QuillAudits

Audit Report
September, 2023

For

VIRTUALNESS

Table of Content

Table of Content

Table of Content

Table of Content

Table of Content

Table of Content

Table of Content

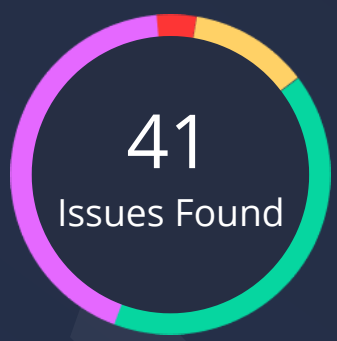
Executive Summary

Project Name	VirtualnessV1
Project URL	*https://virtualness.io/*
Overview	VirtualnessV1 is a nft marketplace for ERC721 and ERC1155 tokens. Virtualness also allows lazy minting of tokens for ERC721Virtualness and ERC1155Virtualness. The protocol uses orders and signed order signature to fulfill the minting of tokens from seller to buyer.
Audit Scope	*https://github.com/virtualness-io/Virtualness-protocol-contracts/tree/feat-audit/contracts*
Contracts in Scope	contracts/ERC721-Common/ERC721Roles.sol contracts/ERC721-Common/ERC721Virtualness.sol contracts/ERC1155-Common/ERC1155Roles.sol contracts/ERC1155-Common/ERC1155Virtualness.sol contracts/Lazy-Mint/erc-1155/IERC1155LazyMint.sol contracts/Lazy-Mint/erc-1155/LibERC1155LazyMint.sol contracts/Lazy-Mint/erc-721/IERC721LazyMint.sol contracts/Lazy-Mint/erc-721/LibERC721LazyMint.sol contracts/protocol-v1/lib/ExecutionDelegate.sol contracts/protocol-v1/lib/ProtocolEIP712.sol contracts/protocol-v1/lib/OrderStructs.sol contracts/protocol-v1/OrderMatch.sol contracts/protocol-v1/VirtualnessProtocolV1.sol contracts/protocol-v1/OrderValidator.sol contracts/protocol-v1/TransferExecutor.sol contracts/protocol-v1/ProtocolV1Core.sol
Commit Hash	ce827776570e3655c194cf54779c3b891eabd02e
Language	Solidity
Blockchain	Polygon
Method	Manual Analysis, Functional Testing, Automated Testing
Review 1	24 July 2023 - 23 August 2023

Executive Summary

Updated Code Received	*https://github.com/virtualness-io/Virtualness-protocol-contracts/commit/1bdd83650a12bb1a91f6a56a2d6aecb65d8d28b0*
Review 2	7 September 2023 - 12 September 2023
Fixed In	1bdd83650a12bb1a91f6a56a2d6aecb65d8d28b0

Number of security issues per severity



High Medium Low Informational

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	2	9	9
Partially Resolved Issues	1	0	1	0
Resolved Issues	0	3	7	9

Checked Vulnerabilities

- Access Management
- Arbitrary write to storage
- Centralization of control
- Ether theft
- Improper or missing events
- Logical issues and flaws
- Arithmetic Correctness
- Race conditions/front running
- SWC Registry
- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- Exception Disorder
- Gasless Send
- Use of tx.origin
- Malicious libraries
- Compiler version not fixed
- Address hardcoded
- Divide before multiply
- Integer overflow/underflow
- ERC's conformance
- Dangerous strict equalities
- Tautology or contradiction
- Return values of low-level calls
- Missing Zero Address Validation
- Private modifier
- Revert/require functions
- Multiple Sends
- Using suicide
- Using delegatecall
- Upgradeable safety
- Using throw

Checked Vulnerabilities

- Using inline assembly
- Style guide violation
- Unsafe type inference
- Implicit visibility level

Techniques and Methods

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

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

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

Structural Analysis

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

Static Analysis

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

Code Review / Manual Analysis

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

Gas Consumption

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

Tools and Platforms used for Audit

Remix IDE, Truffle, Solhint, Mythril, Slither, Solidity static analysis.

Types of Severity

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

High Severity Issues

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

Medium Severity Issues

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

Low Severity Issues

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

Informational

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

Types of Issues

Open

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

Resolved

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

Acknowledged

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

Partially Resolved

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

A. Contract - ProtocolV1Core

NOTE - When the collections which do not have lazy Minted functionality will be listed then fulfillOrder() function will/should be used instead of the fulfillLazyMintOrder() function.

High Severity Issues

A.1 User can use mintData created and signed for different order

Description

A mint data signed for one order can be used for another order. If sell.order.trader and sell.order.collection.tokenId are matching with mintData.creators[0].account and mintData.tokenId respectively then other mintData can be used for another order for which that wasn't created.

Here because other data from the mintData like uri, supply etc is not getting checked (because the mintData is the way owner/minter provides it) user would be able to change the uri, supply to the other mintData's uri and supply if they use different mintData

Example:
1. UserA created mintData1 for the order1 and userB created mintData2 for order2 (it can be even for different collections).
2. Malicious buyer tries to execute the order1 with the mintData2.
3. While lazy minting the the token will get the uri and the supply of mintData2 which wasn't created for order1

The impact of this case varies as mentioned from assigning different uri to the different supply for different collections with the same token id and trader.

Remediation

Add forOrder or similar field into the mintData which would be the hash of the order for which this mintData is getting created , in fulfillLazyMintOrder() it can be checked that the order that is getting used for mintData has the same hash that is mentioned in the mintData.

Additionally the collection field can be mentioned in the mintData which would be the address of collection. And can be checked in the fulfillLazyMintOrder() function to be equal to the sell order's collection.

Status

Partially Resolved

A.1 User can use mintData created and signed for different order

Auditor's comment

Because of the use case of using one mintData for two or more different orders it is not possible to use the order hash in mintData to validate as virtualness team notified us. The team has decided to go with use of collection address for validation which will check that the order's collection address is equal to the mintdata's collection. But with this approach there should not be the different mintData for the same tokenId (as discussed with the developer team).

Virtualness team's comment

There will be only one MintData for a tokenID.

Medium Severity Issues

A.2 Unable to transfer royalty fee while lazyminting

Line

233 - 260

Function - matchAndLazyMint

```
251             unchecked {
252                 totalPrice = price * amount;
253             }
254
255             transferFunds(sell.trader, msg.sender, sell.collection.paymentToken, sell.fees, totalPrice, sell.collection.collectior
256
257             tokenLazyMint(sell.collection.collection, sell.trader, buy.recipient, amount, sell.collection.assetType, mintData);
258
259             return true;
260         }
261
```

Description

While lazy minting tokens with fulfillLazyMintOrder() , in matchAndLazyMint() function transferFunds() and tokenLazyMint() gets executed.
But here while executing transferFunds() the "(address receiver, uint256 royaltyAmount) = IERC2981Upgradeable(collection).royaltyInfo(tokenId, price);" will return receiver as zero address and it won't send royaltyAmount to the receiver.
The reason the receiver would be zero address is because the royalty info for the token is getting set after the transferFunds() call i.e in tokenLazyMint().

A.2 Unable to transfer royalty fee while lazyminting

Remediation

Swap the call sequence of transferFunds() and tokenLazyMint(), so that tokenLazyMint() will set the royalty info and then transferFunds() would be able to send the royaltyAmount to the receiver.

Status

Resolved

A.3: royaltyInfo function call can fail for the token contracts which don't support/implement ERC2981

Description

transferFunds() calls royaltyInfo(tokenId, price) on collection contract to take receiver and royaltyAmount. In case any collection doesn't implement ERC2981 standard the royaltyInfo() function would be absent in that contract and the call will fail.

Remediation

For this call it should be checked first that the collection supports ERC2981 using ERC165's supportsInterface. If the collection supports the interface then only royaltyInfo() can be called on the collection address.

It needs to be noted that there can be a collection which supports ERC2981 but won't have registered support for ERC2981 in that case because of added functionality it won't be able call royaltyInfo.

Status

Resolved

Low Severity Issues

A.4 cancelOrder() doesn’t revert for already canceled order

Line

266

Function - cancelOrder

```
266         function cancelOrder(OrderParameters calldata order) public {
267             /* Assert sender is authorized to cancel order. */
268             require(msg.sender == order.trader);
269
270             bytes32 hash = _hashOrder(order, nonces[order.trader]);
271
272             if (!cancelledOrFilled[hash]) {
273                 /* Mark order as cancelled, preventing it from being matched. */
274                 cancelledOrFilled[hash] = true;
275                 emit OrderCancelled(hash);
276             }
277         }
278
```

Description

Lets say, Seller/trader wants to cancel the order he checks on the website/interface that the order isn’t executed yet and proceeds to cancel it. at that moment someone buys the order before it gets canceled i.e someone bought it before cancelOrder transaction executes.
So in this case the cancelOrder() won’t execute the code in the if block because cancelledOrFilled[hash] was set to true when the buyer bought it. So it will proceed execution without setting it to true again but also without reverting the transaction.

In this case it can happen that the seller will get false assurance that his transaction got canceled even though it was executed.

Adding require statement to revert transaction in this case will help for handling this situation as adding some check from the backend will can’t handle this situation e.g dapp backend checks that the cancelledOrFilled[hash] is false means the order is not filled yet but as mentioned in above example it can happen that the order will get filled once user executes cancelOrder() transaction.

A.4 cancelOrder() doesn’t revert for already canceled order

Remediation

Add require statement in cancelOrder so that for the filled order the transaction to cancel the order can be reverted.

Status

Resolved

A.5 Seller can manipulate the seller order before signing it

Description

While discussing with developers we came to know that the platform fees (fees array) in the seller order are getting added by the dapp backend (similar to other things that the backend would be adding in the mintOrder).

Here, malicious signer can manipulate the content that he is signing in some cases.

Remediation

Make sure that this is getting checked in the smart contract if the fee amount is something that the project team is expecting to be and not something manipulated. This can be achieved by taking a percentage of fees which can be set and changed in the protocol smart contract instead of taking it from the user.

Status

Acknowledged

Virtualness team’s comment

Seller won't be able to manipulate the sell order as Payload is controlled by the server side and signature is getting verified by the server.

A.6 validateOrderTime() allows expirationTime to be set to 0

Description

In the contract VirtualnessV1Core the function validateOrderTime() allows expiration time to be set to 0. It can happen that the order signature remains unexecuted for long time and someone will able to execute that order later where seller may don't want to execute that order for that price specified before.

Remediation

In order to not to let this happen you can remove the statement order.collection.expirationTime == 0 and add a statement to check expiration time is always greater than start time.

Status

Resolved

A.7 cancelOrder function should have error message

Description

In function cancelOrder() the require statement on L274 checks if message.sender is the trader who is going to cancel the order but there is no error message set if the msg.sender is not the trader so it will revert wihtout the error message.

Remediation

Make sure to add error message for the require statement on L274

Status

Resolved

A.8 Lack of test cases for the contract

Description

The contract VirtualnessV1Core lacks proper test cases. Having less test cases makes contract miss issues if there are any also it makes it hard for other devs to understand proper working and functionality of the contract.

A.8 Lack of test cases for the contract

Remediation

Make sure to add test cases for the functions and functions paths.

Status

Partially Resolved

Auditor's comment

Assertions should be written for all the tests to check expected state changes.

A.9: Limit for fees can be added

Description

Currently the fees have no limit so the fee rate can be anything. Fee limit can be added in the contract so that it can't exceed a certain limit. in this way the user would be assured about the fees that they are not getting exceeded from a certain limit.

Remediation

Add a fee limit for the rate of fees.

Status

Acknowledged

Auditor's comment

This issue is regarding adding the fee limit.

A.10: Pass receiveAmount instead of price

Description

Pass the receiveAmount instead of price in royaltyInfo().
As the order fee was transferred and receiveAmount was calculated (that is price - totalFee from _transferFees()) so user would be receiving receiveAmount amount, now while calculating royalty amount the royaltyInfo() will calculate royalty amount with price which would be more than the amount if that would be calculated with receiveAmount.

A.10: Pass receiveAmount instead of price

Remediation

Verify and change the parameter getting passed to royaltyInfo() as mentioned above if its not intentional.

Status

Acknowledged

Informational Issues

A.11: Unlocked pragma

Description

Contracts are using floating pragma (pragma solidity ^0.8.17), 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 gets selected while deploying a contract which has higher chances of having bugs in it.

Remediation

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

Status

Resolved

A.12: Use of fee on transfer token in contract

Description

It is quite possible that users can use a fee on transfer tokens for buying/selling. The issue with the token is that with each functionality(transfer) that happens it takes some fees out of it. Example if userA has 100 tokens he transfers to userB then userB will only get 95 tokens if the fee is 5%. In that way the seller and creator with royalties will get less than what is expected.

A.12: Use of fee on transfer token in contract

Remediation

To solve the issue you can only allow some whitelisted tokens to be used in the marketplace.

Status

Acknowledged

Virtualness team's comment

This is not applicable to us. We wont be using token which will charge any fee for transfer.

Auditor's comment

The protocol can be used for other collections and other orders can use fee-on-transfer payment tokens, in that case it should be noted.

A.13: Consideration for the scenario

Description

The team should note that the scenario like this can happen because of the logic of the code, where the user creates a second order while the first order is still there, e.g first user creates an order for 100 tokens and then again for 100 tokens while the old order is still there.

Remediation

This is just information that scenarios like this can happen and the team should take care of these things on the front end when necessary.

Status

Acknowledged

Virtualness team's comment

This is been controlled at Web2 layer.

A.14: Nonce is getting used as OrderId while emitting OrdersMatched

Description

nonce is getting used as OrderId while emitting OrdersMatched in matchAndTransfer() and matchAndLazyMint()

A.14: Nonce is getting used as OrderId while emitting OrdersMatched

Remediation

Verify and change the used parameter.

Status

Acknowledged

Virtualness team's comment

It is temporary for now. In Version2 we will have different storage for orderId.

B. Contract - ERC721Virtualness

High Severity Issues

No issues were found

Medium Severity Issues

B.1: Centralization in isApprovedForAll

Description

isApprovedForAll returns true for all the addresses which have a minter role. This can be problematic where a malicious person with minter role tries to transfer tokens from someone's (user's) account to another account using tansferFrom() as the malicious address would be able to transfer even without having approvals.

On the other side from the user's perspective users would like to have more control over their token approvals.

Remediation

Remove the code for making auto returning true for address with minter role.

Status

Acknowledged

Low Severity Issues

B.2: Add extra check for zero address signer

Description

It can happen that invalid signature may end up recovering invalid signer as the zero address. The used openzeppelin's ECDSAUpgradeable should revert in the case where it recovers zero address. But it's a good idea to have an extra check in place for handling this situation in lazyMint().

Remediation

Add require check for checking and reverting when signer would be zero address after calculating signer , before checking if the signer==owner() or minter address on the L 193. The added check should revert with a meaningful error message e.g invalid address.

Status

Acknowledged

Auditor's comment

This should be noted that the currently used library version has checks inplace but in future if version gets changed then the care should be taken ahead by adding checks incase there are no checks in the used library.

B.3: transferOwnership should be two step process

Description

In contract ERC721Virtualness there is function transferOwnership() for transferring the ownership to another user but it is just a single function which is prone to some issues if the address to which the ownership is to be transferred is not correct or is malicious.

Remediation

Please make sure to use 2 step ownership transfer which makes sure that in step 1 you set the address of the owner to be ownership is transferred and in step 2 the transfer happens and your ownership is revoked.

Status

Acknowledged

B.4: Call _disableInitializers() in constructor

Description

Add a constructor and call _disableInitializers() in the constructor of this contract,calling this in the constructor of a contract will prevent that contract from being initialized or reinitialized to any version. It is recommended to use this to lock implementation contracts that are designed to be called through proxies.

Remediation

Call function in constructor as suggested
https://docs.openzeppelin.com/contracts/4.x/api/proxy#Initializable-_disableInitializers–

Status

Resolved

Informational Issues

B.5: Unused parameter in LazyMint

Description

LazyMint() function takes a parameter but it's not getting used. Unused parameter should be removed.

Remediation

Remove unused parameter.

Status

Acknowledged

B.6: Unused event in contract

Description

Event Creators (uint256 indexed tokenId, LibPart.Part[] indexed creators) is not used in contract.

Remediation

Make sure to use the event in the appropriate function or remove unused event.

Status

Resolved

B.7: Care needs to be taken for the addresses while removing minter role

Description

In the scenario where address having minter role signs the mintData and later minter role of that address gets removed then while executing the order with that mintData the transaction can revert when lazyMint() checks the signer recovered is owner or the address having minter role, As at that moment the address won't have the minter role.

This scenario should be noted while performing actions such as removing minter role.

Remediation

Care should be taken in these types of scenarios.

Status

Acknowledged

B.8: Unused and commented code in the contract

Description

In the lazyMint() function there is some code on L209 which is commented out and is not used.

Remediation

Consider removing the unused code.

Status

Resolved

B.9: Unused imports

Description

ERC721Upgradeable and AccessControlUpgradeable are not getting inherited by ERC721Virtualness. Unused/uninherited imports can be removed.

Remediation

Please make sure to remove the import statement if not getting used.

Status

Resolved

C. Contract - ERC1155Virtualness

High Severity Issues

No issues were found

Medium Severity Issues

C.1: Centralization in isApprovedForAll

Description

isApprovedForAll returns true for all the addresses which have a minter role. This can be problematic where a malicious person with minter role tries to transfer tokens from someone's (user's) account to another account using tansferFrom() as the malicious address would be able to transfer even without having approvals.

On the other side from the user's perspective users would like to have more control over their token approvals.

Remediation

Remove the code for making auto returning true for address with minter role.

Status

Acknowledged

Low Severity Issues

C.2: Add extra check for zero address signer

Description

It can happen that invalid signature may end up recovering invalid signer as the zero address. The used openzeppelin's ECDSAUpgradeable should revert in the case where it recovers zero address. But it's a good idea to have an extra check in place for handling this situation in lazyMint().

Remediation

Add require check for checking and reverting when signer would be zero address after calculating signer , before checking if the signer==owner() or minter address on the L 218. The added check should revert with a meaningful error message e.g invalid address.

C.2: Add extra check for zero address signer

Status

Acknowledged

Auditor's comment

This should be noted that the currently used library version has checks inplace but in future if version gets changed then the care should be taken ahead by adding checks incase there are no checks in the used library.

C.3: Absence of setBaseURI

Description

Base Uri is not getting set in the initialize function , and the contract is doesn't has internal setBaseURI() from ERC1155URIStorageUpgradeable exposed with public function. So that authorized address can call it .

While setting uri for tokenIds the _setURI is getting used, where full uri needs to be set.

Remediation

Verify that it is intentional to have baseUri uninitialized and it is intentional to set full token uris directly with _setURI. In this case the issue can be acknowledged.

If its against the intentional logic then add external setBaseURI() function which will call internal _setBaseURI(). In this case the external function must have an access modifier. And should be called only once at the start.

Status

Acknowledged

Virtualness team's comment

This is intentional.

C.4: total variable is not getting used

Description

In function _saveCreators() where creators of the collection are getting assigned and stored. The variable total is used to calculate value/share but the total variable then is not used in any of the other calculations making it redundant.

Remediation

If the variable is supposed to be used then please do the calculation carefully otherwise remove the variable altogether.

Status

Acknowledged

C.5: Call _disableInitializers() in constructor

Description

Add a constructor and call _disableInitializers() in the constructor of this contract,calling this in the constructor of a contract will prevent that contract from being initialized or reinitialized to any version. It is recommended to use this to lock implementation contracts that are designed to be called through proxies.

Remediation

Call function in constructor as suggested
https://docs.openzeppelin.com/contracts/4.x/api/proxy#Initializable-_disableInitializers–

Status

Resolved

Informational Issues

C.6: Unused parameter in LazyMint

Description

LazyMint() function takes a parameter but it's not getting used. Unused parameter should be removed.

Remediation

Remove unused parameter.

Status

Acknowledged

C.7: _saveSupply() function should have error message

Description

In function _saveSupply() there is a require statement on L247 stating supply of token should be zero but it does not have a meaningful error message if the function is called on the same token id again.

Remediation

Please make sure to add an error message for the require statement.

Status

Resolved

C.8: Care needs to be taken for the addresses while removing minter role

Description

In the scenario where address having minter role signs the mintData and later minter role of that address gets removed then while executing the order with that mintData the transaction can revert when lazyMint() checks the signer recovered is owner or the address having minter role, As at that moment the address won’t have the minter role.

This scenario should be noted while performing actions such as removing minter role.

C.8: Care needs to be taken for the addresses while removing minter role

Remediation

Care should be taken in these types of scenarios.

Status

Acknowledged

C.9: Check the fixed supply of token logic

Description

(This issue is for information regarding implemented logic and can be acknowledged if this is the intentional logic)
Code implements logic for keeping supply limited according to what it gets assigned first. mint_() checks for the case where the amount exceeds the defined supply while minting the tokenId amount. And because of this in mint(), delegateMint() and in lazyMint() it restricts the supply from exceeding the supply amount that was set earlier.

Remediation

Verify that this is the intentional functionality.

Status

Acknowledged

Virtualness team's comment

Yes, it is intentional logic.

C.10: Unused imports

Description

In the contract ERC1155Upgradeable.sol is not getting inherited.

Remediation

Please make sure to remove the import statement if not getting used.

Status

Acknowledged

C.11: Unused and commented code in the contract

Description

In the lazyMint() function there is some code on L187 and L197 which is commented out and is not used.

Remediation

Consider removing the unused code.

Status

Resolved

D. Contract - TransferExecutor

High Severity Issues

No issues were found

Medium Severity Issues

No issues were found

Low Severity Issues

D.1: Use call{} instead of send for eth transfer

Line

124

Function - transferTo

```
123
124             if (paymentToken == address(0)) {
125                 /* Transfer funds in ETH. */
126                 (bool success ) = payable(to).send(amount);
127
128                 require(success, "Native token transfer failed");
129             } else if (paymentToken_[paymentToken] == 1) {
130                 /* Transfer funds. */
```

Description

In function _transferTo() send() is used to transfer eth which is not the recommended way. The recommended way to transfer eth is to use call{}. The reason being send and transfer both uses 2300 gas for transferring but because there can be changes in gas in future the gas limit for both those options might change which will result in failure of the eth transfer.

Remediation

Please make sure to to use call{}
Ex., payable(user).call{value: msg.value}(“”);

Status

Resolved

Informational Issues

No issues were found

E. Contract - OrderStruct

High Severity Issues

No issues were found

Medium Severity Issues

No issues were found

Low Severity Issues

E.1: Variable is set but not used

Description

In the struct OrderParameters isLazy boolean value is used and it is getting used in _hashOrder() function in ProtocolEIP712 contract but is not used for checking if the nft is minted using lazyMint function. If it's true then the order will be fulfilled with fulfillLazyMintOrder() function otherwise by fulfillOrder() function but as the variable is not used it might create confusion.

Remediation

Remove the unused variable if not used. If this variable is getting used for frontend to handle the orders then this issue can be acknowledged.

Status

Acknowledged

Virtualness team's comment

This is a reference for front-end and backend, Also we may use it in the coming versions.

Informational Issues

No issues were found

F. Contract - VirtualnessProtocolV1

High Severity Issues

No issues were found

Medium Severity Issues

No issues were found

Low Severity Issues

F.1: Call _disableInitializers() in constructor

Description

Add a constructor and call _disableInitializers() in constructor of this contract,calling this in the constructor of a contract will prevent that contract from being initialized or reinitialized to any version. It is recommended to use this to lock implementation contracts that are designed to be called through proxies.

Remediation

Call function in constructor as suggested
https://docs.openzeppelin.com/contracts/4.x/api/proxy#Initializable-_disableInitializers–

Status

Resolved

Informational Issues

F.2: Redundant import

Description

Initializable is getting imported but UUPSUpgradeable already imports Initializable which makes it redundant.

Remediation

Remove the import statement.

Status

Resolved

G. Contract - ERC721Roles

High Severity Issues

No issues were found

Medium Severity Issues

No issues were found

Low Severity Issues

No issues were found

Informational Issues

G.1: Redundant abicoder declaration

Description

pragma abicoder v2; getting declared , but ABI coder v2 is activated by default from 0.8.0 version. As can be seen here https://docs.soliditylang.org/en/v0.8.0/080-breaking-changes.html

Remediation

Remove “pragma abicoder v2;” as it is already activated by default in Solidity v0.8.0 according to solidity documentation.

Status

Resolved

H. Contract - ERC1155Roles

High Severity Issues

No issues were found

Medium Severity Issues

No issues were found

Low Severity Issues

No issues were found

Informational Issues

H.1: Redundant abicoder declaration

Description

pragma abicoder v2; getting declared , but ABI coder v2 is activated by default from 0.8.0 version. As can be seen here https://docs.soliditylang.org/en/v0.8.0/080-breaking-changes.html

Remediation

Remove “pragma abicoder v2;” as it is already activated by default in Solidity v0.8.0 according to solidity documentation.

Status

Resolved

I. Contract - IERC1155LazyMint

High Severity Issues

No issues were found

Medium Severity Issues

No issues were found

Low Severity Issues

No issues were found

Informational Issues

No issues were found

J. Contract - IERC721LazyMint

High Severity Issues

No issues were found

Medium Severity Issues

No issues were found

Low Severity Issues

No issues were found

Informational Issues

No issues were found

K. Contract - LibERC1155LazyMint

High Severity Issues

No issues were found

Medium Severity Issues

No issues were found

Low Severity Issues

No issues were found

Informational Issues

No issues were found

L. Contract - LibERC721LazyMint

High Severity Issues

No issues were found

Medium Severity Issues

No issues were found

Low Severity Issues

No issues were found

Informational Issues

No issues were found

M. Contract - ExecutionDelegate

High Severity Issues

No issues were found

Medium Severity Issues

M.1 Should use SafeERC for critical operations in contract

Line

76

Function - transferTo

```
76          function transferERC20(address token, address from, address to, uint256 amount)
77              internal
78              returns (bool)
79          {
80              return IERC20(token).transferFrom(from, to, amount);
81          }
```

Description

TransferExecutor:transferFunds() function uses _transferTo() to transfer tokens “from” to “to” address. Here _transferTo() calls transferERC20() from the ExecutionDelegate contract.

But in transferERC20() it can happen that the transferFrom() will fail to transfer tokens (e.g in case of not having enough tokens in account). In this case if the token contract is a non reverting token (which doesn’t revert on failure and returns bool value) the boolean status value is not getting checked after that call.

So it can happen that the malicious buyer won’t have anything in the account to transfer and would be able to buy tokens for free (i.e without spending tokens to buy NFT)

Remediation

Make sure to use SafeERC for the ERC20 token operations.

Status

Resolved

Low Severity Issues

No issues were found

Informational Issues

No issues were found

N. Contract - ProtocolEIP712

High Severity Issues

No issues were found

Medium Severity Issues

No issues were found

Low Severity Issues

No issues were found

Informational Issues

No issues were found

O. Contract - OrderMatch

High Severity Issues

No issues were found

Medium Severity Issues

No issues were found

Low Severity Issues

No issues were found

Informational Issues

No issues were found

P. Contract - OrderValidator

High Severity Issues

No issues were found

Medium Severity Issues

No issues were found

Low Severity Issues

No issues were found

Informational Issues

No issues were found

Functional Tests

Some of the tests performed are mentioned below:

ProtocolV1Core

- ✓ Owner should be able to open
- ✓ Owner should be able to close
- ✓ User should be able to use fulfillOrder for selling and buying already minted token(s)
- ✓ User should be able to use fulfillLazyMintOrder for selling and buying through lazy minting
- ✓ Seller should be able to cancel the order
- ✓ Seller should be able to increment the counter to cancel all orders by changing the nonce
- ✓ Should revert while validating the order if the buyer tries to buy more than what the seller wants to sell
- ✓ Should if order has reached the expiration time
- ✓ Should revert if sell signature is invalid while recovering the signer

ERC721Virtualness

- ✓ Should be able to mint tokens
- ✓ Should be able to delegate mint tokens
- ✓ Should be able to lazy mint tokens
- ✓ Owner should be able to add the minter
- ✓ Owner should be able to remove the minter

ERC1155Virtualness

- ✓ Should be able to mint tokens
- ✓ Should be able to delegate mint tokens
- ✓ Should be able to lazy mint tokens
- ✓ Reverts if it tries to mint more than supply defined
- ✓ Owner should be able to add the minter
- ✓ Owner should be able to remove the minter

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 VirtualnessV1. We performed our audit according to the procedure described above.

Some issues of 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 VirtualnessV1 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 VirtualnessV1 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 VirtualnessV1 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

800K
Lines of Code Audited

Follow Our Journey

Audit Report
September, 2023

For

VIRTUALNESS

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com