Security Audit Report for TakeUs Trade Marketplace Contracts

Date: August 30, 2022

Version: 1.0

Contact: contact@blocksec.com

BlockSec
Audit Team

Digitally signed by BlockSec Audit Team
Date: 2022.08.30 15:38:15 +08'00'

Contents

1 Introduction **1**

1.1 About Target Contracts . . . 1
1.2 Disclaimer . . . 1
1.3 Procedure of Auditing . . . 2
1.3.1 Software Security . . . 2
1.3.2 DeFi Security . . . 2
1.3.3 NFT Security . . . 2
1.3.4 Additional Recommendation . . . 3
1.4 Security Model . . . 3

2 Findings **4**

2.1 Software Security . . . 4
2.1.1 Unchecked and inconsistent fee parameters . . . 4
2.2 DeFi Security . . . 5
2.2.1 Unchecked end time when accepting offers . . . 5
2.2.2 Unchecked quantity when selling ERC-721 NFTs . . . 6
2.2.3 Potential listing content manipulation . . . 7
2.2.4 Non-removed order entry after listing deletion . . . 9
2.2.5 Lack of mechanism to ensure the integrity of the offer content . . . 11
2.3 Additional Recommendation . . . 12
2.3.1 Check parameters when setting fee recipient address . . . 12
2.3.2 Remove duplicate checks . . . 12
2.3.3 Fix wrong owner address passed to the `isApprovedForAll` function . . . 14
2.3.4 Fix improper parameters passed to the `_checkOwning` function . . . 14
2.3.5 Revise the royalty fee collection logic . . . 15
2.3.6 Fix the incorrect reference of the interface name . . . 16
2.3.7 Check the quantities of the tokens to be listed . . . 16
2.3.8 Implement a fine-grained approval for ERC-721 tokens . . . 18
2.3.9 Revise code logic of the `acceptOffer` function for ERC-1155 tokens . . . 19

Report Manifest

Item	Description
Client	Prom
Target	TakeUs Trade Marketplace Contracts

Version History

Version	Date	Description
1.0	August 30, 2022	First Release

About BlockSec BlockSec focuses on the security of the blockchain ecosystem and collaborates with leading DeFi projects to secure their products. BlockSec 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 successfully protected digital assets that are worth more than 5 million dollars by blocking multiple attacks. 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

The target of this audit is the new marketplace contracts of the Prom protocol in the repository [1]. Note that the audit scope is limited to contracts under the `contracts/trade-marketplace` folder, while other contracts and files are out of the scope.

The auditing process is iterative. Specifically, we would audit the commits that fix the discovered issues. If there are new issues, we will continue this process. The commit SHA values during the audit are shown in the following table. Our audit report is responsible for the code in the initial version (`Version 1`), as well as new code (in the following versions) to fix issues in the audit report.

Project	Version	Commit Hash
TakeUs Trade Marketplace Contracts	`Version 1`	`e2bed45b063bd8a621f43249cbfbcdef5ed51a5c`
	`Version 2`	`1f974417111f775ebc1257bf14bf15b61901dcd7`
	`Version 3`	`96c6e0ddccbcb2921511e3b56725d3a1f6b295a2`

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 does 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] https://github.com/prom-io/takeus-contracts

1.3 Procedure of Auditing

We perform the audit according to the following procedure.

- **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 item will fall into one of the following four categories:

- **Undetermined** No response yet.
- **Acknowledged** The item has been received by the client, but not confirmed yet.
- **Confirmed** The item has been recognized by the client, but not fixed yet.
- **Fixed** The item 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 **six** potential issues. We have **nine** recommendations.

- High Risk: 5
- Medium Risk: 0
- Low Risk: 1
- Recommendations: 9
- Notes: 0

ID	Severity	Description	Category	Status
1	Low	Unchecked and inconsistent fee parameters	Software Security	Fixed
2	High	Unchecked end time when accepting offers	DeFi Security	Fixed
3	High	Unchecked quantity when selling ERC-721 NFTs	DeFi Security	Fixed
4	High	Potential listing content manipulation	DeFi Security	Fixed
5	High	Non-removed order entry after listing deletion	DeFi Security	Fixed
6	High	Lack of mechanism to ensure the integrity of the offer content	DeFi Security	Fixed
7	-	Check parameters when setting fee recipient address	Recommendation	Fixed
8	-	Remove duplicate checks	Recommendation	Fixed
9	-	Fix wrong owner address passed to the `isApprovedForAll` function	Recommendation	Fixed
10	-	Fix improper parameters passed to the `_checkOwning` function	Recommendation	Fixed
11	-	Revise the royalty fee collection logic	Recommendation	Fixed
12	-	Fix the incorrect reference of the interface name	Recommendation	Fixed
13	-	Check the quantities of the tokens to be listed	Recommendation	Fixed
14	-	Implement a fine-grained approval for ERC-721 tokens	Recommendation	Fixed
15	-	Revise code logic of the `acceptOffer` function for ERC-1155 tokens	Recommendation	Acknowledged

The details are provided in the following sections.

2.1 Software Security

2.1.1 Unchecked and inconsistent fee parameters

Severity Low

Status Fixed in `Version 3`

Introduced by `Version 1`

Description The marketplace charges fees for each NFT trade under different circumstances. Specifically, for a given contract, the fee is charged as a proportion of the NFT price, which is presented in this

contract as a variable named `platformFee`. There are two problems with the `platformFee` variable, as follows:

1. In the constructor of the `PromBundleMarketplace` contract, there is no check on the range of the `_platformFee` parameter. However, the `platformFee` state variable is assumed within the range of $(0, 1000]$. Besides, the function for setting this parameter (i.e., the `updatePlatformFee` function of the `BundleMarketplaceGuard` contract) also does not have such a check.

```
12    constructor(
13        address payable _feeRecipient,
14        uint256 _platformFee,
15        address _addressRegistry
16    ) {
17        platformFee = _platformFee;
18        feeRecipient = _feeRecipient;
19        addressRegistry = IPromAddressRegistry(_addressRegistry);
20    }
```

Listing 2.1: PromBundleMarketplace.sol

```
22    function updatePlatformFee(uint16 _platformFee) external onlyOwner {
23        platformFee = _platformFee;
24        emit UpdatePlatformFee(_platformFee);
25    }
```

Listing 2.2: BundleMarketplaceGuard.sol

2. In the constructor of the `PromTradeMarketplace` contract, there is no check on the range of the `_platformFee` parameter. However, the `platformFee` state variable is assumed within the range of $(0, 10000]$. Besides, the function for setting this parameter (i.e., the `updatePlatformFee` function of the `TradeMarketplaceGuard` contract) also does not have such a check.

```
 7    constructor(uint16 _platformFee, address _addressRegistry) {
 8        platformFee = _platformFee;
 9        addressRegistry = IPrometaAddressRegistry(_addressRegistry);
10    }
```

Listing 2.3: PromTradeMarketplace.sol

```
124    function updatePlatformFee(uint256 _platformFee) external onlyOwner {
125        platformFee = _platformFee;
126        emit UpdatePlatformFee(_platformFee);
127    }
```

Listing 2.4: TradeMarketplaceGuard.sol

Impact Incorrect `_platformFee`s may cause financial losses to the users.

Suggestion Add the corresponding sanity checks.

2.2 DeFi Security

2.2.1 Unchecked end time when accepting offers

Severity High

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the `buyItem` function of the `PromBundleMarketplace` contract, there is a check if the `startingTime` of the listing has passed. However, there is no check whether the current listing has expired, i.e., the current block timestamp is larger than the `endTime` of the listing. As a result, the `endTime` of a listing becomes ineffective, because an expired listing would still be accepted.

```
114    function buyItem(bytes32 _bundleID, uint256 _nonce) external nonReentrant {
115        address owner = owners[_bundleID];
116        require(owner != address(0), "invalid id");
117        Listing memory listing = listings[owner][_bundleID];
118
119        require(listing.price > 0, "not listed");
120        require(block.timestamp >= listing.startingTime, "not buyable");
121
122        uint256 price = listing.price;
123        uint256 feeAmount = (price * platformFee) / 1e3;
124
125        _transfer(msg.sender, feeRecipient, feeAmount, listing.payToken);
126        _transfer(msg.sender, owner, price - feeAmount, listing.payToken);
```

Listing 2.5: PromBundleMarketplace.sol

Impact Expired offers can still be accepted.

Suggestion Add the sanity check accordingly.

2.2.2 Unchecked quantity when selling ERC-721 NFTs

Severity High

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the `PromTradeMarketplace` contract, when an order is settled, the buyer first needs to pay for the order through the `_handleListingPayment` function in the `TradeMarketplaceUtils` contract. After that, the NFT is transferred to the buyer through the `_transferNft` function in the `TradeMarketplaceUtils` contract.

However, there exists a vulnerability when the traded NFTs are ERC-721 tokens. Specifically, in such a case, the quantity specified in the listing is ignored when transferring the tokens while it is counted in calculating the total price. There are two possible scenarios for the quantity of the listing:

- if the quantity is set to be zero, the listing is still valid and anyone can buy the NFT for free.
- if the quantity is set to be *larger than one*, the actual price would be larger than the price displayed.

```
73function _handleListingPayment(
74    Listing memory _listing,
75    address _owner,
76    address _nftAddress
```

```
77  ) internal {
78    uint256 price = _listing.pricePerItem * _listing.quantity;
79    (uint256 royaltyFee, uint256 feeAmount) = getFees(price, _nftAddress);
80
81    _handlePayment(
82      _nftAddress,
83      _listing.payToken,
84      msg.sender,
85      _owner,
86      price,
87      royaltyFee,
88      feeAmount
89    );
90  }
```

Listing 2.6: TradeMarketplaceUtils.sol

```
15    function _transferNft(
16        address _nftAddress,
17        address _from,
18        address _to,
19        uint256 _tokenId,
20        uint256 _quantity
21    ) internal {
22        if (IERC165(_nftAddress).supportsInterface(INTERFACE_ID_ERC721)) {
23            IERC721(_nftAddress).safeTransferFrom(_from, _to, _tokenId);
24        } else {
25            IERC1155(_nftAddress).safeTransferFrom(
26                _from,
27                _to,
28                _tokenId,
29                _quantity,
30                bytes("0x")
31            );
32        }
33    }
```

Listing 2.7: TradeMarketplaceUtils.sol

Impact Unchecked quantity may lead to unexpected results for handling ERC-721 NFT payments.

Suggestion Add sanity checks for the quantity and the corresponding price of the listing.

2.2.3 Potential listing content manipulation

Severity High

Status Fixed in `Version 2`

Introduced by `Version 1`

Description Using the bundle marketplace, a seller can hold multiple NFTs and list them as a bundle, which could be purchased as a whole by a buyer. Specifically, if there is any buyer that provides a offer for the bundle, the offer would be accepted by the seller through the `acceptOffer` function in the

BundleMarketplaceOffers contract. However, the bundle ID can be arbitrarily specified by the seller (see Listing 2.9) by first cancelling the bundle and then re-listing the modified one with the same bundle ID. Therefore, the seller can modify the listing content before calling the acceptOffer function. This inevitably causes severe listing content manipulation problems that the seller is able to sell arbitrary NFTs to the buyer.

```
54  function acceptOffer(bytes32 _bundleID, address _creator)
55    external
56    nonReentrant
57  {
58    require(owners[_bundleID] == msg.sender, "not owning item");
59
60    Offer memory offer = offers[_bundleID][_creator];
61    require(offer.deadline > block.timestamp, "offer not exists or expired");
62
63    uint256 price = offer.price;
64    uint256 feeAmount = (price * platformFee) / 1e3;
65
66    IERC20(offer.payToken).safeTransferFrom(_creator, feeRecipient, feeAmount);
67    IERC20(offer.payToken).safeTransferFrom(
68      _creator,
69      msg.sender,
70      price - feeAmount
71    );
72
73    Listing memory listing = listings[msg.sender][_bundleID];
74    for (uint256 i; i < listing.nfts.length; i++) {
75      address _nft = listing.nfts[i];
76      uint256 _tokenId = listing.tokenIds[i];
77
78      _transferNft(_nft, msg.sender, _creator, _tokenId, listing.quantities[i]);
79
80      IPromLendingMarketplace(addressRegistry.tradeMarketplace())
81        .validateItemSold(_nft, _tokenId, owners[_bundleID]);
82    }
83    emit ItemSold(msg.sender, _creator, _bundleID, price, offer.payToken);
84    _deleteListing(msg.sender, _bundleID, listing);
85    delete (offers[_bundleID][_creator]);
86
87    emit OfferCanceled(_creator, _bundleID);
88  }
```

Listing 2.8: BundleMarketplaceOffers.sol

```
29  function listItem(
30    bytes32 _bundleID,
31    address[] calldata _nftAddresses,
32    uint256[] calldata _tokenIds,
33    uint256[] calldata _quantities,
34    address _payToken,
35    uint256 _price,
36    uint256 _startingTime,
37    uint256 _endTime
```

```
38  ) external validArgs(_price, _payToken) {
39    bundleIds[_bundleID] = _bundleID;
40    require(
41      _nftAddresses.length == _tokenIds.length &&
42        _tokenIds.length == _quantities.length,
43      "invalid data"
44    );
45
46    for (uint256 i = 0; i < _nftAddresses.length; i++) {
47      require(
48        addressRegistry.isTradeCollectionEnabled(_nftAddresses[i]),
49        "collection not enabled"
50      );
51    }
52
53    require(owners[_bundleID] == address(0), "already listed");
54
55    _listItem(_bundleID, _nftAddresses, _tokenIds, _quantities);
56    listings[msg.sender][_bundleID] = Listing({
57      nfts: _nftAddresses,
58      tokenIds: _tokenIds,
59      quantities: _quantities,
60      payToken: _payToken,
61      price: _price,
62      startingTime: _startingTime,
63      endTime: _endTime,
64      nonce: block.number
65    });
66
67    owners[_bundleID] = msg.sender;
68
69    emit ItemListed(
70      msg.sender,
71      _bundleID,
72      _payToken,
73      _price,
74      _startingTime,
75      _endTime
76    );
77  }
```

Listing 2.9: PromBundleMarketplace.sol

Impact The listing content can be manipulated by the seller.

Suggestion Revise the design.

2.2.4 Non-removed order entry after listing deletion

Severity High

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the `PromBundleMarketplace` contract, an offer entry would not be deleted after the listing has been fulfilled by a user through the `buyItem` function. This results in that any offers previously created (for this bundle ID) are still acceptable. Therefore, there exists an attack vector that after a listing has been fulfilled, a malicious user (not necessarily the origin seller) can create another listing with the same bundle ID (see Listing 2.9). After that, if there exists any current outstanding offer, these offers can be traded with different listing contents (or even empty ones).

```
114 function buyItem(bytes32 _bundleID, uint256 _nonce) external nonReentrant {
115   address owner = owners[_bundleID];
116   require(owner != address(0), "invalid id");
117   Listing memory listing = listings[owner][_bundleID];
118
119   require(listing.price > 0, "not listed");
120   require(block.timestamp >= listing.startingTime, "not buyable");
121
122   uint256 price = listing.price;
123   uint256 feeAmount = (price * platformFee) / 1e3;
124
125   _transfer(msg.sender, feeRecipient, feeAmount, listing.payToken);
126   _transfer(msg.sender, owner, price - feeAmount, listing.payToken);
127
128   for (uint256 i; i < listing.nfts.length; i++) {
129     _transferNft(
130       listing.nfts[i],
131       owner,
132       msg.sender,
133       listing.tokenIds[i],
134       listing.quantities[i]
135     );
136     IPromLendingMarketplace(addressRegistry.tradeMarketplace())
137       .validateItemSold(listing.nfts[i], listing.tokenIds[i], owner);
138   }
139   require(listing.nonce == _nonce, "listing was updated");
140   emit ItemSold(owner, msg.sender, _bundleID, price, listing.payToken);
141   _deleteListing(owner, _bundleID, listing);
142
143   emit OfferCanceled(msg.sender, _bundleID);
144 }
```

Listing 2.10: PromBundleMarketplace.sol

```
119 function _deleteListing(
120   address _owner,
121   bytes32 _bundleID,
122   Listing memory listing
123 ) internal {
124   for (uint256 i; i < listing.nfts.length; i++) {
125     bundleIdsPerItem[listing.nfts[i]][listing.tokenIds[i]].remove(_bundleID);
126     delete (nftIndexes[_bundleID][listing.nfts[i]][listing.tokenIds[i]]);
127   }
128   delete (listings[_owner][_bundleID]);
129   delete (owners[_bundleID]);
130   delete (bundleIds[_bundleID]);
```

```
131 }
```

Listing 2.11: BundleMarketplaceUtils.sol

Note that the `acceptOffer` function also suffers from this problem. Although the satisfied offer entry is removed, there may still exist some unsatisfied offers which are vulnerable to the attack vector.

Impact A fulfilled listing can be overwritten (by any other users) to trade current outstanding offers.

Suggestion Make bundle IDs unique and delete all offer information for fulfilled listings.

2.2.5 Lack of mechanism to ensure the integrity of the offer content

Severity High

Status Fixed in `Version 3`

Introduced by `Version 1`

Description In the `TradeMarketplaceOffers` contract, a buyer is able to create an offer for a specified NFT. If this offer satisfies the owner of the NFT, the `acceptOffer` function would be invoked by the owner to sell the NFT accordingly. However, the `acceptOffer` function does not provide any mechanism for ensuring the integrity of the offer content after submitting the request. Therefore, there is an attack vector that may cause financial losses to the NFT owner.

Specifically, an attacker can first create an offer with a pretty price and then lower the amount of the payment token approved to the contract. By doing so, the payment would not be actually executed. After that, the attacker can keep on listening the mempool, waiting for the NFT owner to submit the `acceptOffer` request. Once the request is submitted, the attacker can front-run the request with a higher gas price to cancel and re-create the offer with a much lower price. Since the `acceptOffer` function is not aware of the modification of the offer, the NFT would finally be sold at an unexpected price.

```
75  function acceptOffer(
76    address _nftAddress,
77    uint256 _tokenId,
78    address _creator
79  )
80    external
81    offerExists(_nftAddress, _tokenId, _creator)
82    onlyAssetOwner(
83      _nftAddress,
84      _tokenId,
85      offers[_nftAddress][_tokenId][_creator].quantity
86    )
87  {
88    Offer memory offer = offers[_nftAddress][_tokenId][_creator];
89    _handleOfferPayment(offer, _creator, _nftAddress);
90
91    _transferNft(_nftAddress, msg.sender, _creator, _tokenId, offer.quantity);
92
93    emit ItemSold(
94      msg.sender,
95      _creator,
96      _nftAddress,
```

```
97      _tokenId,
98      offer.quantity,
99      address(offer.payToken),
100     offer.pricePerItem
101   );
102   emit OfferCanceled(_creator, _nftAddress, _tokenId);
103
104   delete (listings[_nftAddress][_tokenId][msg.sender]);
105   delete (offers[_nftAddress][_tokenId][_creator]);
106 }
```

Listing 2.12: TradeMarketplaceOffers.sol

Impact For an NFT token listed on the market, the attacker can front-run the `acceptOffer` function to buy it with an expected price.

Suggestion Add a nonce or a hash to ensure the integrity of the offer content.

2.3 Additional Recommendation

2.3.1 Check parameters when setting fee recipient address

Status Fixed in `Version 3`[1]

Introduced by `Version 1`

Description In the `updatePlatformFeeRecipient` function of the `BundleMarketplaceGuard` contract, there is no check to verify the validity of the new `feeRecipient` address.

```
32    function updatePlatformFeeRecipient(address payable _platformFeeRecipient)
33        external
34        onlyOwner
35    {
36        feeRecipient = _platformFeeRecipient;
37        emit UpdatePlatformFeeRecipient(_platformFeeRecipient);
38    }
```

Listing 2.13: BundleMarketplaceGuard.sol

Impact N/A

Suggestion Add sanity checks accordingly.

2.3.2 Remove duplicate checks

Status Fixed in `Version 2`

Introduced by `Version 1`

Description There exist several duplicate checks which can be removed. For example, in the `createOffer` function of the `PromBundleMarketplace` contract, the check for `_payToken` in line 25 is unnecessary as it has been checked in the `validArgs` modifier.

[1]Fixed by removing the function

```
16  function createOffer(
17    bytes32 _bundleID,
18    address _payToken,
19    uint256 _price,
20    uint256 _deadline
21  ) external validArgs(_price, _payToken) {
22    require((_payToken) != address(0), "only erc20 supported for offers");
23    require(owners[_bundleID] != address(0), "invalid id");
24    require(
25      (IPromAddressRegistry(addressRegistry).isTokenEligible(_payToken)),
26      "invalid pay token"
27    );
28    require(_deadline > block.timestamp, "invalid expiration");
29
30    Offer memory offer = offers[_bundleID][msg.sender];
31    require(offer.deadline <= block.timestamp, "offer exists");
32
33    offers[_bundleID][msg.sender] = Offer(_payToken, _price, _deadline);
34
35    emit OfferCreated(
36      msg.sender,
37      _bundleID,
38      address(_payToken),
39      _price,
40      _deadline
41    );
42  }
```

Listing 2.14: BundleMarketplaceOffers.sol

Furthermore, in the `_listItem` internal function of the `BundleMarketplaceUtils` contract, the NFT addresses are iterated and checked against the address registry to ensure the trading of this NFT is enabled. Note that this function is only invoked by the public `listItem` function, which already implements such a check.

```
82  function _listItem(
83      bytes32 _bundleID,
84      address[] calldata _nftAddresses,
85      uint256[] calldata _tokenIds,
86      uint256[] calldata _quantities
87  ) internal {
88      for (uint256 i; i < _nftAddresses.length; i++) {
89          address _nft = _nftAddresses[i];
90
91          require(
92          addressRegistry.isTradeCollectionEnabled(_nft),
93          "collection not enabled"
94          );
```

Listing 2.15: BundleMarketplaceUtils.sol

```
29  function listItem(
30    bytes32 _bundleID,
```

```
31      address[] calldata _nftAddresses,
32      uint256[] calldata _tokenIds,
33      uint256[] calldata _quantities,
34      address _payToken,
35      uint256 _price,
36      uint256 _startingTime,
37      uint256 _endTime
38    ) external validArgs(_price, _payToken) {
39      bundleIds[_bundleID] = _bundleID;
40      require(
41        _nftAddresses.length == _tokenIds.length &&
42          _tokenIds.length == _quantities.length,
43        "invalid data"
44      );
45
46      for (uint256 i = 0; i < _nftAddresses.length; i++) {
47        require(
48          addressRegistry.isTradeCollectionEnabled(_nftAddresses[i]),
49          "collection not enabled"
50        );
51      }
```

Listing 2.16: PromBundleMarketplace.sol

Impact N/A

Suggestion Remove those duplicate checks.

2.3.3 Fix wrong owner address passed to the `isApprovedForAll` function

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the `BundleMarketplaceUtils` contract, the `_check721Owning` function has a check to verify if the owner has approved the NFT to the market contract by invoking the `isApprovedForAll` function. However, `msg.sender`, rather than `_owner`, is used as the first parameter to invoke the function. Note that the `_check1155Owning` function also has the same problem.

```
54    function _check721Owning(
55        address _nft,
56        uint256 _tokenId,
57        address _owner
58      ) internal view {
59        require(IERC721(_nft).ownerOf(_tokenId) == _owner, "not owning item");
60        require(
61          IERC721(_nft).isApprovedForAll(msg.sender, address(this)),
62          "item not approved"
63        );
64      }
65
66      function _check1155Owning(
67        address _nft,
68        uint256 _tokenId,
```

```
69        uint256 _quantity,
70        address _owner
71      ) internal view {
72        require(
73          IERC1155(_nft).balanceOf(_owner, _tokenId) >= _quantity,
74          "not owning item"
75        );
76        require(
77          IERC1155(_nft).isApprovedForAll(msg.sender, address(this)),
78          "item not approved"
79        );
80      }
```

Listing 2.17: BundleMarketplaceUtils.sol

Impact N/A

Suggestion Fix the address.

2.3.4 Fix improper parameters passed to the `_checkOwning` function

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the `_checkOwning` function of the `BundleMarketplaceUtils` contract, the quantity to check is passed in as the array parameter with an index, i.e., `_quantities[i]`. It should be directly passed in as a single parameter.

```
104   function _checkOwning(
105       uint256[] calldata _quantities,
106       uint256 i,
107       address _nft,
108       uint256 _tokenId
109   ) internal view {
110       if (_supportsInterface(_nft, INTERFACE_ID_ERC721)) {
111           _check721Owning(_nft, _tokenId, msg.sender);
112       } else if (_supportsInterface(_nft, INTERFACE_ID_ERC1155)) {
113           _check1155Owning(_nft, _tokenId, _quantities[i], msg.sender);
114       } else {
115           revert("invalid nft address");
116       }
117   }
```

Listing 2.18: BundleMarketplaceUtils.sol

Impact N/A

Suggestion Revise the code.

2.3.5 Revise the royalty fee collection logic

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the `registerCollectionRoyalty` function of the `TradeMarketplaceGuard` contract, there are two parameters related to the royalty fee charging for each NFT address, i.e., `_royalty` (the royalty fee ratio) and `_feeRecipient` (the fee recipient). However, in actual implementation, the royalty fees are all sent to `tradeMarketplaceFeeReceiver` obtained from the `addressRegistry`, while `_feeRecipient` is not used.

```
140    function registerCollectionRoyalty(
141        address _nftAddress,
142        uint16 _royalty,
143        address _feeRecipient
144    ) external onlyOwner {
145        require(_royalty <= 10000, "invalid royalty");
146        require(_feeRecipient != address(0), "invalid fee recipient address");
147
148        collectionRoyalties[_nftAddress] = CollectionRoyalty(
149            _royalty,
150            _feeRecipient
151        );
152    }
```

Listing 2.19: TradeMarketplaceGuard.sol

```
 99    function _handlePayment(
100        address _nftAddress,
101        address _paymentToken,
102        address _from, // msg.sender for buy, offer create for accepting offers
103        address _receiver,
104        uint256 _price,
105        uint256 _royaltyFee,
106        uint256 _totalFee
107    ) internal {
108        _transfer(
109            _from,
110            addressRegistry.tradeMarketplaceFeeReceiver(),
111            _totalFee,
112            _paymentToken
113        );
114
115        _transfer(_from, _receiver, _price - _totalFee, _paymentToken);
116        emit RoyaltyPayed(_nftAddress, _royaltyFee);
117    }
```

Listing 2.20: TradeMarketplaceUtils.sol

Impact N/A

Suggestion Revise the code.

2.3.6 Fix the incorrect reference of the interface name

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the `buyItem` function of `PromBundleMarketPlace` contract, the `validateItemSold` function (invoked in lines 136 and 137) is implemented in the `TradeMarketplaceValidator` contract, however, the interface is referred to as `IPromLendingMarketplace`.

```
128    for (uint256 i; i < listing.nfts.length; i++) {
129        _transferNft(
130        listing.nfts[i],
131        owner,
132        msg.sender,
133        listing.tokenIds[i],
134        listing.quantities[i]
135        );
136        IPromLendingMarketplace(addressRegistry.tradeMarketplace())
137            .validateItemSold(listing.nfts[i], listing.tokenIds[i], owner);
138    }
```

Listing 2.21: PromBundleMarketplace.sol

Impact N/A

Suggestion Fix the incorrect interface name.

2.3.7 Check the quantities of the tokens to be listed

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The `listItem` functions of the `PromBundleMarketplace` contract and the `TradeMarketplaceCore` contract do not have any checks on the quantities of tokens. As a result, the amount of the ERC1155 tokens listed by the users can be zero.

```
29    function listItem(
30        bytes32 _bundleID,
31        address[] calldata _nftAddresses,
32        uint256[] calldata _tokenIds,
33        uint256[] calldata _quantities,
34        address _payToken,
35        uint256 _price,
36        uint256 _startingTime,
37        uint256 _endTime
38    ) external validArgs(_price, _payToken) {
39        bundleIds[_bundleID] = _bundleID;
40        require(
41        _nftAddresses.length == _tokenIds.length &&
42            _tokenIds.length == _quantities.length,
43        "invalid data"
44        );
45
46        for (uint256 i = 0; i < _nftAddresses.length; i++) {
47        require(
48            addressRegistry.isTradeCollectionEnabled(_nftAddresses[i]),
49            "collection not enabled"
50        );
```

```
51        }
52
53        require(owners[_bundleID] == address(0), "already listed");
54
55        _listItem(_bundleID, _nftAddresses, _tokenIds, _quantities);
56        listings[msg.sender][_bundleID] = Listing({
57        nfts: _nftAddresses,
58        tokenIds: _tokenIds,
59        quantities: _quantities,
60        payToken: _payToken,
61        price: _price,
62        startingTime: _startingTime,
63        endTime: _endTime,
64        nonce: block.number
65        });
66
67        owners[_bundleID] = msg.sender;
68
69        emit ItemListed(
70        msg.sender,
71        _bundleID,
72        _payToken,
73        _price,
74        _startingTime,
75        _endTime
76        );
77    }
```

Listing 2.22: PromBundleMarketplace.sol

```
21    function listItem(
22        address _nftAddress,
23        uint256 _tokenId,
24        uint256 _quantity,
25        address _payToken,
26        uint256 _pricePerItem,
27        uint256 _startingTime,
28        uint256 _endTime
29      ) public onlyAssetOwner(_nftAddress, _tokenId, _quantity) {
30        _checkListing(_nftAddress, _tokenId, msg.sender, _payToken);
31
32        listings[_nftAddress][_tokenId][msg.sender] = Listing({
33          quantity: _quantity,
34          payToken: _payToken,
35          pricePerItem: _pricePerItem,
36          startingTime: _startingTime,
37          endTime: _endTime,
38          nonce: block.number
39        });
40        emit ItemListed(
41          msg.sender,
42          _nftAddress,
43          _tokenId,
```

```
44          _quantity,
45          _payToken,
46          _pricePerItem,
47          _startingTime,
48          _endTime
49        );
50      }
```

Listing 2.23: TradeMarketplaceCore.sol

Impact N/A

Suggestion Check the quantities of the tokens to be listed.

2.3.8 Implement a fine-grained approval for ERC-721 tokens

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the `onlyAssetOwner` modifier in the `TradeMarketplaceGuard` contract, there is a requirement enforced by invoking the `isApprovedForAll` function, which means for a given NFT, the listing creator (i.e., the seller) must call the `setApprovalForAll` function to grant the approval for all tokens to the marketplace contracts. However, only the listed tokens are required to grant the approval. Note that such a fine-grained operation can be achieved through the ERC-721 standard, which allows granting the approval for a single token ID.

```
20    modifier onlyAssetOwner(
21        address _nftAddress,
22        uint256 _tokenId,
23        uint256 _quantity
24    ) {
25        if (IERC165(_nftAddress).supportsInterface(INTERFACE_ID_ERC721)) {
26            IERC721 nft = IERC721(_nftAddress);
27            require(nft.ownerOf(_tokenId) == msg.sender, "not owning item");
28            require(
29                nft.isApprovedForAll(msg.sender, address(this)),
30                "item not approved"
31            );
32        } else if (IERC165(_nftAddress).supportsInterface(INTERFACE_ID_ERC1155)) {
33            IERC1155 nft = IERC1155(_nftAddress);
34            require(
35                nft.balanceOf(msg.sender, _tokenId) >= _quantity,
36                "must hold enough nfts"
37            );
38            require(
39                nft.isApprovedForAll(msg.sender, address(this)),
40                "item not approved"
41            );
42        } else {
43            revert("invalid nft address");
44        }
45        _;
46    }
```

Listing 2.24: TradeMarketplaceGuard.sol

Besides, the `_check721Owning` function of the `BundleMarketplaceUtils` contract has the similar issue.

```
54    function _check721Owning(
55        address _nft,
56        uint256 _tokenId,
57        address _owner
58    ) internal view {
59        require(IERC721(_nft).ownerOf(_tokenId) == _owner, "not owning item");
60        require(
61            IERC721(_nft).isApprovedForAll(msg.sender, address(this)),
62            "item not approved"
63        );
64    }
```

Listing 2.25: BundleMarketplaceUtils.sol

Impact N/A

Suggestion Revise the code to implement a fine-grained approval for ERC-721 tokens.

2.3.9 Revise code logic of the `acceptOffer` function for ERC-1155 tokens

Status Acknowledged

Introduced by `Version 1`

Description In the `acceptOffer` function of the `TradeMarketplaceOffers` contract (see Listing 2.12), once the offer is accepted, the listing slot for the NFT would be deleted. However, the NFT can be an ERC-1155 token and the listed quantity can be larger than the offered quantity. As such, deleting the listing slot actually cancels the listing of the remaining tokens.

Impact N/A

Suggestion Revise the code logic.