SECURITY AUDIT OF

WORLDS BEYOND MODULE

Public Report

Jun 09, 2023

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

ABBREVIATIONS

Name	Description
Sui Blockchain	Sui is an innovative, decentralized Layer 1 blockchain that redefines asset ownership. Sui Move feels like a paradigm change in web3 development. Treating objects as 1st class citizens brings composability to a whole new level. Polymedia. We are thrilled to be building on Sui.
Move	Move is a new programming language that implements all the transactions on the Aptos/Sui blockchain.
Move Module	A Move module defines the rules for updating the global state of the Aptos/Sui blockchain. In the Aptos/Sui protocol, a Move module is a smart contract.
NFT	A non-fungible token (NFT) is a unique digital identifier that cannot be copied, substituted, or subdivided, that is recorded in a blockchain, and that is used to certify authenticity and ownership.

EXECUTIVE SUMMARY

This Security Audit Report was prepared by Verichains Lab on Jun 09, 2023. We would like to thank the Worlds Beyond for trusting Verichains Lab in auditing smart contracts. Delivering high-quality audits is always our top priority.

This audit focused on identifying security flaws in code and the design of the Worlds Beyond Module. The scope of the audit is limited to the source code files provided to Verichains. Verichains Lab completed the assessment using manual, static, and dynamic analysis techniques.

During the audit process, the audit team had identified a vulnerable issue in the application.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About Worlds Beyond Module

The Worlds Beyond Module is scheduled for deployment on the Sui blockchain, a decentralized ledger technology. Its primary purpose is to provide a comprehensive solution for the efficient management of loyalty points and various resources within a gaming environment. These resources encompass a wide range of virtual items, including cards, boxes, and accessories. By leveraging the capabilities of the Sui blockchain, the product aims to enhance the transparency, security, and accessibility of managing these in-game assets.

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of the smart contracts of Worlds Beyond Module. It was conducted on the source code provided by the Worlds Beyond team.

The latest version of the following files were made available in the course of the review:

SHA256 Sum	File
28bdf586f3dbd530bb0959c3c0f1215bc9372c7a9ac4a6e3cfe20fa3d82df150	card.move
cbf6b56dbfb27c62d026f93233a64c3ad417a7e291df41fbc1d5865d2f26eb72	resource.move
d7e3d515d70520e8218fd73d206c29ef6c4fd69cb6d94300884c11eaa18d1640	utility.move
6e0735e1c676f6dfa8a28a5dc1723f7816632357fdaf412c4c28ef42019fefba	box.move
a335a6f99680db3b437724c7cb0d595106797834337a5796a0bbecd19cbecc3a	loyalty.move

1.3. Audit methodology

Our security audit process for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using public and RK87, our in-house smart contract security analysis tool.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that were considered during the audit of the smart contract:

- Numerical precision errors
- Transaction-Ordering Dependence

- DoS with (Unexpected) revert
- Gas Usage, Gas Limit and Loops
- Logic Flaws

For vulnerabilities, we categorize the findings into categories as listed in table below, depending on their severity level:

SEVERITY LEVEL	DESCRIPTION
CRITICAL	A vulnerability that can disrupt the contract functioning; creates a critical risk to the contract; required to be fixed immediately.
HIGH	A vulnerability that could affect the desired outcome of executing the contract with high impact; needs to be fixed with high priority.
MEDIUM	A vulnerability that could affect the desired outcome of executing the contract with medium impact in a specific scenario; needs to be fixed.
LOW	An issue that does not have a significant impact, can be considered as less important.

Table 1. Severity levels

1.4. Disclaimer

Please note that security auditing cannot uncover all existing vulnerabilities, and even an audit in which no vulnerabilities are found is not a guarantee for a 100% secure smart contract. However, auditing allows discovering vulnerabilities that were unobserved, overlooked during development and areas where additional security measures are necessary.

2. AUDIT RESULT

2.1. Overview

The Worlds Beyond Module was written in `Move` language which dependencies `Sui Framework` and `NftProtocol`.

2.1.1. Roles & Actors

- Owner: The contract deployer acts as the default owner, granting them the authority to mint NFTs for any individual without requiring a signature from a designated signer.
- Admin: The configuration contract establishes the admin, who possesses the capability to modify the name and description of NFTs, as well as designate a signer within the global configuration.
- Signer: The signer, in this context, does not hold a role but rather represents an address used to verify messages sent by players that have been signed by the designated signer.
- Player: In this system, players have the ability to mint or burn NFTs. Specifically, players can initiate transactions to mint boxes, loyalty points, and resources, but it is crucial that these requests are accompanied by a valid signature from the designated signer in the backend of the game. This ensures the authenticity and validity of the player's actions within the game.

2.1.2. Features & Data Structures

2.1.2.1.Loyalty

```
struct Box has key {
    id: UID,
    type: u64,
    name: String,
    loyalty_points: u64,
}

struct Ticket has key, store {
    id: UID,
    type: u64,
    name: String,
    loyalty_points: u64,
}
```

- `mint_box_for`:
  - Admin can mint a `Box` and send it to a specific recipient, using OriginByte `MintCap`.
- `claim_box`:
  - Admin allow user to claim loyalty box by signing a signature, contains the data to claim.

   - User get that signature, along with the data, to mint a loyalty box.
- `sync_loyalty_points`
   - Admin allow user to sync loyalty points by signing a signature, contains the points to be synced.
   - User get that signature, along with the points, to sync the loyalty points in the loyalty box.
- `issue_loyalty_ticket`
   - If user want to split loyalty points to list for sale, they can issue a loyalty ticket. The ticketcontains the loyalty points, subtract by the points in the loyalty box.
- `apply_loyalty_ticket`
   - When having a loyalty ticket, a user can apply that ticket to his loyalty box, to increase theloyalty points in that box.

2.1.2.2. Resource

```
struct Resource has key, store {
    id: UID,
    type: u64,
    name: String,
}
```

- `mint_for`:
   - Admin can mint a `Resource` and send it to a specific recipient, using OriginByte `MintCap`.
- `claim`:
   - Admin allow user to claim resource by signing a signature, contains the data to claim.
   - User get that signature, along with the data, to mint a resource.
- `craft`:
   - User can mint a box (different with loyalty `Box`, see below) by crafting two resources together. This will burn two providing resources.

2.1.2.3. Card

```
struct Card has key, store {
    id: UID,
    type: u64,
    name: String,
    expired_at: u64,
}
```

- `mint_for`:
   - Admin can mint a `Card` and send it to a specific recipient, using OriginByte `MintCap`.
- `mint`:
   - Admin allow user to mint card by signing a signature, contains the data to mint.
   - User get that signature, along with the data, to mint a card.

- `upgrade`:
    - Admin allow user to upgrade an existing card by signing a signature, contains the data to upgrade.
    - User get that signature, along with the data, to upgrade that card.
- `upgrade_and_burn`:
    - Admin allow user to upgrade an existing card, and also burn some other cards, by signing a signature, contains the data to upgrade.
    - User get that signature, along with the data, to upgrade that card.

2.1.2.4. Box

```
struct Box has key, store {
    id: UID,
}
```

- `mint_for`:
    - Admin can mint a `Box` and send it to a specific recipient, using OriginByte `MintCap`.
- `open`:
    - User can open a box. This will burn that box and emit an `BoxOpened` event.
- `claim`:
    - Admin allow user to claim box by signing a signature, contains the data to claim.
    - User get that signature, along with the data, to claim that box.

2.2. Findings

During the audit process, the audit team found a vulnerability in the given version of Worlds Beyond Module.

#	Issue	Severity	Status
1	There are no NFT ID in signed message	LOW	Acknowledged

2.2.1. LOW - There are no NFT ID in signed message

Position:

- `loyalty.move`: `sync_loyalty_points()`, `issue_loyalty_ticket()`, `apply_loyalty_ticket()`
- `resource.move`: `craft()`, `claim()`
- `box.move`: `open()`
- `card.move`: `mint()`, `upgrade()`, `upgrade_and_burn()`

2.2.1.1. Description

Although the functions mentioned above utilize signature verification, it is important to note that they do not hash all parameters, such as the Box ID, Card ID, Ticket ID, and Resource

ID. This omission could potentially pose security risks in the future, as it opens up the possibility for players to exploit this vulnerability and gain unauthorized control over NFT items within the game. To ensure a higher level of security and mitigate such risks, it is advisable to include hashing for all relevant parameters.

RECOMMENDATION

We highly recommend the signature have to include hashing for all relevant parameters.

UPDATES

Jun 09, 2023: The issues was acknowledged by the Worlds Beyond Module team

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Jun 01, 2023*	Public Report	Verichains Lab
1.1	*Jun 09, 2023*	Public Report	Verichains Lab

Table 2. Report versions history