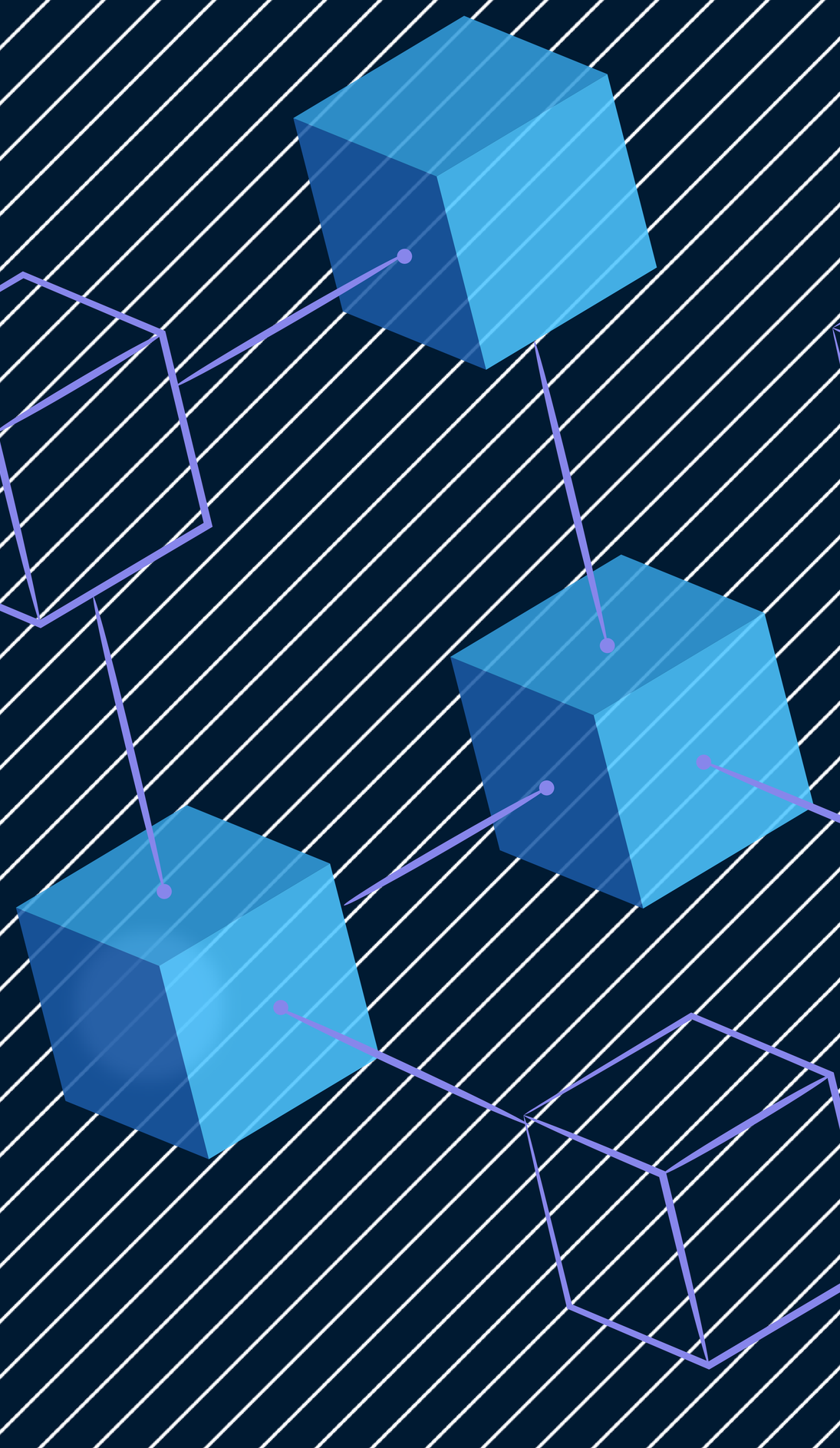
QuillAudits

Audit Report
January, 2022



For

unicus

Contents

Contents

Scope of the Audit

The scope of this audit was to analyze and document the Unicus smart contract codebase for quality, security, and correctness.

Checked Vulnerabilities

We have scanned the smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered:

- 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
- ERC20/721 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 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-20/721 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, Theo.

Issue Categories

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.

Risk-level	Description
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.

Number of issues per severity

Type	High	Medium	Low	Informational
Open	0	0	0	0
Acknowledged	0	0	4	10
Closed	0	2	6	0

Introduction

During the period of **29th Nov 2021 to 26th December 2021** - QuillAudits Team performed a security audit for **Unicus** smart contracts.

The code for the audit was taken from the following official link:

https://testnet.bscscan.com/address/0x73301F0f26042C563e986792044854E3e9621455#code

https://testnet.bscscan.com/address/0xe14051a7eAbBA69136fdf00198fa98933249203A#code

https://testnet.bscscan.com/address/0x1cab4dE7735094630104576C2C9142AEF593a205#code

Latest commit id: 19cbd28da2e6bce988c09072a61c9e2250d04e8d

Issues Found

A. Contract – Mint

Number of security issues per severity – Mint contract

Type	High	Medium	Low	Informational
Open	0	0	0	0
Acknowledged	0	0	2	2
Closed	0	0	2	0

High severity issues

No issues were found.

Medium severity issues

No issues were found.

Low severity issues

A.1 Outdated Compiler Version (SWC 102) → All solidity files contract→ MintNft

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

/**
```

Description

Using an outdated compiler version can be problematic especially if there are publicly disclosed bugs and issues that affect the current compiler version.

Remediation

It is recommended to use a recent version of the Solidity compiler, which is Version 0.8.9.

Status: Closed

A.2 Public function that could be declared external contract→ MintNft

A function with a **public** visibility modifier that is not called internally. Changing the visibility level to **external** increases code readability. Moreover, in many cases, functions with **external** visibility modifiers spend less gas compared to functions with **public** visibility modifiers.

The functions defined in the file, which are marked as public, are below

- getApproved
- "safeTrasnferFrom"
- renounceOwnership

However, it is never directly called by another function in the same contract or in any of its descendants. Consider marking it as "external" instead.

Recommendations
Use the external functions never called from the contract via internal call. Reading Link.

Status: Closed

A.3 Costly Loop contract→ MintNft

Description
The loop in the contract includes state variables like the .length of a non-memory array in the condition of the for loops.
As a result, these state variables consume a lot more extra gas for every iteration of the for a loop.

```
    return (NftId);
}
// function to mint multiple nfts
function batchMint(uint256 _totalNft, string[] memory _uri, uint256[] memory _royalty) external nonRe
    require(_totalNft <= 15, "Minting more than 15 Nfts are not allowed");
    require(_totalNft == _uri.length, "Total Uri and TotalNft does not match");

    for(uint i = 0; i< _totalNft; i++) {
        require(_royalty[i]<100,"Royalty cannnot be 100 or more");
        _mintNft(msg.sender, _uri[i], _royalty[i]);
    }

    emit BatchMint(_totalNft, "Batch Mint Success");
    return true;
}
```

The below functions include such loops at the above-mentioned lines:
- batchMint
- _getCurrentSupply

Recommendation
It's quite effective to use a local variable instead of a state variable like .length in a loop.
For instance,

```
uint256 local_variable = _groupInfo.addresses.length;
 for (uint256 i = 0; i < local_variable; i++) {
          if (_groupInfo.addresses[i] == msg.sender) {
              _isAddressExistInGroup = true;
              _senderIndex = i;
              break;
          }
       }
```

Reading reference link: https://blog.b9lab.com/getting-loopy-with-solidity-1d51794622ad

Status: Acknowledged

A.4 Overflow issue with few functions contract→ MintNft

Description
There is a required check needed for a few functions to avoid the overflow condition. List of functions are
- royaltyForToken()
- UpdateTokenURI()
- UpdateTokenURI

Recommendations
Put the required check to avoid the overflow resulting in a Denial of Service attack. It's recommended to use the mint function only for the owner. Few examples can be found here at this link.

Status: Acknowledged

Informational issues

1. Upgradeable contract was missing. Smart contracts deployed using OpenZeppelin Upgrades Plugins can be upgraded to modify their code while preserving their address, state, and balance. This allows you to iteratively add new features to your project, or fix any bugs you may find in production. Link. Later after a discussion with the client, they suggested switching to a new contract isn't in their plan. So we acknowledged this issue.

2. Linting issues were found

```
mint.sol
    93:2  error  Line length must be no more than 120 but current length is 129  max-line-length
    94:2  error  Line length must be no more than 120 but current length is 136  max-line-length
   176:2  error  Line length must be no more than 120 but current length is 136  max-line-length
   202:2  error  Line length must be no more than 120 but current length is 122  max-line-length
   288:2  error  Line length must be no more than 120 but current length is 160  max-line-length
   306:2  error  Line length must be no more than 120 but current length is 156  max-line-length
   767:2  error  Line length must be no more than 120 but current length is 136  max-line-length
   812:2  error  Line length must be no more than 120 but current length is 136  max-line-length
  1167:2  error  Line length must be no more than 120 but current length is 129  max-line-length

✖ 9 problems (9 errors, 0 warnings)
```

Functional Tests

We did functional tests for different contracts manually. Below is the report:

- function UpdateTokenURI → PASS → Able to Update Token URI
- function batchMint → PASS → function to mint multiple nfts
- Function getTokenCounter → PASS → Returns the total amount of NFT minted
- Function _setTokenURI → PASS → set URI of Token
- Function royalty of Token → PASS → Returns royalty of Token
- Function batchMint → PASS → function to mint multiple NFTs
- Function trasnferOwnership → PASS → transfer the ownership to the new owner
- Function renounceownership → PASS → Renounce ownership

B. Contract – NFTAuction

Number of security issues per severity – NFTAuction

Type	High	Medium	Low	Informational
Open	0	0	0	0
Acknowledged	0	0	0	7
Closed	0	2	4	0

High severity issues

No issues were found.

Medium severity issues

B.1 No condition when two bidding amount is same

Description

If there are two bidders- Alice and Bob and say Alice is the highest bidder with bid amount 0.1 BNB and then Bob comes and places the bid for the same amount i.e. 0.1 BNB, Bob wins the auction. But this should not be the case, as the auction should be won by the bidder who first bids the highest amount. Otherwise an attacker on knowing the highest bid price can bid for the same price at the closing moments of the auction, consequently hurting both the auction creator and the true highest bidder(i.e. Alice here).

Remediation

On line: 1562 in placeBid() function, remove the equality sign from the required statement and change it to strictly greater than sign.

Status: Closed

B.2 Funds get stuck

Description
There is a condition when the auction ends. When the auction ends, and the NFT owner doesn't end the auction then the highest bidder funds are locked until or unless the NFT owner doesn't cancel the auction.

Remediation
It is advised to allow the highest bidder to explicitly "claim" the NFT from his side at the end of the auction via a separate function (and also a separate function for the minter and seller to claim their funds).

Status: Closed

Low severity issues

B.3 Outdated Compiler Version (SWC 102) → All solidity files contract

```
23  */
24
25  // SPDX-License-Identifier: MIT
26  pragma solidity ^0.8.3;
27
28  interface IERC165 {
29      /**
30       * @dev Returns true if this contract implements the interface defined by
31       * `interfaceId`. See the corresponding
32       * https://eips.ethereum.org/EIPS/eip-165#how-interfaces-are-identified[EIP section]
33       * to learn more about how these ids are created.
```

Description
Using an outdated compiler version can be problematic especially if there are publicly disclosed bugs and issues that affect the current compiler version.

Remediation
It is recommended to use a recent version of the Solidity compiler, which is Version 0.8.9.

Status: Closed

B.4 Public function that could be declared external

Description
A function with a **public** visibility modifier that is not called internally. Changing the visibility level to **external** increases code readability. Moreover, in many cases, functions with **external** visibility modifiers spend less gas compared to functions with **public** visibility modifiers.

The functions defined in the file, which are marked as public, are below
- fetchauctionBid

However, it is never directly called by another function in the same contract or in any of its descendants. Consider marking it as "external" instead.

Remediation
Use the external functions never called from the contract via internal call. Reading Link.

Status: Closed

B.5 Costly Loop

Description
The loop in the contract includes state variables like the length of a non-memory array in the condition of the for loops.

As a result, these state variables consume a lot more extra gas for every iteration of the for a loop.

The below functions include such loops at the above-mentioned lines:
- batchMint

Remediation
It's quite effective to use a local variable instead of a state variable like .length in a loop.
For instance,

```
uint256 local_variable = _groupInfo.addresses.length;
 for (uint256 i = 0; i < local_variable; i++) {
        if (_groupInfo.addresses[i] == msg.sender) {
            _isAddressExistInGroup = true;
            _senderIndex = i;
            break;
        }
    }
```

Reading reference link: https://blog.b9lab.com/getting-loopy-with-solidity-1d51794622ad

Status: Closed

B.6 Cancel function shouldn't be called by auction creator once the auction ends

Description
The auction creator should not be able to call cancelAuction() once the auction ends, as it can result in unfair auction wherein the auction creator may choose to keep the NFT for himself after discovering the true valuation of the NFT in the market. Also the auction creator can use this info to create another auction for the same NFT and set the starting price to somewhere close to the highest bid of the last auction.

Status: Closed

Informational issues

1. Add and emit event for these functions carrying out critical operations- cancelAuction() and updateListingPrice(). This would be a best practice for offchain monitoring.

   Status: **Acknowledged**

2. Use safeTransferFrom instead of transferFrom at lines: 1528, 1582, 1590 and 1609, so that it gets checked first that contract recipients are aware of the ERC721 protocol to prevent tokens from being forever locked.

   Status: **Acknowledged**

3. Overflow of _value variable is possible for _auctionIds that is used from the Counter library. This is because the Counter library uses unchecked arithmetic. This means that on overflow, the require check in isValidId() modifier will fail for all the previous auctions which in turn would deny the usage of placeBid(), endAuction(), cancelAuction() and timeLeftForAuctionToEnd() functions. This can be pretty serious as the NFTs for the existing/ongoing auctions would be locked forever inside the NFTAuction contract.

   Status: **Acknowledged**

4. Block.timestamp is used in the contract. The timestamp does not always reflect the current time and may be inaccurate. The transactions to be included in a block can be influenced by miners. Thus, as this contract depends on the order of transactions and time, it is susceptible to front-running attack.

   Status: **Acknowledged**

5. There is unnecessary transfer of funds with zero value to the null adwhappens in the following scenario-

   Step 1- No one bids for the entire duration of the auction and the highest bidder is the null address itself.

   Step 2- The duration of the auction ends and the auction creator (who called the createAuction() function) calls the cancelAuction() function.

   Step 3- This results in an unnecessary transaction of 0 BNB to the null address.

   Status: **Acknowledged**

6. Add and emit event for these functions carrying out critical operations- cancelAuction() and updateListingPrice(). This would be a best practice for offchain monitoring.

   Status: **Acknowledged**

7. Solhint gives some code alignment recommendations

Status: **Acknowledged**

```
auction.sol
    87:2  error  Line length must be no more than 120 but current length is 129  max-line-length
    88:2  error  Line length must be no more than 120 but current length is 136  max-line-length
   170:2  error  Line length must be no more than 120 but current length is 136  max-line-length
   196:2  error  Line length must be no more than 120 but current length is 122  max-line-length
   282:2  error  Line length must be no more than 120 but current length is 160  max-line-length
   300:2  error  Line length must be no more than 120 but current length is 156  max-line-length
   761:2  error  Line length must be no more than 120 but current length is 136  max-line-length
   806:2  error  Line length must be no more than 120 but current length is 136  max-line-length
  1503:2  error  Line length must be no more than 120 but current length is 150  max-line-length
  1551:2  error  Line length must be no more than 120 but current length is 122  max-line-length
  1556:2  error  Line length must be no more than 120 but current length is 129  max-line-length
  1562:2  error  Line length must be no more than 120 but current length is 122  max-line-length
  1633:2  error  Line length must be no more than 120 but current length is 129  max-line-length

✖ 13 problems (13 errors, 0 warnings)
```

Functional Tests

We did functional tests for different contracts manually. Below is the report:

- function fetchauctionsBid → PASS → Able to fetch the Auction Bid
- function fetchMyNFTs → PASS → Returns auctions that a user has created
- function transferFundsToLastBidder → PASS → transfer funds to last bidder
- function transferFunds → PASS → transfers funds to seller and the minter gets royalty
- function updateListingPrice → PASS → function to update the listingPrice
- function fetchSoldAuctions → PASS →returns all thr nfts sold in the auction
- function timeLeftForAuctionToEnd → PASS →returns the blocktimestamp when auction will end and time left for auction to end
- function fetchUnsoldAuctions → PASS → Returns all unsold market auctions
- function cancelAuction → PASS → function to cancel auction
- function endAuction → PASS → function to end auction
- function placeBid → PASS → function to place bid

C. Contract – NFTMarket

Number of security issues per severity – NFTMarket

Type	High	Medium	Low	Informational
Open	0	0	0	0
Acknowledged	0	0	2	1
Closed	0	0	0	0

High severity issues

No issues were found.

Medium severity issues

No issues were found.

Low severity issues

C.1 Outdated Compiler Version (SWC 102)

Description

Using an outdated compiler version can be problematic especially if there are publicly disclosed bugs and issues that affect the current compiler version.

Remediation

It is recommended to use a recent version of the Solidity compiler, which is Version 0.8.9.

Status: Acknowledged

C.2 Costly Loop

Description

The loop in the contract includes state variables like the length of a non-memory array in the condition of the for loops.

As a result, these state variables consume a lot more extra gas for every iteration of the for a loop.

The below functions include such loops at the above-mentioned lines:
- batchMint

Remediation

It's quite effective to use a local variable instead of a state variable like .length in a loop.
For instance,

```
uint256 local_variable = _groupInfo.addresses.length;
 for (uint256 i = 0; i < local_variable; i++) {
            if (_groupInfo.addresses[i] == msg.sender) {
                _isAddressExistInGroup = true;
                _senderIndex = i;
                break;
            }
        }
```

Reading reference link: https://blog.b9lab.com/getting-loopy-with-solidity-1d51794622ad

Status: Acknowledged

Informational issues

1. Solhint reports are below

```
market.sol
    83:2  error  Line length must be no more than 120 but current length is 129  max-line-length
    84:2  error  Line length must be no more than 120 but current length is 136  max-line-length
   166:2  error  Line length must be no more than 120 but current length is 136  max-line-length
   192:2  error  Line length must be no more than 120 but current length is 122  max-line-length
   278:2  error  Line length must be no more than 120 but current length is 160  max-line-length
   296:2  error  Line length must be no more than 120 but current length is 156  max-line-length
   757:2  error  Line length must be no more than 120 but current length is 136  max-line-length
   802:2  error  Line length must be no more than 120 but current length is 136  max-line-length
  1568:2  error  Line length must be no more than 120 but current length is 131  max-line-length
  1588:2  error  Line length must be no more than 120 but current length is 141  max-line-length
  1591:2  error  Line length must be no more than 120 but current length is 134  max-line-length
  1592:2  error  Line length must be no more than 120 but current length is 122  max-line-length

✖ 12 problems (12 errors, 0 warnings)
```

Functional Tests

We did functional tests for different contracts manually. Below is the report:

- function fetchItemsCreated → PASS →Returns only items a user has created
- function fetchMyNFTs → PASS → Returns auctions that a user has created
- function fetchMarketItems → PASS → Returns all unsold market items
- function royaltyForItem → PASS → returns royalty for a specific item
- function updateListingPrice → PASS → function to update the listing price
- function EndSale → PASS → function to end the sale
- function buyItem → PASS → function to buyItem
- function createSale → PASS → Places an item for sale on the marketplace

Results
Few issues were found highlighted in the issue section. Some false positive errors were reported by the tool. All the other issues have been categorized above according to their level of severity.

Closing Summary

Overall, smart contracts are very well written and adhere to guidelines.

No instances of Integer Overflow and Underflow vulnerabilities or Back-Door Entry were found in the contract, but relying on other contracts might cause Reentrancy Vulnerability.

Disclaimer

Quillhash audit is not a security warranty, investment advice, or an endorsement of **Unicus**. 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 **Unicus** Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

Audit Report
January, 2022

For

unicus

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com