QuillAudits

Audit Report
March, 2024

For

buk

DON'T CANCEL. RESELL

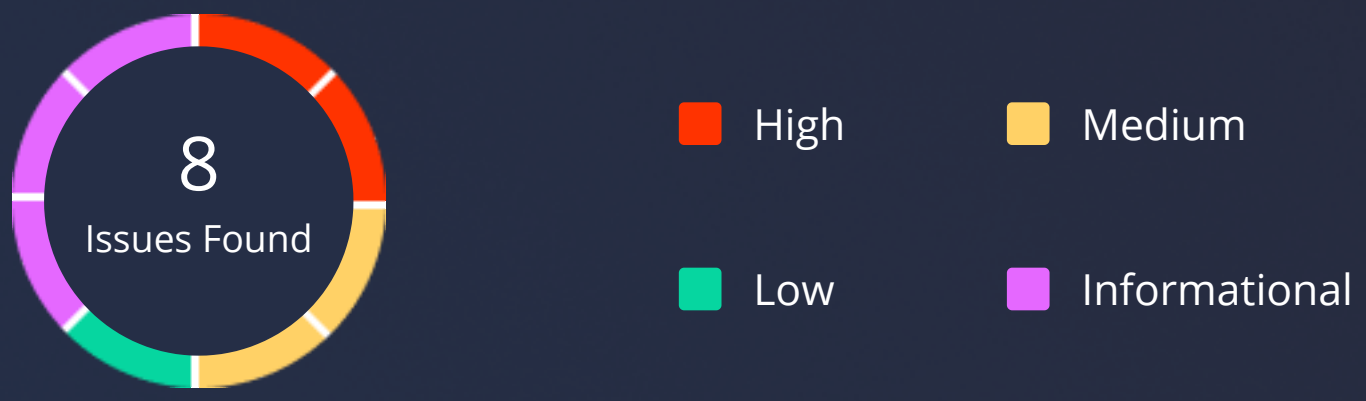
Table of Content

Table of Content

Executive Summary

Project Name	Buk Protocol
Overview	Tokenization platform for hotel room bookings creating a transparent secondary market for the hospitality industry.
Timeline	12th February 2024 - 29th February 2024
Updated Code Received	6th March 2024
Second Review	8th March 2024 - 11th March 2024
Method	Manual Review, Functional Testing, Automated Testing, etc. All the raised flags were manually reviewed and re-tested to identify any false positives.
Audit Scope	The scope of this audit was to analyze the buk protocol Codebase for quality, security, and correctness.
Source Code	*https://github.com/BUK-protocol/buk-smart-contract-v2/tree/development*
Branch	Development
Fixed In	38e25a8bca23a4f0e43e685b2a3aca5936f2847c

Number of Security Issues per Severity



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

Checked Vulnerabilities

- 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 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 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-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 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

Hardhat, Foundry.

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.

High Severity Issues

1. Unrestricted Booking and Potential NFT Misuse in bookRooms() function (BukProtocol.sol [#L163])

Path

BukProtocol.sol

Function

bookRooms()

Description

The **bookRooms()** function in **BukProtocol.sol** (line **[#L163]**) suffers from a critical security vulnerability that allows for unrestricted booking and potential misuse of minted NFTs.

The function allows any user to book rooms by providing details, including the booking price. However, the current implementation accepts bookings with a price of zero. This creates two concerning scenarios:

Free Room Bookings: Malicious actors can exploit this weakness to book any number of rooms without paying the intended price. This could lead to reservation flooding and disrupt the intended functionality of the booking system.

Potential NFT Misuse: After booking the rooms a malicious user can mint the NFT using **mintBukNFT()** function. However, due to the unvalidated booking price, a malicious actor could:

- Mint the bukNFT without paying the booking price.
- List the NFT on secondary marketplaces for sale at a higher price or even nominal prices.
- Potentially claim the actual booking using the minted NFT (although the impact of this is unclear due to the out-of-scope nature of the booking claim functionality).

These vulnerabilities can have significant consequences for the Buk protocol:

Financial Loss: Unrestricted free bookings can lead to lost revenue for the platform.

Market Manipulation: Malicious actors could manipulate the secondary market by selling NFTs associated with unvalidated bookings.

Reputational Damage: If users are unable to claim their booked rooms due to malicious activity, it can damage the platform's reputation and user trust.

Recommendation

To address these vulnerabilities, it is crucial to implement the following:

- Either restrict the user to mint the NFT and make **mintBukNFT() onlyAdmin**.
- Or make sure that **bookRooms()** function will not take prices as params, It should be read through some trusted oracle.

Status

Resolved

2. Inconsistent NFT Ownership Handling in checkout() function (BukProtocol.sol [#L328])

Path

BukProtocol.sol

Function

checkout()

Description

The **checkout()** function within **BukProtocol.sol** (line **[#L328]**) exhibits a security vulnerability related to inconsistent NFT ownership handling during the checkout process.

The function attempts to Burn the **bukNFT** associated with the booking & Mint a new **bukPOSNFT** and assign it to the **firstOwner**.

However, this implementation suffers from a critical flaw:

The function does not verify the current owner of the bukNFT before attempting to burn it. Since bukNFTs are tradeable, it's possible for the firstOwner to no longer hold the NFT at the time of checkout. Due to which burn call will revert if the **firstOwner** is not the current owner, disrupting the checkout process.

Recommendation

To address this vulnerability, it is crucial to implement the following:

Before attempting to burn the bukNFT, the checkout() function should verify that the firstOwner is still the current owner using the appropriate ownership check function.

Handle Non-Matching Ownership: If the ownership check fails, Then find the owner of the given tokenId using **ownerOf** function and use that owner to burn and mint new bukPOSNFT.

Status

Resolved

Medium Severity Issues

3. Inconsistent Commission Handling in bookingRefund() function (BukPOSProtocol.sol [#L247])

Path

BukPOSProtocol.sol

Function

bookingRefund()

Description

The **bookingRefund()** function in **BukPOSProtocol.sol** (line **[#L247]**) calculates the total refund amount for a booking based on its cost and the commission charged by the Buk protocol. However, this implementation introduces a security concern due to:

Dynamic Commission: The commission rate can be modified by the admin. This raises concerns.

Incorrect Refunds: If a booking was made with an older commission rate and the admin has since increased the rate, the refund calculation would be inaccurate. This could result in the owner receiving less than the entitled amount or the protocol overpaying the owner, potentially leading to **BukTreasury** insolvency.

This vulnerability can have significant financial implications for the Buk protocol:

Unintended Financial Loss: Inconsistent refund calculations due to commission changes could lead to the protocol losing revenue or experiencing unexpected financial strain.

User Discontent: Users receiving incorrect refunds could be dissatisfied and damage the platform's reputation.

Recommendation

Fixed Commission for Past Bookings: The **bookingRefund()** function should utilize the commission rate applicable at the time of booking when calculating refunds for past bookings. This ensures consistent and accurate calculations regardless of future commission changes.

Status

Resolved

4. Inconsistent Transaction Logic in emergencyCancellation() Function (BukProtocol.sol)

Path

BukProtocol.sol

Function

bookRooms()

Description

The **emergencyCancellation()** function within **BukProtocol.sol** exhibits an inconsistency in its transaction logic related to booking cancellation and associated charges.

During an emergency cancellation initiated through this function:

- A refund is initiated for the booking owner.
- The cancellation charges are transferred to the **_bukTreasury** contract.

However, this implementation contradicts the intended behavior established in the **cancelRooms()** function, The **cancelRooms()** function mandates that cancellation charges should be transferred to the **_bukWallet** address.

This inconsistency can lead to:

Misdirected Funds: Cancellation charges meant for the **_bukWallet** are incorrectly directed towards the **_bukTreasury**. This can disrupt the intended financial flow and potentially impact budgeting and revenue tracking.

Operational Inconsistencies: The discrepancy in logic between the two functions can create confusion and inconsistencies within the protocol's internal operations.

Recommendation

To rectify this vulnerability, it is crucial to align the transaction logic in the **emergencyCancellation()** function with that of the **cancelRooms()** function. Specifically, the **emergencyCancellation()** function should be modified to transfer the cancellation charges to the designated **_bukWallet** address instead of the **_bukTreasury**.

Status

Resolved

Low Severity Issues

5. Unrestricted Past Booking Refunds in bookingRefund() Function (BukPOSNFTs.sol [#L228])

Path

bukProtocol

Function

bookingRefund()

Description

The **bookingRefund()** function in **BukPOSNFTs.sol** (line **[#L228]**) grants the admin the ability to refund bookings based on provided bookingIds. However, this implementation poses a security risk due to:

Unrestricted Refund Window: The function permits refunds even for bookings with a check-in time in the past.

Accidental Refunds: Admins could unintentionally issue refunds for expired bookings due to human error, leading to financial loss for the platform.

Malicious Activity: Malicious actors with compromised admin access could exploit this vulnerability to issue unauthorized refunds for past bookings, causing financial harm.

Recommendation

We recommend to add a sanity check into the **bookingRefund()** function that only bookings whose check-in time in future can be refunded.

Status

Acknowledged

Informational Issues

6. Inconsistent Use of Role Management Functions in BukPOSNFTs.sol

Path

BukPONFTs.sol

Function

Role management

Description

The **BukPOSNFTs.sol** smart contract exhibits an inconsistency in its approach to role management. While the contract utilizes both **_setupRole()** (line **[#L61]**) and **_grantRole()** (line **[#L62]**) functions to assign permissions, it's crucial to note that:

_setupRole() has been deprecated in recent OpenZeppelin libraries. This means its continued use can lead to compatibility issues and potential security vulnerabilities in the future as the library evolves.

Recommendation

To ensure clarity, maintainability, and future compatibility, it is strongly recommended to adopt a consistent approach to role management. We recommend the following:

- Completely remove all instances of **_setupRole()** from the smart contract.
- Replace any usage of **_setupRole()** with **_grantRole()**, ensuring all permission assignments are handled through the recommended and supported function.

By adhering to these recommendations, the smart contract will achieve consistent role management practices, eliminate potential compatibility issues, and reduce overall gas consumption due to the removal of the deprecated function. This will enhance the overall security and maintainability of the contract.

Status

Resolved

7. Throughout Codebase Inadequate Use of Safe Transfer Functions

Path

BukPONFTs.sol

Function

Role management

Description

A pervasive security concern is identified throughout the codebase: the inconsistent use of the **safeTransfer** and **safeTransferFrom** functions from the **OpenZeppelin SafeERC20** library.

The majority of instances involving stablecoin transfers do not leverage the recommended **safeTransfer** and **safeTransferFrom** functions. These functions offer crucial advantages:

Reentrancy Protection: They prevent reentrancy attacks, a common exploit where a malicious function can call itself recursively to steal funds during a transfer.

Error Handling: They automatically revert the transaction in case of failure, providing valuable feedback and preventing unexpected behavior.

The absence of these safeguards exposes the smart contract to various attack vectors like Reentrancy Attacks, Unexpected Transaction Reversion.

Recommendation

To mitigate these risks, it is imperative to systematically review and refactor the codebase to ensure that all stablecoin transfers utilize the **safeTransfer** and **safeTransferFrom** functions provided by the OpenZeppelin SafeERC20 library. This comprehensive update will significantly enhance the security and reliability of the smart contract by preventing reentrancy attacks, ensuring proper error handling, and safeguarding transferred funds.

Status

Resolved

8. Inadequate Message Construction in generateAndVerify Function (BukProtocol.sol)

Path

BukProtocol.sol

Function

generateAndVerify

Description

The **generateAndVerify** function within **BukProtocol.sol** exhibits a security concern related to the construction of the message used for signature verification during room cancellation refunds.

The function fails to incorporate the **nonce** and **address(this)** parameters into the message that is signed. This omission creates a vulnerability known as a replay attack:

- An attacker can intercept a valid signature used for a previous refund request.
- The attacker can then reuse the intercepted signature with the same message in a subsequent request, tricking the smart contract into processing a duplicate refund.

While the current implementation restricts the call of **cancelRooms()** (which utilizes **generateAndVerify**) to the admin address, recommending the adoption of industry best practices is still crucial.

This vulnerability can lead to:

Unauthorized Refunds: Malicious actors could exploit the vulnerability to initiate unauthorized refunds, potentially leading to financial losses for the protocol.

Recommendation

To mitigate this risk and adhere to industry best practices, it is highly recommended to modify the **generateAndVerify** function to incorporate the following elements into the message construction:

Nonce: A unique identifier that increases with each signature request. This prevents the reuse of previous signatures.

Address of the Calling Contract: The address of the contract calling generateAndVerify. This ensures that only authorized contracts can generate valid signatures.

Status

Acknowledged

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 Buk protocol codebase. We performed our audit according to the procedure described above.

Some issues of High, 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 Buk protocol 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 Buk protocol 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 Buk protocol 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.

1000+
Audits Completed

$30B
Secured

1M
Lines of Code Audited

Follow Our Journey

Audit Report
March, 2024

For

buk
DON'T CANCEL. RESELL

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com