Security Audit Report for Paras Marketplace Contract

Date: September 24, 2022

Version: 1.0

Contact: contact@blocksec.com

BlockSec Audit Team

Digitally signed by BlockSec Audit Team
Date: 2022.10.05 14:18:59 +08'00'

Contents

Report Manifest

Item	Description
Client	Paras
Target	Paras Marketplace Contract

Version History

Version	Date	Description
1.0	September 24, 2022	First Release

About BlockSec The 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	Rust
Approach	Semi-automatic and manual verification

The repository that has been audited includes the **Paras Marketplace** contract [1].

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 during the audit are shown in the following. Our audit report is responsible for the only initial version (`Version 1`), as well as new codes (in the following versions) to fix issues in the audit report.

Project		Commit SHA
Paras Marketplace Contract	`Version 1`	`faae37b43eb6b98d6265e76bf3611179060a5e18`
	`Version 2`	`a9162f391440019dd6030e1fc97def2af543e861`

Note that, we did **NOT** audit all the modules in the repository. The modules covered by this audit report include **paras-marketplace-contract/src** folder contract only. Specifically, the file covered in this audit include:

- external.rs
- lib.rs
- nft_callbacks.rs

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/ParasHQ/paras-marketplace-contract

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 **eight** recommendations and **one** note.

- High Risk: 0
- Medium Risk: 3
- Low Risk: 3
- Recommendations: 8
- Notes: 1

ID	Severity	Description	Category	Status
1	Low	Denial of User's Legitimate Request	Software Security	Fixed
2	Low	Improper State Rollback	Software Security	Confirmed
3	Low	Improper Market Type Assertion	Software Security	Fixed
4	Medium	Extra Attached NEARs May Be Locked	DeFi Security	Fixed
5	Medium	Potential Lost of Users' Assets Due to Improper Treasury Fee	DeFi Security	Fixed
6	Medium	Failure of NEAR Transfer without Enough Balance	Defi Security	Fixed
7	-	Potential Centralization Problem	Recommendation	Confirmed
8	-	Potential Unsupported NFT Contracts Problem	Recommendation	Confirmed
9	-	Lack of assert_one_yocto() in Privileged Functions	Recommendation	Confirmed
10	-	Precision Loss	Recommendation	Fixed
11	-	Code Optimization (I)	Recommendation	Fixed
12	-	Code Optimization (II)	Recommendation	Fixed
13	-	Redundant Code	Recommendation	Fixed
14	-	Inconsistent Function Prototype Definitions	Recommendation	Fixed
15	-	Auctions Can Be Canceled Arbitrarily by the Seller	Note	Confirmed

The details are provided in the following sections.

2.1 Software Security

2.1.1 Denial of User's Legitimate Request

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description When adding the `market_data`, if the optional flag `is_auction` is set as `false` and the argument `ended_at` is not provided, the assertion in line 1987 will throw into a `panic`. However, according to the current implementation, the `sale` market does not need a start time or an end time.

```
1944    fn internal_add_market_data(
1945        &mut self,
1946        owner_id: AccountId,
1947        approval_id: u64,
```

```
1948        nft_contract_id: AccountId,
1949        token_id: TokenId,
1950        ft_token_id: AccountId,
1951        price: U128,
1952        mut started_at: Option<U64>,
1953        ended_at: Option<U64>,
1954        end_price: Option<U128>,
1955        is_auction: Option<bool>,
1956    ) {
1957        let contract_and_token_id = format!("{}{}{}", nft_contract_id, DELIMETER, token_id);
1958
1959        let bids: Option<Bids> = match is_auction {
1960            Some(u) => {
1961                if u {
1962                    Some(Vec::new())
1963                } else {
1964                    None
1965                }
1966            }
1967            None => None,
1968        };
1969
1970        let current_time: u64 = env::block_timestamp();
1971
1972        if started_at.is_some() {
1973            assert!(started_at.unwrap().0 >= current_time);
1974
1975            if ended_at.is_some() {
1976                assert!(started_at.unwrap().0 < ended_at.unwrap().0);
1977            }
1978        }
1979
1980        if let Some(is_auction) = is_auction {
1981            if is_auction == true {
1982                if started_at.is_none() {
1983                    started_at = Some(U64(current_time));
1984                }
1985            }
1986
1987            assert!(ended_at.is_some(), "Paras: Ended at is none")
1988        }
```

Listing 2.1: paras-marketplace-contract/src/lib.rs

Impact User's legitimate request can be denied unexpectedly.

Suggestion It's suggested to check whether the `end_at` timestamp is provided only when `is_auction` is `true`.

2.1.2 Improper State Rollback

Severity Low

Status Confirmed

Introduced by Version 1

Description The deleted `market_data` (line 467) cannot be recovered in the callback function `resolve_purchase()` if the `promise_result` of cross-contract invocation `nft_transfer_payout()` is checked as failed (lines 533-550).

```
459   fn internal_process_purchase(
460       &mut self,
461       nft_contract_id: AccountId,
462       token_id: TokenId,
463       buyer_id: AccountId,
464       price: u128,
465   ) -> Promise {
466       let market_data = self
467           .internal_delete_market_data(&nft_contract_id, &token_id)
468           .expect("Paras: Sale does not exist");
469
470       ext_contract::nft_transfer_payout(
471           buyer_id.clone(),
472           token_id,
473           Some(market_data.approval_id),
474           Some(price.into()),
475           Some(50u32), // max length payout
476           nft_contract_id,
477           1,
478           GAS_FOR_NFT_TRANSFER,
479       )
480       .then(ext_self::resolve_purchase(
481           buyer_id,
482           market_data,
483           price.into(),
484           env::current_account_id(),
485           NO_DEPOSIT,
486           GAS_FOR_ROYALTIES,
487       ))
488   }
```

Listing 2.2: paras-marketplace-contract/src/lib.rs

```
532           // leave function and return all FTs in ft_resolve_transfer
533           if !is_promise_success() {
534               if market_data.ft_token_id == near_account() {
535                   Promise::new(buyer_id.clone()).transfer(u128::from(price));
536               }
537               env::log_str(
538                   &json!({
539                   "type": "resolve_purchase_fail",
540                   "params": {
541                       "owner_id": market_data.owner_id,
542                       "nft_contract_id": market_data.nft_contract_id,
543                       "token_id": market_data.token_id,
544                       "ft_token_id": market_data.ft_token_id,
545                       "price": price,
546                       "buyer_id": buyer_id,
```

```
547                 }
548             })
549                     .to_string(),
550             );
551         } else if market_data.ft_token_id == near_account() {
```

Listing 2.3: Function `resolve_purchase()` in paras-marketplace-contract/src/lib.rs

Impact The `market_data` will be deleted unexpectedly due to a failed purchase.

Suggestion Recover the deleted `market_data` in the callback function `resolve_purchase()` if the `promise_result` of cross-contract invocation `nft_transfer_payout` is checked as failed (lines 533-550).

Feedback from the Project This is by design, usually when promise is not success it is caused by Unauthorized or faulty `approval_id` which is the intended solution is to delete the `market_data`.

2.1.3 Improper Market Type Assertion

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The optional flag `is_auction` can be set to `false` when adding the `sale` market via the function `internal_add_market_data()`. However, in this case, buyers are not able to buy the NFT due to the assertion in function `buy()` (line 436).

```
388     #[payable]
389     pub fn buy(
390         &mut self,
391         nft_contract_id: AccountId,
392         token_id: TokenId,
393         ft_token_id: Option<AccountId>,
394         price: Option<U128>,
395     ) {
396         let contract_and_token_id = format!("{}{}{}", &nft_contract_id, DELIMETER, token_id);
397         let market_data: Option<MarketData> =
398             if let Some(market_data) = self.old_market.get(&contract_and_token_id) {
399                 Some(MarketData {
400                     owner_id: market_data.owner_id,
401                     approval_id: market_data.approval_id,
402                     nft_contract_id: market_data.nft_contract_id,
403                     token_id: market_data.token_id,
404                     ft_token_id: market_data.ft_token_id,
405                     price: market_data.price,
406                     bids: None,
407                     started_at: None,
408                     ended_at: None,
409                     end_price: None,
410                     accept_nft_contract_id: None,
411                     accept_token_id: None,
412                     is_auction: None,
413                 })
414             } else if let Some(market_data) = self.market.get(&contract_and_token_id) {
```

```
415                 Some(market_data)
416             } else {
417                 env::panic_str(&"Paras: Market data does not exist");
418             };
419
420         let market_data: MarketData = market_data.expect("Paras: Market data does not exist");
421
422         let buyer_id = env::predecessor_account_id();
423
424         assert_ne!(
425             buyer_id, market_data.owner_id,
426             "Paras: Cannot buy your own sale"
427         );
428
429         // only NEAR supported for now
430         assert_eq!(
431             market_data.ft_token_id.to_string(),
432             NEAR,
433             "Paras: NEAR support only"
434         );
435
436         assert!(market_data.is_auction.is_none(), "Paras: the NFT is on auction");
437
438         if ft_token_id.is_some() {
439             assert_eq!(
440                 ft_token_id.unwrap().to_string(),
441                 market_data.ft_token_id.to_string()
442             )
443         }
444         if price.is_some() {
445             assert_eq!(price.unwrap().0, market_data.price);
446         }
447
448         let price = market_data.price;
449
450         assert!(
451             env::attached_deposit() >= price,
452             "Paras: Attached deposit is less than price {}",
453             price
454         );
455
456         self.internal_process_purchase(nft_contract_id.into(), token_id, buyer_id, price);
457     }
```

Listing 2.4: paras-marketplace-contract/src/lib.rs

Impact An NFT in `sale` may not be able to be purchased by buyers.

Suggestion It's suggested to check the `market_data.is_auction` is not `true` in function `buy()` instead of the assertion.

2.2 DeFi Security

2.2.1 Extra Attached NEARs May Be Locked

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description If the user purchases an NFT in sale via the function `buy()` with more NEARs attached than the price of this sale, the extra NEARs will not be refunded and will be locked in the contract permanently unless the owner helps to transfer them out.

```
388    #[payable]
389    pub fn buy(
390        &mut self,
391        nft_contract_id: AccountId,
392        token_id: TokenId,
393        ft_token_id: Option<AccountId>,
394        price: Option<U128>,
395    ) {
396        let contract_and_token_id = format!("{}{}{}", &nft_contract_id, DELIMETER, token_id);
397        let market_data: Option<MarketData> =
398            if let Some(market_data) = self.old_market.get(&contract_and_token_id) {
399                Some(MarketData {
400                    owner_id: market_data.owner_id,
401                    approval_id: market_data.approval_id,
402                    nft_contract_id: market_data.nft_contract_id,
403                    token_id: market_data.token_id,
404                    ft_token_id: market_data.ft_token_id,
405                    price: market_data.price,
406                    bids: None,
407                    started_at: None,
408                    ended_at: None,
409                    end_price: None,
410                    accept_nft_contract_id: None,
411                    accept_token_id: None,
412                    is_auction: None,
413                })
414            } else if let Some(market_data) = self.market.get(&contract_and_token_id) {
415                Some(market_data)
416            } else {
417                env::panic_str(&"Paras: Market data does not exist");
418            };
419
420        let market_data: MarketData = market_data.expect("Paras: Market data does not exist");
421
422        let buyer_id = env::predecessor_account_id();
423
424        assert_ne!(
425            buyer_id, market_data.owner_id,
426            "Paras: Cannot buy your own sale"
427        );
```

```
428
429         // only NEAR supported for now
430         assert_eq!(
431             market_data.ft_token_id.to_string(),
432             NEAR,
433             "Paras: NEAR support only"
434         );
435
436         assert!(market_data.is_auction.is_none(), "Paras: the NFT is on auction");
437
438         if ft_token_id.is_some() {
439             assert_eq!(
440                 ft_token_id.unwrap().to_string(),
441                 market_data.ft_token_id.to_string()
442             )
443         }
444         if price.is_some() {
445             assert_eq!(price.unwrap().0, market_data.price);
446         }
447
448         let price = market_data.price;
449
450         assert!(
451             env::attached_deposit() >= price,
452             "Paras: Attached deposit is less than price {}",
453             price
454         );
455
456         self.internal_process_purchase(nft_contract_id.into(), token_id, buyer_id, price);
457     }
```

Listing 2.5: paras-marketplace-contract/src/lib.rs

Impact Buyer's extra NEARs may be locked by the contract.

Suggestion Refund the extra attached NEARs.

2.2.2 Potential Lost of Users' Assets Due to Improper Treasury Fee

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The treasury fee, payouts for royalties, and the payout for the `market_data.owner_id` are all calculated based on the price of the NFT and meet the equation below:

$$market\_data.price = sum(payouts\ for\ royalties) + payout\ for\ market\_data.owner\_id$$

However, the treasury fee is actually deducted from the payment which is intended to be transferred to the `market_data.owner_id`. If the treasury fee is larger than the payout for the `market_data.owner_id`, there will be an underflow in callback function `resolve_purchase()` (line 587) or `resolve_offer()` (line 1099), resulting in panic.

```
578    // Payout (transfer to royalties and seller)
579    if market_data.ft_token_id == near_account() {
580        // 5% fee for treasury
581        let treasury_fee = price.0 * self.calculate_market_data_transaction_fee(&market_data.
               nft_contract_id, &market_data.token_id) / 10_000u128;
582        let contract_and_token_id = format!("{}{}{}", &market_data.nft_contract_id, DELIMETER, &
               market_data.token_id);
583        self.market_data_transaction_fee.transaction_fee.remove(&contract_and_token_id);
584
585        for (receiver_id, amount) in payout {
586            if receiver_id == market_data.owner_id {
587                Promise::new(receiver_id).transfer(amount.0 - treasury_fee);
588                if treasury_fee != 0 {
589                    Promise::new(self.treasury_id.clone()).transfer(treasury_fee);
590                }
591            } else {
592                Promise::new(receiver_id).transfer(amount.0);
593            }
594        }
```

Listing 2.6: Function `resolve_purchase()` in paras-marketplace-contract/src/lib.rs

```
1091    // Payout (transfer to royalties and seller)
1092    if offer_data.ft_token_id == near_account() {
1093        // 5% fee for treasury
1094        let treasury_fee =
1095            offer_data.price as u128 * self.calculate_current_transaction_fee() / 10_000u128;
1096
1097        for (receiver_id, amount) in payout {
1098            if receiver_id == seller_id {
1099                Promise::new(receiver_id).transfer(amount.0 - treasury_fee);
1100                if treasury_fee != 0 {
1101                    Promise::new(self.treasury_id.clone()).transfer(treasury_fee);
1102                }
1103            } else {
1104                Promise::new(receiver_id).transfer(amount.0);
1105            }
1106        }
```

Listing 2.7: Function `resolve_offer()` in paras-marketplace-contract/src/lib.rs

Impact Users' assets will be lost due to the panic of the callback functions.

Suggestion Avoid underflow when the treasury fee is larger than the payout to `market_data.owner_id`.

2.2.3 Failure of NEAR Transfer without Enough Balance

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description If the balance of NEARs in this contract is not enough, the transfer of NEAR executed in the next block may fail without rolling back the contract state, resulting in the loss of users' assets. Line 710 of function `add_offer()` shows an example.

```
666    #[payable]
667    pub fn add_offer(
668        &mut self,
669        nft_contract_id: AccountId,
670        token_id: Option<TokenId>,
671        token_series_id: Option<String>,
672        ft_token_id: AccountId,
673        price: U128,
674    ) {
675        let token = if token_id.is_some() {
676            token_id.as_ref().unwrap().to_string()
677        } else {
678            assert!(
679                self.paras_nft_contracts.contains(&nft_contract_id),
680                "Paras: offer series for Paras NFT only"
681            );
682            token_series_id.as_ref().unwrap().to_string()
683        };
684
685        assert_eq!(
686            env::attached_deposit(),
687            price.0,
688            "Paras: Attached deposit != price"
689        );
690
691        assert_eq!(
692            ft_token_id.to_string(),
693            "near",
694            "Paras: Only NEAR is supported"
695        );
696
697        assert!(
698            self.approved_nft_contract_ids.contains(&nft_contract_id),
699            "Paras: nft_contract_id is not approved"
700        );
701
702        let buyer_id = env::predecessor_account_id();
703        let offer_data = self.internal_delete_offer(
704            nft_contract_id.clone().into(),
705            buyer_id.clone(),
706            token.clone(),
707        );
708
709        if offer_data.is_some() {
710            Promise::new(buyer_id.clone()).transfer(offer_data.unwrap().price);
711        }
```

Listing 2.8: paras-marketplace-contract/src/lib.rs

Suggestion It is suggested to check the balance of NEARs in this contract `(env::account_balance)` before NEAR transfer.

2.3 Additional Recommendation

2.3.1 Potential Centralization Problem

Status Confirmed

Introduced by `Version 1`

Description The privileged account `Contract.owner_id` has the ability to configure some of the system parameters (e.g., `Contract.transaction_fee` and `Contract.treasury_id)`, remove the `market_data`, and change whitelist (e.g., `Contract.approved_nft_contract_ids` and `Contract.paras_nft_contracts)`. Additionally, the person with the full access key of this contract could transfer assets out (e.g., NEARs) and upgrade the contract directly.

Suggestion It's suggested to remove the full access key of the contract from the blockchain (via `DeleteKey` transaction) and implement the privileged upgrade function. Besides, a decentralization design is also recommended to be introduced in the contract. The privileged roles are suggested to be transferred to a multi-signature account or DAO.

Feedback from the Project Will move to multi-sig.

2.3.2 Potential Unsupported NFT Contracts Problem

Status Confirmed

Introduced by `Version 1`

Description Do not add contracts that do not implement NEP-199 (Non-Fungible Token Royalties and Payouts Extension) to the whitelist. Otherwise, the progress of purchase may always fail due to the default cross-contract invocation `nft_transfer_payout()` in function `internal_process_purchase()` (line 470).

```
459   fn internal_process_purchase(
460       &mut self,
461       nft_contract_id: AccountId,
462       token_id: TokenId,
463       buyer_id: AccountId,
464       price: u128,
465   ) -> Promise {
466       let market_data = self
467           .internal_delete_market_data(&nft_contract_id, &token_id)
468           .expect("Paras: Sale does not exist");
469
470       ext_contract::nft_transfer_payout(
471           buyer_id.clone(),
472           token_id,
473           Some(market_data.approval_id),
474           Some(price.into()),
475           Some(50u32), // max length payout
476           nft_contract_id,
477           1,
```

```
478             GAS_FOR_NFT_TRANSFER,
479         )
480         .then(ext_self::resolve_purchase(
481             buyer_id,
482             market_data,
483             price.into(),
484             env::current_account_id(),
485             NO_DEPOSIT,
486             GAS_FOR_ROYALTIES,
487         ))
488     }
```

Listing 2.9: paras-marketplace-contract/src/lib.rs

Suggestion Only NFT contracts that implement NEP-199 (Non-Fungible Token Royalties and Payouts Extension) should be added to the whitelist.

Feedback from the Project We checked this from our backend.

2.3.3 Lack of assert_one_yocto() in Privileged Functions

Status Confirmed

Introduced by Version 1

Description Functions `add_approved_nft_contract_ids()`, `remove_approved_nft_contract_ids()`, `add_approved_paras_nft_contract_ids()`, and `add_approved_ft_token_ids()` are sensitive operations, and function `assert_one_yocto()` should be added in them to enable the 2FA.

```
353     #[payable]
354     pub fn transfer_ownership(&mut self, owner_id: AccountId) {
355         assert_one_yocto();
356         self.assert_owner();
357         self.owner_id = owner_id;
358     }
359
360     // Approved contracts
361     #[payable]
362     pub fn add_approved_nft_contract_ids(&mut self, nft_contract_ids: Vec<AccountId>) {
363         self.assert_owner();
364         add_accounts(Some(nft_contract_ids), &mut self.approved_nft_contract_ids);
365     }
366
367     #[payable]
368     pub fn remove_approved_nft_contract_ids(&mut self, nft_contract_ids: Vec<AccountId>) {
369         self.assert_owner();
370         remove_accounts(Some(nft_contract_ids), &mut self.approved_nft_contract_ids);
371     }
372
373     // Approved paras contracts
374     #[payable]
375     pub fn add_approved_paras_nft_contract_ids(&mut self, nft_contract_ids: Vec<AccountId>) {
376         self.assert_owner();
377         add_accounts(Some(nft_contract_ids), &mut self.paras_nft_contracts);
```

```
378     }
379
380     #[payable]
381     pub fn add_approved_ft_token_ids(&mut self, ft_token_ids: Vec<AccountId>) {
382         self.assert_owner();
383         add_accounts(Some(ft_token_ids), &mut self.approved_ft_token_ids);
384     }
```

Listing 2.10: paras-marketplace-contract/src/lib.rs

Suggestion Function `assert_one_yocto()` is suggested to be added to enable 2FA.

Feedback from the Project This is a tricky one, we run these functions using a function call access key using our backend worker, which can't attach deposit.

2.3.4 Precision Loss

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In line 1706 of function `add_bid()`, a division is performed before multiplication when calculating the expected lower bound of the user's bid, which may result in precision loss.

```
1705     assert!(
1706         amount.0 >= current_bid.price.0 + (current_bid.price.0 / 100 * 5),
1707         "Paras: Can't pay less than or equal to current bid price + 5% : {:?}",
1708         current_bid.price.0 + (current_bid.price.0 / 100 * 5)
1709       );
```

Listing 2.11: Function `add_bid()` in paras-marketplace-contract/src/lib.rs

Suggestion Modify this calculation to perform multiplication before division.

2.3.5 Code Optimization (I)

Status Fixed in `Version 2`

Introduced by `Version 1`

Description According to the current implementation of contract, the parameters `started_at` and `ended_at` are only used for `auction` market type, so there is no need to check them for `sale` market (lines 2022-2028).

```
1994     fn internal_add_market_data(
1995         &mut self,
1996         owner_id: AccountId,
1997         approval_id: u64,
1998         nft_contract_id: AccountId,
1999         token_id: TokenId,
2000         ft_token_id: AccountId,
2001         price: U128,
2002         mut started_at: Option<U64>,
2003         ended_at: Option<U64>,
2004         end_price: Option<U128>,
2005         is_auction: Option<bool>,
```

```
2006    ) {
2007        let contract_and_token_id = format!("{}{}{}", nft_contract_id, DELIMETER, token_id);
2008
2009        let bids: Option<Bids> = match is_auction {
2010            Some(u) => {
2011                if u {
2012                    Some(Vec::new())
2013                } else {
2014                    None
2015                }
2016            }
2017            None => None,
2018        };
2019
2020        let current_time: u64 = env::block_timestamp();
2021
2022        if started_at.is_some() {
2023            assert!(started_at.unwrap().0 >= current_time);
2024
2025            if ended_at.is_some() {
2026                assert!(started_at.unwrap().0 < ended_at.unwrap().0);
2027            }
2028        }
2029
2030        if let Some(is_auction) = is_auction {
2031            if is_auction == true {
2032                if started_at.is_none() {
2033                    started_at = Some(U64(current_time));
2034                }
2035            }
2036
2037            assert!(ended_at.is_some(), "Paras: Ended at is none")
2038        }
```

Listing 2.12: paras-marketplace-contract/src/lib.rs

Suggestion Check the parameters `started_at` and `ended_at` only for the `auction` market in function `internal_add_market_data()`.

2.3.6 Code Optimization (II)

Status Fixed in `Version 2`

Introduced by `Version 1`

Description When a user bids for a particular auction, the `started_at` and `ended_at` timestamps will not be `None` as they are already set during the initialization via the function `internal_add_market_data()` (lines 1980-1987).

```
1634    // Auction bids
1635    #[payable]
1636    pub fn add_bid(
1637        &mut self,
1638        nft_contract_id: AccountId,
```

```
1639        ft_token_id: AccountId,
1640        token_id: TokenId,
1641        amount: U128,
1642    ) {
1643        let contract_and_token_id = format!("{}{}{}", &nft_contract_id, DELIMETER, token_id);
1644        let mut market_data = self
1645            .market
1646            .get(&contract_and_token_id)
1647            .expect("Paras: Token id does not exist");
1648
1649        assert_eq!(market_data.is_auction.unwrap(), true, "Paras: not auction");
1650
1651        let bidder_id = env::predecessor_account_id();
1652        let current_time = env::block_timestamp();
1653
1654        if market_data.started_at.is_some() {
1655            assert!(
1656                current_time >= market_data.started_at.unwrap(),
1657                "Paras: Sale has not started yet"
1658            );
1659        }
1660
1661        if market_data.ended_at.is_some() {
1662            assert!(
1663                current_time <= market_data.ended_at.unwrap(),
1664                "Paras: Sale has ended"
1665            );
1666        }
1667
1668        let remaining_time = market_data.ended_at.unwrap() - current_time;
1669        if remaining_time <= FIVE_MINUTES {
1670          let extended_ended_at = market_data.ended_at.unwrap() + FIVE_MINUTES;
1671          market_data.ended_at = Some(extended_ended_at);
1672
1673          env::log_str(
1674            &json!({
1675                "type": "extend_auction",
1676                "params": {
1677                    "nft_contract_id": nft_contract_id,
1678                    "token_id": token_id,
1679                    "ended_at": extended_ended_at,
1680                }
1681            })
1682            .to_string(),
1683          );
1684        }
```

Listing 2.13: paras-marketplace-contract/src/lib.rs

```
1944    fn internal_add_market_data(
1945        &mut self,
1946        owner_id: AccountId,
1947        approval_id: u64,
```

```
1948        nft_contract_id: AccountId,
1949        token_id: TokenId,
1950        ft_token_id: AccountId,
1951        price: U128,
1952        mut started_at: Option<U64>,
1953        ended_at: Option<U64>,
1954        end_price: Option<U128>,
1955        is_auction: Option<bool>,
1956    ) {
1957        let contract_and_token_id = format!("{}{}{}", nft_contract_id, DELIMETER, token_id);
1958
1959        let bids: Option<Bids> = match is_auction {
1960            Some(u) => {
1961                if u {
1962                    Some(Vec::new())
1963                } else {
1964                    None
1965                }
1966            }
1967            None => None,
1968        };
1969
1970        let current_time: u64 = env::block_timestamp();
1971
1972        if started_at.is_some() {
1973            assert!(started_at.unwrap().0 >= current_time);
1974
1975            if ended_at.is_some() {
1976                assert!(started_at.unwrap().0 < ended_at.unwrap().0);
1977            }
1978        }
1979
1980        if let Some(is_auction) = is_auction {
1981            if is_auction == true {
1982                if started_at.is_none() {
1983                    started_at = Some(U64(current_time));
1984                }
1985            }
1986
1987            assert!(ended_at.is_some(), "Paras: Ended at is none")
1988        }
```

Listing 2.14: paras-marketplace-contract/src/lib.rs

Suggestion There is no need to check the existences of `started_at` and `ended_at` timestamps in function `add_bid()` (line 1654 and line 1661).

2.3.7 Redundant Code

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The existence of the corresponding `trade_data` is already checked in function `delete_trade()` and there is no need to handle `trade_data` as `None` in its callee (i.e., `internal_delete_trade()`).

```
1242    #[payable]
1243    pub fn delete_trade(
1244        &mut self,
1245        nft_contract_id: AccountId,
1246        token_id: Option<TokenId>,
1247        token_series_id: Option<TokenSeriesId>,
1248        buyer_nft_contract_id: AccountId,
1249        buyer_token_id: TokenId,
1250    ) {
1251        assert_one_yocto();
1252        let token = if token_id.is_some() {
1253            token_id.as_ref().unwrap().to_string()
1254        } else {
1255            token_series_id.as_ref().unwrap().to_string()
1256        };
1257
1258        let buyer_id = env::predecessor_account_id();
1259        let buyer_contract_account_id_token_id =
1260            make_triple(&buyer_nft_contract_id, &buyer_id, &buyer_token_id);
1261        let contract_account_id_token_id = make_triple(&nft_contract_id, &buyer_id, &token);
1262
1263        let trade_list = self
1264            .trades
1265            .get(&buyer_contract_account_id_token_id)
1266            .expect("Paras: Trade list does not exist");
1267
1268        let trade_data = trade_list
1269            .trade_data
1270            .get(&contract_account_id_token_id)
1271            .expect("Paras: Trade data does not exist");
1272
1273        if token_id.is_some() {
1274            assert_eq!(trade_data.clone().token_id.unwrap(), token)
1275        } else {
1276            assert_eq!(trade_data.clone().token_series_id.unwrap(), token)
1277        }
1278
1279        self.internal_delete_trade(
1280            nft_contract_id.clone().into(),
1281            buyer_id.clone(),
1282            token.clone(),
1283            buyer_nft_contract_id.clone(),
1284            buyer_token_id.clone(),
1285        )
1286        .expect("Paras: Trade not found");
1287
1288        env::log_str(
1289            &json!({
1290                "type": "delete_trade",
1291                "params": {
```

```
                "nft_contract_id": nft_contract_id,
                "buyer_id": buyer_id,
                "token_id": token_id,
                "token_series_id": token_series_id,
                "buyer_nft_contract_id": buyer_nft_contract_id,
                "buyer_token_id": buyer_token_id
            }
        })
        .to_string(),
    );
}
```

Listing 2.15: paras-marketplace-contract/src/lib.rs

```
fn internal_delete_trade(
    &mut self,
    nft_contract_id: AccountId,
    buyer_id: AccountId,
    token_id: TokenId,
    buyer_nft_contract_id: AccountId,
    buyer_token_id: TokenId,
) -> Option<TradeData> {
    let buyer_contract_account_id_token_id =
        make_triple(&buyer_nft_contract_id, &buyer_id, &buyer_token_id);
    let contract_account_id_token_id = make_triple(&nft_contract_id, &buyer_id, &token_id);

    let mut trade_list = self
        .trades
        .get(&buyer_contract_account_id_token_id)
        .expect("Paras: Trade list does not exist");

    let trade_data = trade_list.trade_data.remove(&contract_account_id_token_id);

    self.trades
        .insert(&buyer_contract_account_id_token_id, &trade_list);

    match trade_data {
        Some(trade) => {
            let mut by_owner_id = self
                .by_owner_id
                .get(&buyer_id)
                .expect("Paras: no market data by account_id");
            by_owner_id.remove(&make_key_owner_by_id_trade(contract_account_id_token_id));
            if by_owner_id.is_empty() {
                self.by_owner_id.remove(&buyer_id);
            } else {
                self.by_owner_id.insert(&buyer_id, &by_owner_id);
            }
            return Some(trade);
        }
        None => {
            self.trades
                .remove(&buyer_contract_account_id_token_id)
```

```
1343                 .expect("Paras: Error delete trade list");
1344             return None;
1345         }
1346     };
1347 }
```

Listing 2.16: paras-marketplace-contract/src/lib.rs

Suggestion Remove the redundant code.

2.3.8 Inconsistent Function Prototype Definitions

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The function signature of `nft_transfer()` defined in contract `Paras_MarketPlace_Contract` does not match some of the existing NFT contracts (e.g., `Paras_NFT_Contract`) and does not support adding `memo`.

```
5#[ext_contract(ext_contract)]
6trait ExtContract {
7    fn nft_transfer_payout(
8        &mut self,
9        receiver_id: AccountId,
10       token_id: TokenId,
11       approval_id: Option<u64>,
12       balance: Option<U128>,
13       max_len_payout: Option<u32>,
14   );
15   fn nft_transfer(&mut self, receiver_id: AccountId, token_id: TokenId, approval_id: Option<u64
         >);
16}
```

Listing 2.17: paras-marketplace-contract/src/external.rs

```
853     #[payable]
854     pub fn nft_transfer(
855         &mut self,
856         receiver_id: ValidAccountId,
857         token_id: TokenId,
858         approval_id: Option<u64>,
859         memo: Option<String>,
860     ) {
861         let sender_id = env::predecessor_account_id();
862         let previous_owner_id = self.tokens.owner_by_id.get(&token_id).expect("Token not found");
863         let receiver_id_str = receiver_id.to_string();
864         self.tokens.nft_transfer(receiver_id, token_id.clone(), approval_id, memo.clone());
865 
866         let authorized_id : Option<AccountId> = if sender_id != previous_owner_id {
867             Some(sender_id)
868         } else {
869             None
870         };
```

Listing 2.18: paras-nft-contract/src/lib.rs

Suggestion Add the support if necessary.

2.4 Notes

2.4.1 Auctions Can Be Canceled Arbitrarily by the Seller

Status Confirmed

Introduced by Version 1

Description The auction `market_data` can be arbitrarily removed by corresponding seller via function `deleta_market_data()` (line 2181). For example, an auction can be canceled even if the auction has already ended but has not been accepted by the seller.

```
2133    #[payable]
2134    pub fn delete_market_data(&mut self, nft_contract_id: AccountId, token_id: TokenId) {
2135        let predecessor_account_id = env::predecessor_account_id();
2136        if predecessor_account_id != self.owner_id {
2137            assert_one_yocto();
2138        }
2139
2140        let contract_and_token_id = format!("{}{}{}", nft_contract_id, DELIMETER, token_id);
2141        let current_time: u64 = env::block_timestamp();
2142
2143        let market_data: Option<MarketData> =
2144            if let Some(market_data) = self.old_market.get(&contract_and_token_id) {
2145                Some(MarketData {
2146                    owner_id: market_data.owner_id,
2147                    approval_id: market_data.approval_id,
2148                    nft_contract_id: market_data.nft_contract_id,
2149                    token_id: market_data.token_id,
2150                    ft_token_id: market_data.ft_token_id,
2151                    price: market_data.price,
2152                    bids: None,
2153                    started_at: None,
2154                    ended_at: None,
2155                    end_price: None,
2156                    accept_nft_contract_id: None,
2157                    accept_token_id: None,
2158                    is_auction: None,
2159                })
2160            } else if let Some(market_data) = self.market.get(&contract_and_token_id) {
2161                Some(market_data)
2162            } else {
2163                None
2164            };
2165
2166        let market_data: MarketData = market_data.expect("Paras: Market data does not exist");
2167
```

```
2168        assert!(
2169            [market_data.owner_id.clone(), self.owner_id.clone()]
2170                .contains(&predecessor_account_id),
2171            "Paras: Seller or owner only"
2172        );
2173
2174        if market_data.is_auction.is_some() && predecessor_account_id == self.owner_id {
2175          assert!(
2176            current_time >= market_data.ended_at.unwrap(),
2177            "Paras: Auction has not ended yet"
2178          );
2179        }
2180
2181        self.internal_delete_market_data(&nft_contract_id, &token_id);
2182
2183        env::log_str(
2184            &json!({
2185                "type": "delete_market_data",
2186                "params": {
2187                    "owner_id": market_data.owner_id,
2188                    "nft_contract_id": nft_contract_id,
2189                    "token_id": token_id,
2190                }
2191            })
2192            .to_string(),
2193        );
2194    }
```

Listing 2.19: paras-marketplace-contract/src/lib.rs

```
2708    fn internal_delete_market_data(
2709        &mut self,
2710        nft_contract_id: &AccountId,
2711        token_id: &TokenId,
2712    ) -> Option<MarketData> {
2713        let contract_and_token_id = format!("{}{}{}", &nft_contract_id, DELIMETER, token_id);
2714
2715        let market_data: Option<MarketData> =
2716            if let Some(market_data) = self.old_market.get(&contract_and_token_id) {
2717                self.old_market.remove(&contract_and_token_id);
2718                Some(MarketData {
2719                    owner_id: market_data.owner_id,
2720                    approval_id: market_data.approval_id,
2721                    nft_contract_id: market_data.nft_contract_id,
2722                    token_id: market_data.token_id,
2723                    ft_token_id: market_data.ft_token_id,
2724                    price: market_data.price,
2725                    bids: None,
2726                    started_at: None,
2727                    ended_at: None,
2728                    end_price: None,
2729                    accept_nft_contract_id: None,
2730                    accept_token_id: None,
```

```
2731                    is_auction: None,
2732                })
2733            } else if let Some(market_data) = self.market.get(&contract_and_token_id) {
2734                self.market.remove(&contract_and_token_id);
2735
2736                if let Some(ref bids) = market_data.bids {
2737                    for bid in bids {
2738                        Promise::new(bid.bidder_id.clone()).transfer(bid.price.0);
2739                    }
2740                };
2741
2742                Some(market_data)
2743            } else {
2744                None
2745            };
2746
2747        market_data.map(|market_data| {
2748            let by_owner_id = self
2749                .by_owner_id
2750                .get(&market_data.owner_id);
2751            if let Some(mut by_owner_id) = by_owner_id {
2752                by_owner_id.remove(&contract_and_token_id);
2753                if by_owner_id.is_empty() {
2754                self.by_owner_id.remove(&market_data.owner_id);
2755                } else {
2756                self.by_owner_id.insert(&market_data.owner_id, &by_owner_id);
2757                }
2758            }
2759            market_data
2760        })
2761    }
```

Listing 2.20: paras-marketplace-contract/src/lib.rs

Feedback from the Project This is by design, even if we prevent this, the owner can still transfer or revoke the `approval_id` which invalidate the auction.