QuillAudits

Audit Report
August, 2023

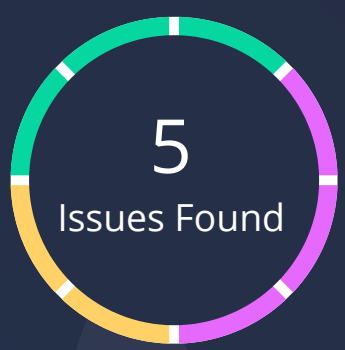
For

VTR

Table of Content

Executive Summary

Project Name	VTRConnect
Audit Scope	The scope of this audit was to analyze and document the VTRConnect Token smart contract codebase for quality, security, and correctness. *https://github.com/virtualspirit/blockchain-solidity-contracts-vtrc/tree/9c3a01c4e948242ee91b60ffb808800c99bc83c9/contracts*
Version 1	July 26, 2023 9c3a01c4e948242ee91b60ffb808800c99bc83c9
Version 2	July 28,2023 *https://github.com/virtualspirit/blockchain-solidity-contracts-vtrc/tree/3b7c52f8c63ae3f84756788e3aaafba51d396c50*



High
Medium
Low
Informational

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

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. VTRConnect contracts

High Severity Issues

No issues found

Medium Severity Issues

A.1 Missing validation of updatedAt

Line #321 *(, int256 price, , ,) = priceFeed.latestRoundData();*

Description

Insufficient validation of the priceFeed is apparent, as it neglects to include verification of the updatedAt value. It is imperative to validate the updatedAt value as well. Failure to do so could expose the system to outdated data, potentially facilitating oracle manipulation attacks. Thus, it is crucial to implement comprehensive validation mechanisms to mitigate such risks effectively.

Remediation

Before utilizing the price value in cost price calculations, it is of utmost importance to rigorously validate the updatedAt value. By confirming the freshness of the updatedAt value, the system can ensure that the price data being used is up-to-date and not subject to manipulation. Therefore, prior to any cost price calculations, the system must perform thorough and accurate validation of the updatedAt timestamp to maintain the integrity and security of the process.

Status

Resolved

Client's Comment: *We are using Chainlink price oracle's latestRoundData function to get the latest price. Since we have a dependency on chainlink here, there can be an issue if chainlink stops updating the price from their end. To handle this scenario, we have made the pricefeeds updateable by admin, So if chainlink fails sometimes in future, or changes their contract address admin can change the price oracle's address. Also admin can pause this functionality from the smart contract. QuillAudits' suggestion can also be implemented, we can fail the transaction if the price has not been updated in the last two hours (or any decided time).*

Low Severity Issues

A.2 NFTPriceUSDUpdated event should include changes in `nftPriceUSDDecimals`

Line #156 - 163

```
function setNftPriceUSD(
    uint256 newNftPriceUSD,
    uint256 newNFTPriceUSDDecimals
) external onlyRole(DEFAULT_ADMIN_ROLE) {
    nftPriceUSD = newNftPriceUSD;
    nftPriceUSDDecimals = newNFTPriceUSDDecimals;
        emit NFTPriceUSDUpdated(newNftPriceUSD);
}
```

Description

To ensure better monitoring and transparency, any changes made to the nftPriceUSDDecimals should be reflected in the NFTPriceUSDUpdated event. By doing so, stakeholders and monitoring systems can stay informed about updates to the decimal precision of the NFT price in USD.
To implement this improvement, you need to modify the contract's code where the nftPriceUSDDecimals value is updated. Ensure that after the update, the contract emits the NFTPriceUSDUpdated event with the updated nftPriceUSDDecimals value, allowing external systems to track the changes more accurately.

Status

Resolved

One extra variable to be added in emitting event

A.3 Use ERC721 as base contract instead of ERC1155

Line #18 *contract VTRConnect is ERC1155, AccessControl, Pausable, ReentrancyGuard, NativeMetaTransaction*

Description

Since the current contract only mints non-fungible tokens with a fixed value of 1, using ERC721 instead of ERC1155 seems to be a more appropriate choice for NFTs. By replacing the base contract of ERC1155 with ERC721 and implementing batch transfer functionality, you can achieve concurrency with the actual product and optimize gas usage during deployment and runtime. ERC721 is a straightforward standard for representing individual NFTs, and with batch transfer functionality, you can efficiently handle multiple transfers while maintaining simplicity and catering to the project's requirements. This approach eliminates the overhead associated with managing fungible balances and IDs, which are not necessary for your use case. Furthermore, ERC721 offers a familiar and widely recognized user experience for NFT collectors and enthusiasts. As ERC721 has better interoperability with existing NFT marketplaces, wallets, and platforms, it ensures seamless integration with the broader NFT ecosystem.

Status

Resolved

VTR Team Comment: *Will stay with ERC1155.*
Using the ERC1155 contract was a part of PSD, and we built upon it further. Since we needed batch minting and transfer functionality, we stayed with ERC1155 instead of moving to ERC721.

Informational Issues

A.4 Validation of adding tokens in whitelist during deployment

Line #92-99

```
for (uint256 i = 0; i < _acceptedTokens.length; i++) {
        acceptedTokens.push(_acceptedTokens[i]);
        acceptedTokenIndex[_acceptedTokens[i]] = i;
        isAcceptedToken[_acceptedTokens[i]] = true;
        priceFeeds[_acceptedTokens[i]] = AggregatorV3Interface(
            _priceFeeds[i]
        );
    }
```

Description

Please exercise caution when adding tokens during the constructor phase, as the parameters are not currently validated. This oversight can potentially lead to the insertion of a single token twice, causing unintended consequences. To address this, I recommend implementing a check in the constructor using require(!isAcceptedToken[token], "VTRConnect: Token already added"); before adding a new token. This validation will ensure that duplicate tokens cannot be inserted.
Alternatively, you can opt to validate the tokens after the contract is deployed. By conducting token validation after deployment, you can mitigate the risk of duplicates being inadvertently added during the constructor phase.
Both of these approaches will enhance the integrity of your contract and prevent any unintended duplications of tokens, ensuring a smooth and reliable functionality.

Status

Resolved

VTR Team Comment: *These tokens are being added in the constructor, and data will be passed from the env file, so we can assume the data will be correctly passed. Adding checks will only increase deployment gas costs.*
Also, even if in a rare scenario, two tokens are added by mistake, the admin can remove them anytime. The only thing affected will be the count of tokens which is just for information and not used anywhere.

A.5 Gas Golfing

Line #239-242

```
require(
        totalSupply + tokenIds.length <= maxSupply,
        "VTRConnect: Exceeds the maximum supply limit"
    );
```

Description

The previously highlighted code snippet can be safely removed since it duplicates the validation already performed within the for loop of the _mint function. This additional validation would only increase the runtime gas consumption, making it redundant and unnecessary.

Status

Resolved

One check is added twice. Can be removed.

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

Summary

Overall, in the initial audit, there is one medium severity issue associated with missing validation of an oracle date.

No instances of Integer Overflow and Underflow vulnerabilities are found in the contract.

Numerous issues were discovered during the initial audit. It is recommended to kindly go through the above-mentioned details and fix the code accordingly.

Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in VTRConnect 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 VTRConnect 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 VTRConnect 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
August, 2023

For

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com