BlockUs

Audit Report

MOVEBIT

contact@bitslab.xyz

https://twitter.com/movebit_

Fri Nov 15 2024

BlockUs Audit Report

1 Executive Summary

1.1 Project Information

Description	Sui Marketplace for Game studio
Type	NFT Marketplace
Auditors	MoveBit
Timeline	Thu Nov 07 2024 - Fri Nov 15 2024
Languages	Move
Platform	Sui
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/BlockUs0/marketplace-sui
Commits	e8a96a03f80e7b4d39df314bf76093745d51bb21 97311e574c5e1502f1fa5ff53412d84a09bada23

1.2 Files in Scope

The following are the SHA1 hashes of the original reviewed files.

ID	File	SHA-1 Hash
MFT	sources/marketplace_fixed_trade.move	44c0ed137bc21621e04d2bf1d1e31906eb49a49b
EVE	sources/events.move	2fe0bd2e7a5838063345b6b9d7c70373545245d8
EXT	sources/extension.move	4d7b12877d864bd164bf546a4ac14b874aa75cbe
COR	sources/core.move	9150f948847895a50086dc4ac34d2d4a3d058191
FEE	sources/fees.move	feebfab5770300d43fa761e9aa949b9df6997a53
TRE	sources/treasury.move	7d3229285a86c82343264fa4ad8a26c6d819e929

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	3	3	0
Informational	0	0	0
Minor	1	1	0
Medium	1	1	0
Major	1	1	0
Critical	0	0	0

1.4 MoveBit Audit Breakdown

MoveBit aims to assess repositories for security-related issues, code quality, and compliance with specifications and best practices. Possible issues our team looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Integer overflow/underflow by bit operations
- Number of rounding errors
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting
- Unchecked CALL Return Values
- The flow of capability
- Witness Type

1.5 Methodology

The security team adopted the **"Testing and Automated Analysis"**, **"Code Review"** and **"Formal Verification"** strategy to perform a complete security test on the code in a way that is closest to the real attack. The main entrance and scope of security testing are stated in the conventions in the "Audit Objective", which can expand to contexts beyond the scope according to the actual testing needs. The main types of this security audit include:

(1) Testing and Automated Analysis

Items to check: state consistency / failure rollback / unit testing / value overflows / parameter verification / unhandled errors / boundary checking / coding specifications.

(2) Code Review

The code scope is illustrated in section 1.2.

(3) Formal Verification(Optional)

Perform formal verification for key functions with the Move Prover.

(4) Audit Process

- Carry out relevant security tests on the testnet or the mainnet;
- If there are any questions during the audit process, communicate with the code owner in time. The code owners should actively cooperate (this might include providing the latest stable source code, relevant deployment scripts or methods, transaction signature scripts, exchange docking schemes, etc.);
- The necessary information during the audit process will be well documented for both the audit team and the code owner in a timely manner.

2 Summary

This report has been commissioned by BlockUs to identify any potential issues and vulnerabilities in the source code of the BlockUs smart contract, as well as any contract dependencies that were not part of an officially recognized library. In this audit, we have utilized various techniques, including manual code review and static analysis, to identify potential vulnerabilities and security issues.

During the audit, we identified 3 issues of varying severity, listed below.

ID	Title	Severity	Status
COR-1	Functions Lack of Version Check	Major	Fixed
COR-2	Incorrect Comments	Medium	Fixed
COR-3	Duplicate Code	Minor	Fixed

3 Participant Process

Here are the relevant actors with their respective abilities within the BlockUs Smart Contract
:

MarketOwner

- Owner can update `base_fee` through `update_base_fee()` function.
- Owner can withdraw the profits through `withdraw_profits()` function.

User

- User can pay money to buy an `item` through `purchase()` function.
- User can list an item on KoiMarketplace through `list()/delist` function, create a Kiosk for it, and install the extension to sell it.

4 Findings

COR-1 Functions Lack of Version Check

Severity: Major

Status: Fixed

Code Location:

sources/core.move

Descriptions:

We have noticed that the contracts under the `core.move` file do not check for the version, only the `purchase` and `update_base_fee` functions have version checks.

Suggestion:

It is recommended that any function that uses `KoiMarketplace` should be version-checked.

COR-2 Incorrect Comments

Severity: Medium

Status: Fixed

Code Location:

sources/core.move#63

Descriptions:

The function comment describes the value of `base_fee` as 2%, but according to the actual code `base_fee` has a value of `20_000_000/100_000_000_000`, which is 0.02%

```
public(package) fun calculate_fee(
    price: u64,
    fee_structure: &KoiMarketplaceFeeStructure,
): (u64) {
    let base_fee = fee_structure.base_fee;
    let fee = (base_fee * price) / 100_000_000_000;
    (fee)
}
```

On the other hand, in the current `fee` calculation, hard division is used for calculation, but there is no limit on the minimum range of `price`. According to the current value, `price < 10000` does not make any sense, because `price = 10000` is also required to ensure `fee = 0`.

Suggestion:

It is recommended to modify the incorrect comments.

COR-3 Duplicate Code

Severity: Minor

Status: Fixed

Code Location:

sources/core.move#90

Descriptions:

There are two duplicate code checks in the function.

```
public entry fun update_base_fee(
      _: &KoiMarketplaceOwnerCap,
      marketplace: &mut KoiMarketplace,
      base_fee: u64,
   ) {
      assert!(marketplace.version == VERSION, EKoiMarketplaceVersionMismatch);
      assert!(marketplace.version == VERSION, EKoiMarketplaceVersionMismatch);
```

Suggestion:

It is recommended to remove duplicate code.

Appendix 1

Issue Level

- **Informational** issues are often recommendations to improve the style of the code or to optimize code that does not affect the overall functionality.
- **Minor** issues are general suggestions relevant to best practices and readability. They don't post any direct risk. Developers are encouraged to fix them.
- **Medium** issues are non-exploitable problems and not security vulnerabilities. They should be fixed unless there is a specific reason not to.
- **Major** issues are security vulnerabilities. They put a portion of users' sensitive information at risk, and often are not directly exploitable. All major issues should be fixed.
- **Critical** issues are directly exploitable security vulnerabilities. They put users' sensitive information at risk. All critical issues should be fixed.

Issue Status

- **Fixed:** The issue has been resolved.
- **Partially Fixed:** The issue has been partially resolved.
- **Acknowledged:** The issue has been acknowledged by the code owner, and the code owner confirms it's as designed, and decides to keep it.

Appendix 2

Disclaimer