Security Audit Report for Prometeus Network Takeus Contracts

Date: Oct 13, 2022

Version: 1.1

Contact: contact@blocksec.com

BlockSec
Audit Team

Digitally signed by
BlockSec Audit Team
Date: 2022.10.13 19:29:22
+08'00'

Contents

Report Manifest

Item	Description
Client	Prometeus Network
Target	Prometeus Network Takeus Contracts

Version History

Version	Date	Description
1.0	Apr 7, 2022	First Release
1.1	Oct 13, 2022	Remove code snippets

About BlockSec The BlockSec Team focuses on the security of the blockchain ecosystem, and collaborates with leading DeFi projects to secure their products. The team is founded by top-notch security researchers and experienced experts from both academia and industry. They have published multiple blockchain security papers in prestigious conferences, reported several zero-day attacks of DeFi applications, and released detailed analysis reports of high-impact security incidents. They can be reached at Email, Twitter and Medium.

Chapter 1 Introduction

1.1 About Target Contracts

Information	Description
Type	Smart Contract
Language	Solidity
Approach	Semi-automatic and manual verification

Prometeus Network Takeus is a marketplace for users to lend their NFTs to other users. Specifically, the lenders list their NFTs on the market, and the borrowers can register vaults to borrow those NFTs. The vault is developed based on the GnosisSafe project to restrict the interactions with other contracts.

The auditing process is iterative. Specifically, we will audit the commits that fix the discovered issues. If there are new issues, we will continue this process. The commit SHA values of the repo [1] during the audit are shown in the following.

Project		Commit SHA
takeus-contracts	Version 1	e53fd969dc4956aea1220b8b36bc93d4f252b29d
	Version 2	eb492e5b782e5f4e8311f88e6c253973b973f783

1.2 Disclaimer

This audit report does not constitute investment advice or a personal recommendation. It does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Any entity should not rely on this report in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset.

This audit report is not an endorsement of any particular project or team, and the report do not guarantee the security of any particular project. This audit does not give any warranties on discovering all security issues of the smart contracts, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of smart contracts.

The scope of this audit is limited to the code mentioned in Section 1.1. Unless explicitly specified, the security of the language itself (e.g., the solidity language), the underlying compiling toolchain and the computing infrastructure are out of the scope.

1.3 Procedure of Auditing

We perform the audit according to the following procedure.

[1] https://github.com/Prometeus-Network/takeus-contracts

- **Vulnerability Detection** We first scan smart contracts with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of smart contracts and conduct further investigation on the possible vulnerabilities using an automatic fuzzing tool (developed by our research team). We also manually analyze possible attack scenarios with independent auditors to cross-check the result.
- **Recommendation** We provide some useful advice to developers from the perspective of good programming practice, including gas optimization, code style, and etc.

We show the main concrete checkpoints in the following.

1.3.1 Software Security

- Reentrancy
- DoS
- Access control
- Data handling and data flow
- Exception handling
- Untrusted external call and control flow
- Initialization consistency
- Events operation
- Error-prone randomness
- Improper use of the proxy system

1.3.2 DeFi Security

- Semantic consistency
- Functionality consistency
- Permission management
- Business logic
- Token operation
- Emergency mechanism
- Oracle security
- Whitelist and blacklist
- Economic impact
- Batch transfer

1.3.3 NFT Security

- Duplicated item
- Verification of the token receiver
- Off-chain metadata security

1.3.4 Additional Recommendation

- Gas optimization

- Code quality and style

Note *The previous checkpoints are the main ones. We may use more checkpoints during the auditing process according to the functionality of the project.*

1.4 Security Model

To evaluate the risk, we follow the standards or suggestions that are widely adopted by both industry and academy, including OWASP Risk Rating Methodology [2] and Common Weakness Enumeration [3]. The overall *severity* of the risk is determined by *likelihood* and *impact*. Specifically, likelihood is used to estimate how likely a particular vulnerability can be uncovered and exploited by an attacker, while impact is used to measure the consequences of a successful exploit.

In this report, both likelihood and impact are categorized into two ratings, i.e., *high* and *low* respectively, and their combinations are shown in Table 1.1.

Table 1.1: Vulnerability Severity Classification

Impact \ Likelihood	*High*	*Low*
High	High	Medium
Low	Medium	Low

Accordingly, the severity measured in this report are classified into three categories: **High**, **Medium**, **Low**. For the sake of completeness, **Undetermined** is also used to cover circumstances when the risk cannot be well determined.

Furthermore, the status of a discovered issue will fall into one of the following four categories:

- **Undetermined** No response yet.
- **Acknowledged** The issue has been received by the client, but not confirmed yet.
- **Confirmed** The issue has been recognized by the client, but not fixed yet.
- **Fixed** The issue has been confirmed and fixed by the client.

[2] https://owasp.org/www-community/OWASP_Risk_Rating_Methodology

[3] https://cwe.mitre.org/

Chapter 2 Findings

In total, we find `eight` potential issues. Besides, we also have `four` recommendations.

- High Risk: 1
- Medium Risk: 3
- Low Risk: 4
- Recommendation: 4

ID	Severity	Description	Category	Status
1	Low	Inconsistent Value of the Function Parameter	Software Security	Fixed
2	Medium	Missing Checks on Parameters	Software Security	Fixed
3	Low	Inconsistent Comment with Implementation	Software Security	Fixed
4	Low	Insufficient Listing Identification Information	DeFi Security	Fixed
5	Medium	Potential Front-Running Attacks Against Borrowers	DeFi Security	Fixed
6	Medium	Stale Status Information in the VaultManager Contract	DeFi Security	Fixed
7	High	Unlimited ERC1155 Withdrawal	DeFi Security	Fixed
8	Low	Incorrect Information Updated in the updateERC1155() Function	DeFi Security	Fixed
9	-	Removing Unused Variables	Recommendation	Fixed
10	-	Avoiding Unnecessary Code Logic	Recommendation	Fixed
11	-	Considering Whitelisting Valid Tokens	Recommendation	Fixed
12	-	Removing Redundant Code Logic	Recommendation	Fixed

The details are provided in the following sections. Note that, **the code snippets related to the findings have been removed from the report upon the request of the project**.

2.1 Software Security

2.1.1 Inconsistent Value of the Function Parameter

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the `TakeUsMarketplace` contract, the `_duration` field in the following several functions, including `updateERC1155()`, `listERC1155()`, `listERC721()` and `updateERC721()`, are inconsistent. For example, the `_duration` field in the `Lending` struct is inconsistent with the event emitted later in function `updateERC1155()`.

Impact N/A

Suggestion Fix the usages of the `_duration` parameter.

2.1.2 Missing Checks on Parameters

Status Fixed in `Version 2`

Introduced by `Version 1`

Description There are some important parameters that require checks but none is performed. For example, in the `listERC1155()` function:

1. The `_duration` should not be zero or other extreme values.
2. The `_paymentToken` isn't checked or whitelisted, which means the lender can set an arbitrary token as the `paymentToken` when lending ERC1155 NFTs.

Impact Arbitrary inputs without verification may lead to unexpected results.

Suggestion Add necessary sanity checks.

2.1.3 Inconsistent Comment with Implementation

Status Fixed in `Version 2`

Introduced by `Version 1`

Description There are some comments which are inconsistent with the implementation. For example, the `SafeVault` allows the borrower to return the item directly without calling the `returnItemERC721()` function, which is conflict with the comment in line 87 in the `VaultManager` contract.

Impact N/A

Suggestion Fix the wrong comments.

2.2 DeFi Security

2.2.1 Insufficient Listing Identification Information

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The lending of each NFT is solely identified by its token address and token ID. However, there does not exist any way (e.g., "listing ID" or "listing nonce") to distinguish the listings from each other, which may lead to unexpected consequences. For example, suppose there is a NFT that was originally owned by a user A, and this NFT has been sold to another user B. If A has listed that NFT on `TakeUsMarketplace` before the selling, B MUST cancel the listing if he wants to use the market as well (note that at this moment B already becomes the owner of the NFT, and is allowed to cancel the listing).

Impact It may bring inconvenience to users.

Suggestion Provide more identification information for a NFT lending registration.

2.2.2 Potential Front-Running Attacks Against Borrowers

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The borrower is subject to the front-running attack. A malicious lender may keep monitoring a borrower for his borrowing transaction and front-run it with a transaction that modifies the lending parameters, e.g.,

1. If the lender modifies the price, and the borrower has the maximum allowance to the marketplace, it may cost all the `paymentToken` the borrower owns to borrow the item.
2. If the lender modifies the `_duration` to 0 as described in Section 2.1.2, a borrower may lose the `paymentToken` without actually holding the item.
3. The lender may modify the `paymentToken` by calling the `cancelERC1155()` function after the invocation of the `listERC1155()` function. The malicious lender can get the borrower's ERC721 NFT by setting the `paymentToken` address to be the corresponding ERC721 address, if the borrower approves his ERC721 NFT to the marketplace.

Note that `Version 2` still suffers from this issue. Specifically, the `nonce` variable in the following code snippet is a `memory` variable. Checking a memory variable at the end of this function will not have any effect. What's more, it cannot be used to prevent the front-run attack. A better way to achieve the goal is to make the `nonce` as the function parameter, and check the `nonce` at the very beginning of the `borrow` functions.

Besides, the `nonce` variable is initialized to zero. As an empty `Lending` struct is also specified with a zero `nonce`, this field cannot be used to check whether the mapping entry is present or not.

Impact The functions may suffer from the front-running attack.

Suggestion N/A

2.2.3 Stale Status Information in the VaultManager Contract

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The variables `lendingERC721` and `lendingERC1155` in the `VaultManager` contract are not reset when an item is returned, with or without calling the corresponding functions (i.e., the `returnItem()` function family). These variables are used in the `checkIfNFTisLocked()` function to check if the NFT is locked. It may cause some problems if such status information of the returned item was not erased, e.g.,

1. They are public variables and may mislead the users.
2. Suppose a NFT has been borrowed from the marketplace, and returned to the lender. After that, if the NFT is transferred into the `SafeVault` DIRECTLY, it will be locked in the `SafeVault`. What's worse, since the `lendingERC721` and `lendingERC1155` variables record the former owner (i.e., the lender), after the duration of the lending, the NFT can be transferred from the current owner to the former owner by an arbitrary user.

Impact Stale status information may cause misleading or even worse behaviors.

Suggestion Update the status information properly.

2.2.4 Unlimited ERC1155 Withdrawal

Status Fixed in `Version 2`

Introduced by `Version 1`

Description At any moment, an ERC721 token in a form of `<address-tokenId>` pair can only be exclusively owned by a single identity. However, a `<address-tokenId>` pair representing a group of ERC1155 tokens can be held by different identities.

In the `borrowERC1155()` function, the variabl named `listingERC1155` merely holds one slot for a `<address-tokenId>` ERC1155 token. Two users holding tokens of the same group can overwrite the each other's listing by simply invoking the `listERC1155()` function (see Section 2.1.2). And the `borrowERC1155()` function only checks if `_nftAddress` is eligible and if the borrower is registered. As a result, a user could borrow any token listed to a registered borrower without the borrower's agreement.

Furthermore, the `returnItemERC1155()` function only checks the expiration time of a lending slot. The lender of that slot can drain the same ERC1155 tokens from the `SafeVault` by calling this function repeatedly.

Impact If a malicious user get to know there is a `<address-tokenId>` ERC1155 token in a `SafeVault`, he can firstly get a same token and call the `listERC1155()` function by specifying his address as the `lender` field of the listing. Then the user can pay the lending fee to borrow his token to the `SafeVault`. After the duration, he can repeatedly invoke the `returnItemERC1155()` function and drain the balance.

Suggestion The `lending` mapping in `VaultManager` can be overwritten by a different lender of the same token. However, this overwriting can be prevented by moving the lender field of the `Lending` struct as a part of the mapping index.

Communication with the Project

Developer: Added whitelist for ERC721 and ERC1155 collections. Now checking if ERC1155 `tokenId` is disabled for the platform. As for now I have a lazy fix, where we could restrict listing fungible 1155 tokens. Anyway, it is not a huge issue anyhow, since TakeUs is the platform to rent non-fungible tokens.

Auditor: It is a good point to delete the `lending` slot at the end of `returnItemERC1155`. However, this fix can only prevent continuous withdrawing erc1155 via external calls to that function. As a result of ERC1155's `callback` characteristic, the lender can reenter this function by its `onERC1155Received` function. This can be easily fixed by deleting the `lending` slot before transferring as a practice of Checks-Effects-Interactions pattern.

Developer: That's not really a bad thing for TakeUs, since the platform is created for renting of non-fungible assets. So I thought, that it would be sufficient what I just did.

2.2.5 Incorrect Information Updated in the updateERC1155() Function

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The `updateERC1155()` function will modify the lender of the original listing as `msg.sender`, which seems to be problematic due to the following two reasons. First, this is inconsistent with the `updateERC721()` function. Second, due to the fact that ERC1155 tokens may have multiple owners for the same ID, this may result in unexpected results (see the `updateERC1155()` function in Section 2.1.1).

Impact Owners of the same listed ERC1155 token are able to cancel a listing by first updating that listing, and then cancelling it.

Suggestion Make different users operate in different spaces.

2.3 Additional Recommendation

2.3.1 Removing Unused Variables

Status Fixed in `Version 2`

Introduced by `Version 1`

Description Some variables are not used in the code logic, including:

1. In the `SafeVault` contract: `_deprecatedDomainSeparator` and `signedMessages`
2. In the `AddressRegistry` contract: `tokenRegistry` and `error AlreadyListed()`

Impact N/A

Suggestion Remove those variables.

2.3.2 Avoiding Unnecessary Code Logic

Status Fixed in `Version 2`

Introduced by `Version 1`

Description There's no necessity to divide ERC721 and ERC1155 `lending` into two mappings since `_nftAddress` is distinct. Besides, the conversion from `address` type to IERC721/IERC1155 is of no use.

Impact N/A

Suggestion Adjust the code logic.

2.3.3 Considering Whitelisting Valid Tokens

Status Fixed in `Version 2`

Introduced by `Version 1`

Description Currently no mechanism is specified to verify valid NFT tokens. To mitigate the potential risk of the fake tokens, it is recommended to apply a whitelist to specify the valid ones explicitly.

Impact N/A

Suggestion Use a whitelist to indicate the valid tokens explicitly.

2.3.4 Removing Redundant Code Logic

Status Fixed in `Version 2`

Introduced by `Version 2`

Description `Version 2` fixed recommend 2.3.2, while leaving a repetitive code logic, i.e., the `if` statements. Note that this code snippet comes from `Version 2`.

Impact N/A

Suggestion Remove/Adjust the corresponding code logic.