QuillAudits

Audit Report
November, 2022

For

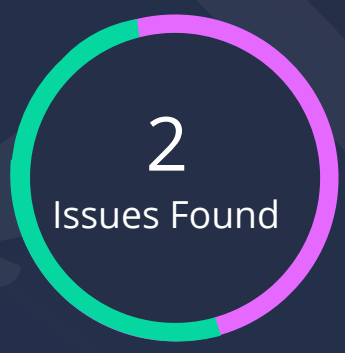
Vegas ONE

Table of Content

Executive Summary

Project Name	VegasOne
Overview	VegasONE NFTMarketPlace contract where NFTs of whitelisted contracts can be listed for sale and NFT auctions. Buying and selling is allowable using ETH and VegasOne as supported currencies.
Timeline	31 October ,2022 to 9th November,2022
Method	Manual Review, Functional Testing, Automated Testing etc.
Scope of Audit	The scope of this audit was to analyze VegasOne NFTMarketPlace codebase for quality, security, and correctness.This included testing of smart contracts to ensure proper logic was followed, manual analysis ,checking for bugs and vulnerabilities, checks for dead code, checks for code style, security and more.
Pull Request	*https://github.com/taisys-technologies/audit-marketplace/pull/4*
Fixed In	*https://github.com/taisys-technologies/audit-marketplace/commit/e6904faf65b2ef6d946e8eca9fbef57270110183*

Commit Hash: e6904faf65b2ef6d946e8eca9fbef57270110183



High
Medium
Low
Informational

	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	0	0	1	1

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.

Contract Information

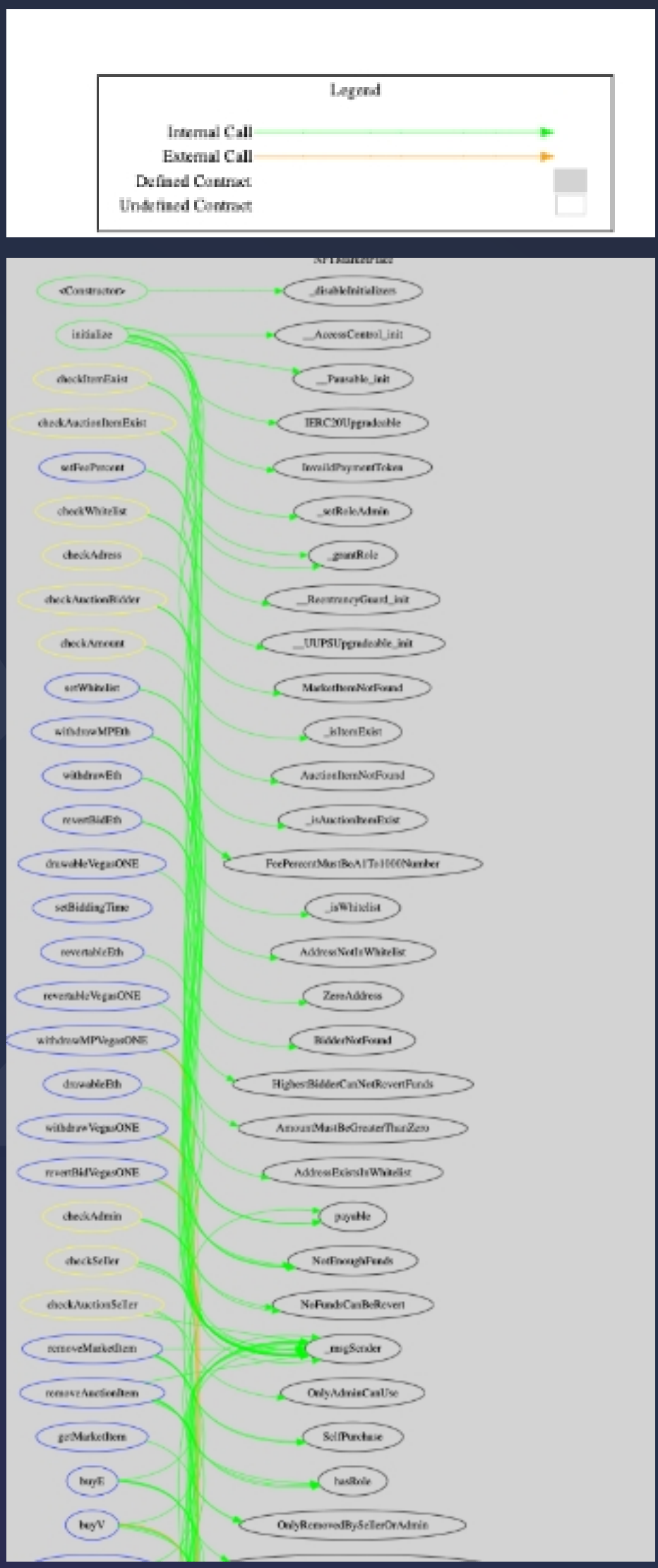
NFTMarketPlace.sol Contract

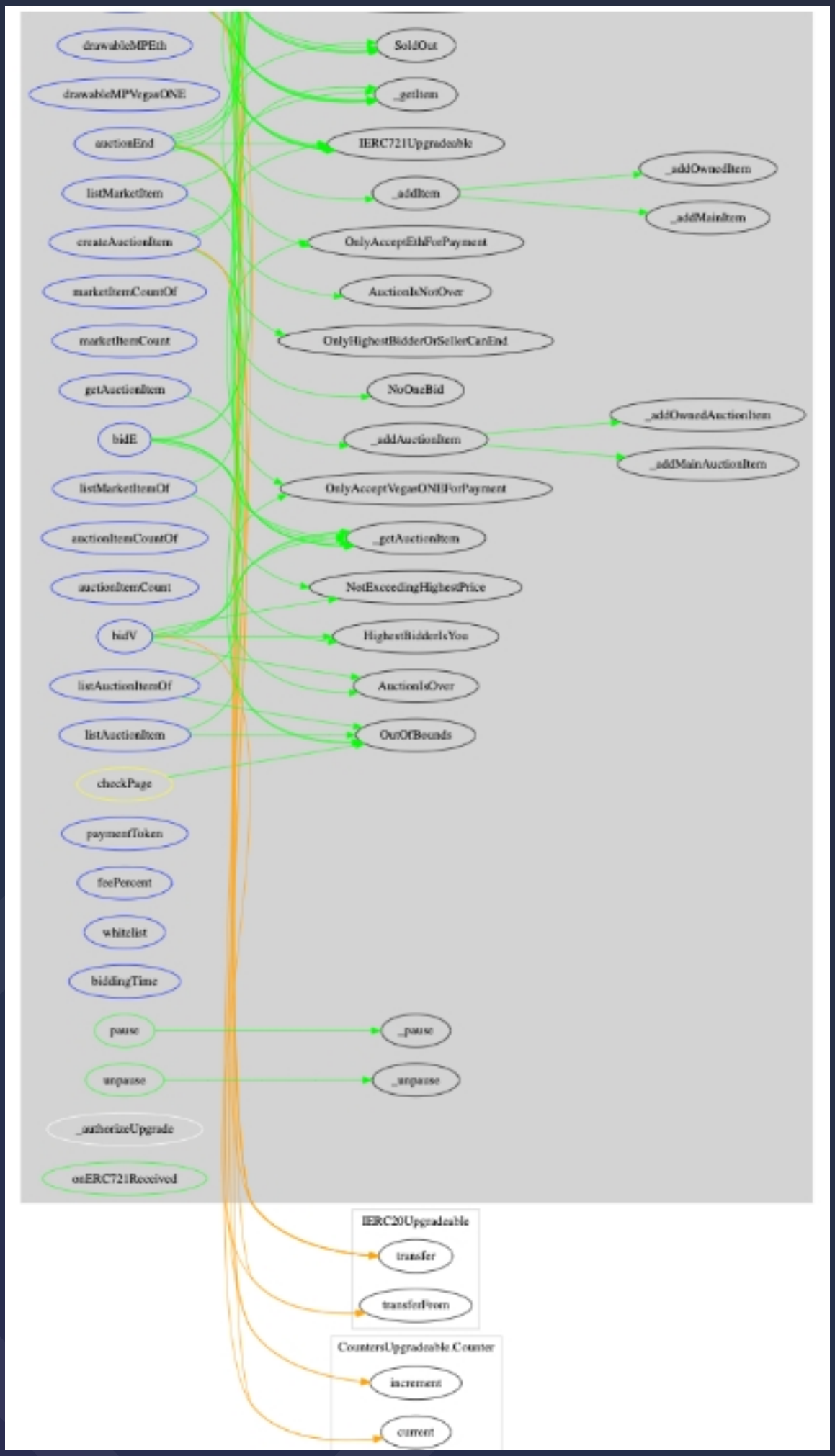
Contract	Lines	Complexity Score	Capabilities
contracts/NFTMarketPlace.sol	1512	493	Payable,initiates ETH value transfer, uses hash functions

NFTMarketPlace.sol Dependencies

	Count
@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol	1
@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol	1
@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol	1
@openzeppelin/contracts-upgradeable/security/PausableUpgradeable.sol	1
@openzeppelin/contracts-upgradeable/security/ReentrancyGuardUpgradeable.sol	1
@openzeppelin/contracts-upgradeable/token/ERC20/IERC20Upgradeable.sol	1
@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol	1
@openzeppelin/contracts-upgradeable/token/ERC721/IERC721ReceiverUpgradeable.sol	1
@openzeppelin/contracts-upgradeable/token/ERC721/IERC721Upgradeable.sol	1
@openzeppelin/contracts-upgradeable/utils/CountersUpgradeable.sol	1

Call Graphs


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
«Constructor»
initialize
checkItemExist
checkAuctionItemExist
setFeePercent
checkWhiteList
checkAdress
checkAuctionBidder
checkAccount
setWhiteList
withdrawMPEth
withdrawEth
revertBidEth
drawableVegasONE
setBiddingTime
revertableEth
revertableVegasONE
withdrawMPVegasONE
drawableEth
withdrawVegasONE
revertBidVegasONE
checkAdmin
checkSeller
checkAuctionSeller
removeMarketItem
removeAuctionItem
getMarketItem
buyE
buyV
_disableInitializers
_AccessControl_init
_Pausable_init
IERC20Upgradeable
InvalidPayment Token
_setRoleAdmin
_grantRole
_ReentrancyGuard_init
_UUPSUpgradeable_init
MarketItemNotFound
_isItemExist
AuctionItemNotFound
_isAuctionItemExist
FeePercentMustBeA1To1000Number
_isWhitelist
AddressNotInWhiteList
ZeroAddress
BidderNotFound
HighestBidderCanNotRevertFunds
AmountMustBeGreaterThanZero
AddressExistsInWhitelist
payable
NotEnoughFunds
NoFundsCanBeRevert
_msgSender
OnlyAdminCanUse
SelfPurchase
hasRole
OnlyRemovedBySellerOrAdmin


drawableMPEth
drawableMPVegasONE
auctionEnd
listMarketItem
createAuctionItem
marketItemCountOf
marketItemCount
getAuctionItem
bidE
listMarketItemOf
auctionItemCountOf
auctionItemCount
bidV
listAuctionItemOf
listAuctionItem
checkPage
paymentToken
feePercent
whitelist
biddingTime
pause
unpause
_authorizeUpgrade
onERC721Received
Sold2Out
_getItem
IERC721Upgradeable
_addItem
OnlyAcceptEthForPayment
AuctionIsNotOver
OnlyHighestBidderOrSellerCanEnd
NoOneBid
_addAuctionItem
OnlyAcceptVegasONEForPayment
_getAuctionItem
NotExceedingHighestPrice
HighestBidderIsYou
AuctionIsOver
OutOfBounds
_addOwnedItem
_addMainItem
_pause
_unpause
_addOwnedAuctionItem
_addMainAuctionItem
IERC20Upgradeable
transfer
transferFrom
CountersUpgradeable.Counter
increment
current

Manual Testing

A. Contract -NFTMarketPlace.sol

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

A1. Use of .transfer() and or .send() to send ETH

Description

Use of transfer() in the following functions
NFTMarketPlace.sol line 418 function withdrawMPEth
NFTMarketPlace.sol line 674 function withdrawEth
NFTMarketPlace.sol line 973 function revertBidEth
NFTMarketPlace.sol line 1022 function auctionEnd

Above mentioned functions were introduced to mitigate reentrancy as they restricted the amount of gas sent. However the best practice is to use .call{} whilst ensuring checks-effects-interactions are done to avoid Reentrancy or make use of Reentrancy guards. Reentrancy protection is already used in function.

Remediation

It is recommended to use .call{value: _amount}("") as in the above example and all other cases where necessary

Status

Resolved

Informational Issues

A2. Implicit use of uint

Description

In the code there are places where uint is used without being specific if it is uint256, uint96 etc although it defaults to uint256 this can cause confusion and or errors.
NFTMarketPlace.sol line 45 uint auctionStartTime
NFTMarketPlace.sol line 102 uint auctionStartTime
NFTMarketPlace.sol line 112 uint auctionStartTime
NFTMarketPlace.sol line 140 uint auctionEndTime
NFTMarketPlace.sol line 166 uint private _biddingTime
NFTMarketPlace.sol line 238 uint newBiddingTime
NFTMarketPlace.sol line 396 setBiddingTime(uint time)
NFTMarketPlace.sol line 734 uint auctionStartTime = block.timestamp
NFTMarketPlace.sol line 1359 external view returns (uint)

Remediation

It is recommended for readability, maintainability and avoidance of errors to be explicit with uint declarations. If it is to be uint256 specify as such or specify as required uint e.g uint32 etc. Remember that smaller units e.g uint8 etc are not always cheaper than uint256

Status

Resolved

Functional Testing

constructor		PASS
checkAdmin	Modifier	PASS
checkItemExist	Modifier	PASS
checkAuctionItemExist	Modifier	PASS
checkWhitelist	Modifier	PASS
checkSeller	Modifier	PASS
checkPage	Modifier	PASS
checkAuctionSeller	Modifier	PASS
checkAddress	Modifier	PASS
checkAmount	Modifier	PASS
initialize	function	PASS
setFeePercent	function	PASS
setWhitelist	function	PASS
setBiddingTime	function	PASS
withdrawMPEth	function	PASS
withdrawMPVegasONE	function	PASS
createMarketItem	function	PASS
removeMarketItem	function	PASS
buyE	function	PASS
buyV	function	PASS
withdrawEth	function	PASS
withdrawVegasONE	function	PASS
createAuctionItem	function	PASS
removeAuctionItem	function	PASS
bidV	function	PASS
bidE	function	PASS

Functional Testing

revertBidVegasONE	function	PASS
revertBidEth	function	PASS
auctionEnd	function	PASS
drawableMPEth	function	PASS
drawableMPVegasONE	function	PASS
getMarketItem	function	PASS
listMarketItem	function	PASS
listMarketItemOf	function	PASS
marketItemCountOf	function	PASS
marketItemCount	function	PASS
getAuctionItem	function	PASS
listAuctionItem	function	PASS
listAuctionItemOf	function	PASS
auctionItemCountOf	function	PASS
auctionItemCount	function	PASS
drawableEth	function	PASS
drawableVegasONE	function	PASS
revertableEth	function	PASS
revertableVegasONE	function	PASS
paymentToken	function	PASS
feePercent	function	PASS
whitelist	function	PASS
biddingTime	function	PASS
_addItem	function	PASS
_addMainItem	function	PASS
_addOwnedItem	function	PASS

Functional Testing

_addMainAuctionItem	function	PASS
_addOwnedAuctionItem	function	PASS
_isItemExist	function	PASS
_getItem	function	PASS
_isAuctionItemExist	function	PASS
_getAuctionItem	function	PASS
_isWhitelist	function	PASS
pause	function	PASS
unpause	function	PASS
onERC721Received	function	PASS

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 VegasOne NFTMarketPlace Contract. We performed our audit according to the procedure described above.

No Major Issues Found During the Audit.

Disclaimer

QuillAudits smart contract audit is not a security warranty, investment advice, or an endorsement of the VegasOne 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 VegasOne 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
November, 2022

For

Vegas ONE

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com