideum, 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 audit is not a security warranty, investment advice, or an endorsement of the DecryptMarketplace Platform. This audit does not provide a security or correctness guarantee of the audited smart contracts.

The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the DecryptMarketplace Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

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.

700+
Audits Completed

$15B
Secured

700K
Lines of Code Audited

Follow Our Journey

Audit Report
December, 2022

For

Blockchain Australia®

NEO VIRTUAL TECH

NEO LEGAL

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com