QuillAudits

Audit Report
September, 2022

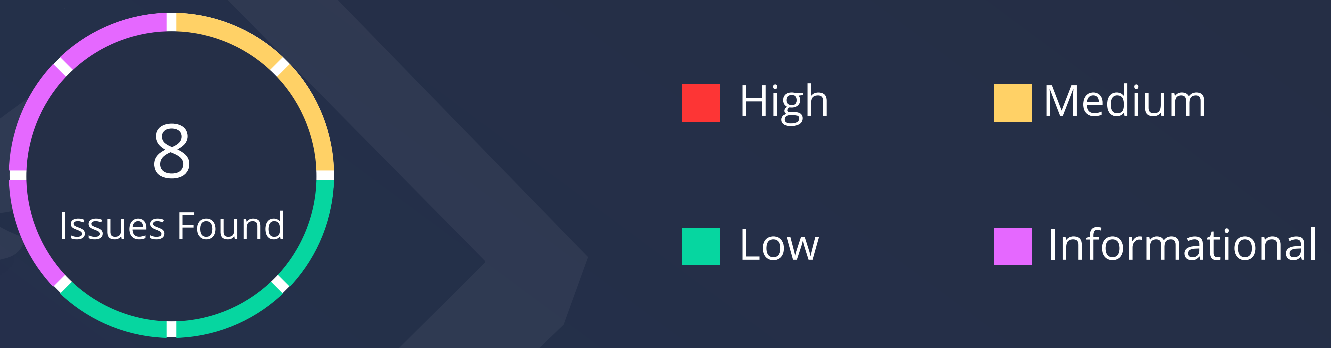
For

Vegas ONE

Table of Content

Executive Summary

Project Name	VegasONE
Overview	The contract is based on ERC721, but in order to save gas fee, they chose to inherit ERC721A. In addition it is an upgradeable contract and integrates the eip-712 signature scheme. Here poll creator creates poll calling createPoll() and voters can vote() for the poll creator. A poll can be in waiting, success, expired status. If a poll is successful then poll creator can mint NFT with the admin recommended signer.
Timeline	10 August, 2022 to 23 September, 2022.
Method	Manual Review, Functional Testing, Automated Testing etc.
Scope of Audit	The scope of this audit was to analyse VegasONE codebase for quality, security, and correctness. *https://github.com/taisys-technologies/audit-exchange/blob/main/contracts/Exchange.sol*
Fixed in	44fd44d5bbd3dd6d7594855f89e47fc4143fe7e9



	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	**1**	0	0
Partially Resolved Issues	0	0	0	0
Resolved Issues	0	**1**	**3**	**3**

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

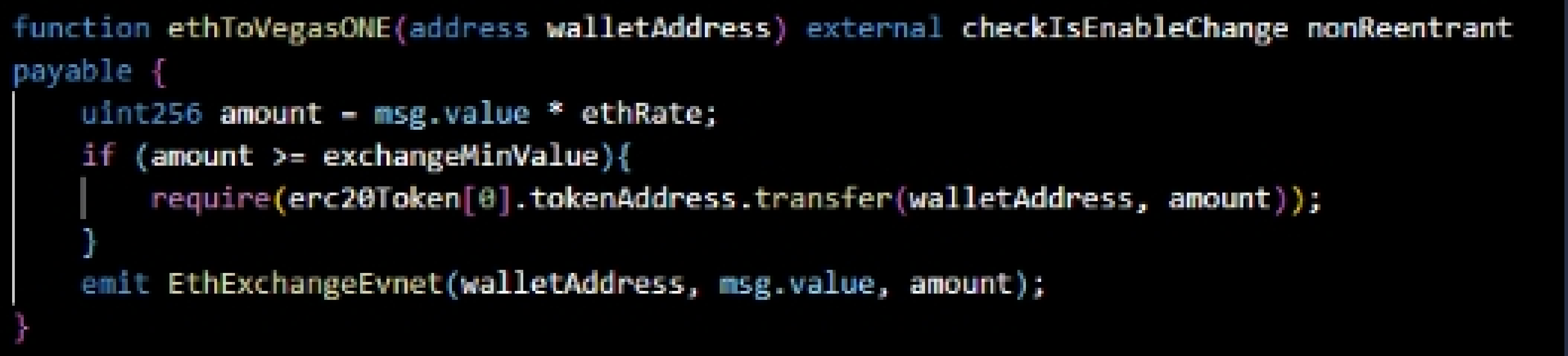
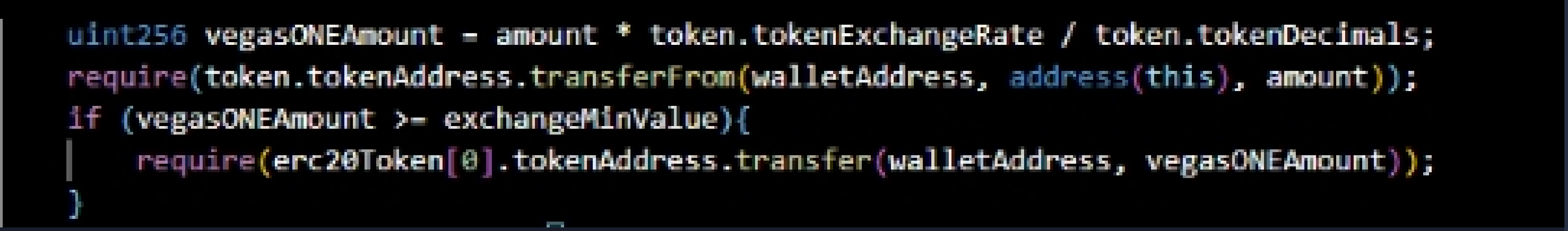
A. Contract - EXCHANGE

High Severity Issues

No issues found

Medium Severity Issues

A1. Negligence for Users With Tokens Less Than Exchange Minimum Value

Line	Function - ethToVegasONE & erc20ToVegasONE
164	
172	

```
function ethToVegasONE(address walletAddress) external checkIsEnableChange nonReentrant
payable {
    uint256 amount = msg.value * ethRate;
    if (amount >= exchangeMinValue){
        require(erc20Token[0].tokenAddress.transfer(walletAddress, amount));
    }
    emit EthExchangeEvnet(walletAddress, msg.value, amount);
}
```

```
    uint256 vegasONEAmount = amount * token.tokenExchangeRate / token.tokenDecimals;
    require(token.tokenAddress.transferFrom(walletAddress, address(this), amount));
    if (vegasONEAmount >= exchangeMinValue){
        require(erc20Token[0].tokenAddress.transfer(walletAddress, vegasONEAmount));
    }
```

Description

These critical functions allow for users to interact with the contract with either ethers or one of the accepted ERC20 tokens. After the mathematical computation for the exchange to occur, there is a precondition that checks that the amount of VegasONE token to be sent to the user is greater than the exchange minimum value. However, the contract fails to handle users interacting with little amount. When values less than an ether are sent to the contract, the contract receives these funds without correspondingly sending VegasONE token to the users. If a person sends 1 wBtc, which is padded with 8 zeros, 100000000.

The contract sets the token exchange rate to 1 * 1e13, which is 10000000000000, padded with 13 zeros.

And the token decimal of wBtc is 8, which is 100000000
According to erc20ToVegasONE, the amount of vegasONEAmount is computed as follows;

amount * token.tokenExchangeRate / token.tokenDecimal

100000000 * 10000000000000 / 100000000

vegasONEAmount = 10000000000000

In that case, vegasONEAmount becomes less than exchangeMinValue (which is 1e18 - 1000000000000000000).

Contract won’t send any vegasToken because it ignores taking care of that instance. The user gets no VegasONE token in return.

Remediation

These functions can be designed to have a revert condition that explains the minimum amount required from users. This will inadvertently prevent receiving ethers or ERC20 tokens from users without giving them VegasONE tokens in return.

Status

Resolved

A2. Centralization Power of the Admin

Description

The contract possesses the feature of centralization of power. This feature allows for Admin to withdraw all tokens and ethers sent into the contract.

Remediation

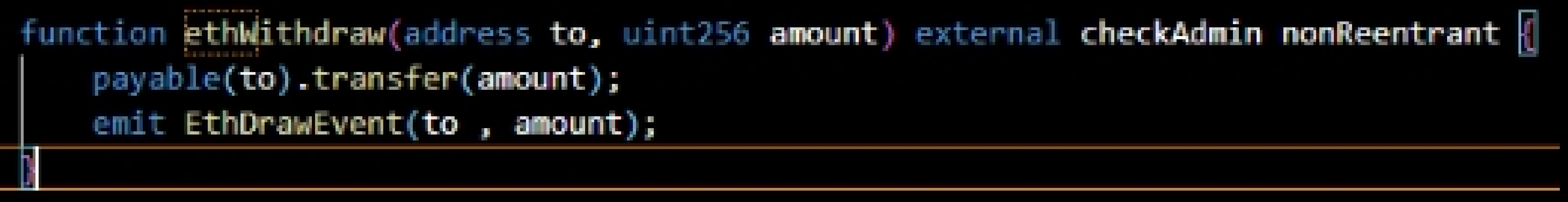
If there is any possibility of funds being moved from exchange to any address or platform. make sure to alert users before moving funds about it on social platforms in communities giving reasons why funds are getting moved.

Status

Acknowledged

Low Severity Issues

A3. Missing Check for Address Zero

Line	Function - ethWithdraw
192	

```
function ethWithdraw(address to, uint256 amount) external checkAdmin nonReentrant {
    payable(to).transfer(amount);
    emit EthDrawEvent(to , amount);
}
```

Description

While the function is restricted to be called by the Admin and possesses a reentrancy prevention check, it fails to check for when address zero is passed as a parameter.

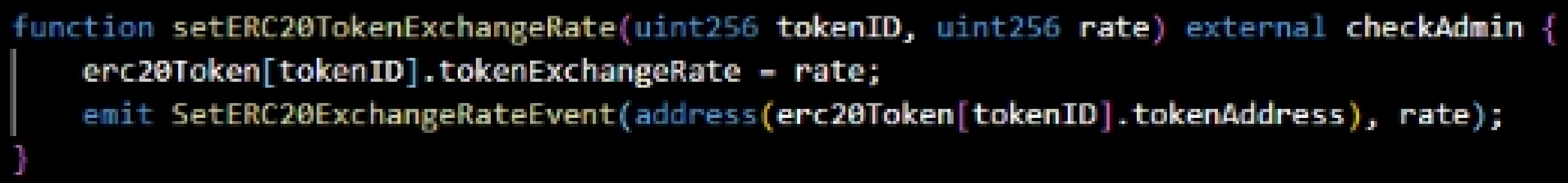
Remediation

Quality check should be added to the function to avoid sending ethers to the null address.

Status

Resolved

A4. Missing Check for Token Existence

Line	Function - setERC20TokenExchangeRate
211	

```
function setERC20TokenExchangeRate(uint256 tokenID, uint256 rate) external checkAdmin {
    erc20Token[tokenID].tokenExchangeRate = rate;
    emit SetERC20ExchangeRateEvent(address(erc20Token[tokenID].tokenAddress), rate);
}
```

Description

This function aids in setting the rate for a token exchange. It fails to check the existence of a token id with the checkTokenIDExist modifier before it sets. The present implementation will allow for setting of tokens that the contract doesn't even recognize.

Remediation

If there is any possibility of funds being moved from exchange to any address or platform. make sure to alert users before moving funds about it on social platforms in communities giving reasons why funds are getting moved.

Status

Resolved

A.5 Use of SafeERC20 library to Avoid Function Call Failure of Some Token

Description

Contract is designed to allow users to interact with several ERC20 tokens with different decimals. Due to the discrepancies of token decimals, there are possibilities of failure when the erc20ToVegasONE function is called, most especially USDT with a token decimal of 6.

Remediation

A good strategy to avoid the failure of the function call and to moderate safe token transfer, as a remedy, the integration of SafeERC20 library will aid in this regard.

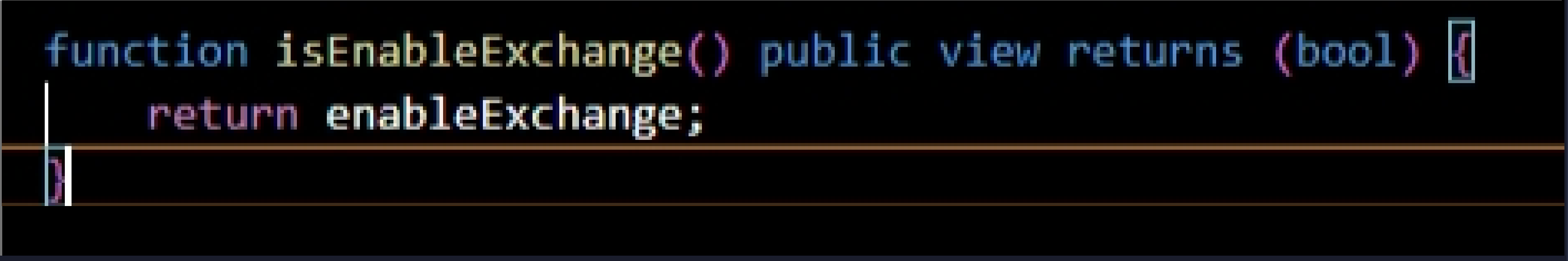
Status

Resolved

Informational Issues

A6. Change Public Function Visibility to External Function

Line	Function - isEnableExchange

234	 `function isEnableExchange() public view returns (bool) {`
	`    return enableExchange;`
	`}`

Description

For gas optimization, public visibility costs more than an external function and from the contract, the isEnableExchange function was not called within the contract.

Remediation

It is recommended to declare this function as external to save gas.

Status

Resolved

A7. Absence of Proper Code Comment

Description

Proper code comment is a necessary feature of a standard smart contract because it details the motive behind the business logic and helps for quick understanding of smart contracts by prospective users and developers.

Remediation

It is recommended that the whole contract should be properly commented. The Natspec code format is mostly recommended.

Status

Resolved

A8. General Recommendations

Description

The VegasONE exchange contract inherits the reentrancy guard and access enumerable from Openzeppelin. While these contracts aid in handling the administrative role and guarding reentrancy, it is recommended to give critical look to the highlighted issues above; proper code comments, use of external visibility to save gas, check for address zero for critical functions and the function that allows for the swaps of ethers or ERC20 token to VegasONE token.

Status

Resolved

Functional Testing

- ✓ Should get the state of the isEnableExchange
- ✓ Should revert if not the admin adding an ERC20Token to contract
- ✓ Should emit exact events expected when token is added
- ✓ Should revert when an unknown account sets ETHExchangeRate
- ✓ Should emit expected values when ethExhangeRate is set by admin
- ✓ Should revert when an unknown account sets ERC20TokenExchangeRate
- ✓ Should emit expected values when ERC20Token Exchange Rate is set by admin
- ✓ Should revert when ERC20TokenExchangeRate is set for non-exist token id.
- ✓ Should revert when an unknown account sets setContractExchangeStatus
- ✓ Should emit expected values when Exchange Status is set by admin
- ✓ Should revert when an unknown account sets setTokenExchangeStatus
- ✓ Should emit expected values when Token Exchange Status is set by admin
- ✓ Should revert when Token Exchange Status is called but token does not exist.
- ✓ Should set ExchangeMin Value when admin calls setExchangeMinValue function.
- ✓ Should reverts when accounts exchange ethers for VegasONE token but checkIsEnableChange is not active
- ✓ Should revert when no ether is sent alongside the ethToVegasONE
- ✓ Should allow accounts exchange ethers for vegasONE Token
- ✓ Should allow accounts to exchange BUSD for vegasONE Token
- ✓ Should allow accounts to exchange USDC for vegasONE Token
- ✓ Should allow accounts to exchange BNB for vegasONE Token
- ✓ Should allow accounts to exchange wBTC for vegasONE Token
- ✓ Should withdraw all ethers to an address after an exchange of ethers to vegasONE token
- ✓ Should withdraw BUSD tokens after the exchange of BUSD to vegasONE token
- ✓ Should withdraw USDC tokens after the exchange of USDC to vegasONE token

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. 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. In the End, VegasOne Team Resolved almost all Issues except one Issue.

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.

600+
Audits Completed

$15B
Secured

600K
Lines of Code Audited

Follow Our Journey

Audit Report
September, 2022

For

Vegas ONE

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com