#HashEx
BLOCKCHAIN SECURITY

Paras Marketplace

smart contracts
final audit report

July 2022


hashex.org


contact@hashex.org

Contents

1. Disclaimer

This is a limited report on our findings based on our analysis, in accordance with good industry practice at the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the disclaimer below – please make sure to read it in full.

By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any and all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and HashEx and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers, and other representatives) (HashEx) owe no duty of care towards you or any other person, nor does HashEx make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties, or other terms of any kind except as set out in this disclaimer, and HashEx hereby excludes all representations, warranties, conditions, and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, HashEx hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against HashEx, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of the use of this report, and any reliance on this report. The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed. HashEx owns all copyright rights to the text, images, photographs, and other content provided in the following document. When using or sharing partly or in full, third parties must provide a direct link to the original document mentioning the author (hashex.org).

2. Overview

HashEx was commissioned to perform an audit of the Paras Marketplace smart contract. The audit was conducted between 30/06/2022 and 08/07/2022.

The purpose of this audit was to achieve the following:

- Identify potential security issues with smart contracts
- Formally check the logic behind given smart contracts.

Information in this report should be used for understanding the risk exposure of smart contracts, and as a guide to improving the security posture of smart contracts by remediating the issues that were identified.

The code is available at the GitHub repository @ParasHQ/paras-marketplace-contract after the commit 48161fb.

This smart contract is deployed in the NEAR blockchain at @marketplace.paras.near.

2.1 Summary

Project name	Paras Marketplace
URL	https://paras.id/
Platform	NEAR
Language	Rust

2.2 Contracts

Name	Address
lib.rs	`@marketplace.paras.near`
nft_callbacks.rs	`@marketplace.paras.near`

3. Found issues

High	2 (14%)
Medium	2 (14%)
Low	4 (29%)
Info	6 (43%)

C1. lib.rs

ID	Severity	Title	Status
C1-01	High	Locked funds	Open
C1-02	High	No check for promise results	Open
C1-03	Medium	Bad implementation of storage management	Open
C1-04	Medium	Not enough funds for a transfer	Open
C1-05	Low	Unlimited fee amount	Open
C1-06	Low	Treasury fee won't be transferred	Open
C1-07	Low	Division before multiplication	Open
C1-08	Low	Transaction fee view method	Open
C1-09	Info	Accept bids for ended auctions	Open
C1-10	Info	Doesn't check length of payout	Open

C1-11	Info	No checked math used	Open
C1-12	Info	Typos and comments	Open
C1-13	Info	No checks for promise results with transfer native tokens	Open
C1-14	Info	Remainder in NFT contracts	Open

4. Contracts

C1. lib.rs

Overview

This contract has the functionality of an NFT marketplace for Paras NFTs alongside any of the whitelisted NEP-171 NFT contracts. Documentation is available on the team website.

Issues

C1-01 Locked funds

High Open

In the function `buy()` if the buyer attaches a value more than the variable price then the remainder of the attached amount will be permanently locked in the contract balance.

```
#[payable]
pub fn buy(..., price: Option<U128>) {
  ...
  assert!(env::attached_deposit() >= price,
          "Paras: Attached deposit is less than price {}",
          price);
  ...
}
```

Recommendation

It is recommended to check the number of tokens attached, and refund the caller if there are more tokens attached than necessary.

C1-02 No check for promise results

High Open

In `resolve_purchase()` and `resolve_offer()` functions on L611, L1094 there is no guarantee that the variable amount is not less than the variable's `treasury_fee`. Because of that user may lose funds (the promise result isn't checked).

```
pub fn resolve_purchase(...) -> U128 {
  ...
  for (receiver_id, amount) in payout {
    if receiver_id == market_data.owner_id {
      Promise::new(receiver_id).transfer(amount.0 - treasury_fee);
      if treasury_fee != 0 {
        Promise::new(self.treasury_id.clone()).transfer(treasury_fee);
      }
      ...
    }
    ...
  }
  ...
}
```

Recommendation

We recommend conforming to the NEP-199 Royalties and Payouts by subtracting a commission prior to the `nft_transfer_payout` call. Safe math and promise result verification are also good practices.

C1-03 Bad implementation of storage management

Medium Open

In the contract, there is an incomplete implementation of storage management. For example, bids can make market data exceed 859 bytes. Because of that, users' funds can be locked in the contract.

```
pub const STORAGE_ADD_MARKET_DATA: u128 = 8590000000000000000000;
#[payable]
pub fn add_bid(...) {
  ...
  bids.push(new_bid);
  ...
  if updated_bids.len() >= 100 {
    self.internal_cancel_bid(nft_contract_id.clone(), token_id.clone(),
updated_bids[0].bidder_id.clone())
  }
  ...
```

```
}
```

Recommend

Either the `STORAGE_ADD_MARKET_DATA` constant should be increased or a check of storage deposit should be added to the `add_bid()` function.

C1-04 Not enough funds for a transfer

Medium Open

In the function `buy()`, `internal_accept_offer_series()`, and `internal_accept_offer()` if the buyer attaches a value that equals variable price then treasury won't receive his funds because 1 YOCTO from the user's attached value was attached to the NFT transfer and because of that contract may not have enough funds for a transfer.

Recommendation

In the `buy()` function, line 475

```
env::attached_deposit() >= price
```

should be replaced with

`env::attached_deposit() == price + 1` or `env::attached_deposit() > price`

In the case of `internal_accept_offer_series()` and `internal_accept_offer()`, it should help if someone predeposits some NEAR to the contract (approximately 10^5 YOCTO).

C1-05 Unlimited fee amount

Low Open

The contract admin has the ability to set any fee up to 100% using the `set_transaction_fee()` function.

```
#[payable]
pub fn set_transaction_fee(&mut self, next_fee: u16, start_time: Option<TimestampSec>) {
  ...
```

```
  assert!(next_fee < 10_000, "Paras: fee is higher than 10_000");
  ...
}
```

Recommendation

It is necessary to limit the fee percent that the admin can set.

C1-06 Treasury fee won't be transferred

Low Open

In `resolve_purchase()` and `resolve_offer()` functions in the cycle in line 609 if a payout doesn't contain an `owner_id` address the treasury fee won't be transferred.

```
#[private]
pub fn resolve_purchase(ююю) -> U128 {
  ...
  for (receiver_id, amount) in payout {
     if receiver_id == market_data.owner_id {
         ...
     } else {
         Promise::new(receiver_id).transfer(amount.0);
     }
  }
  ...
}
```

C1-07 Division before multiplication

Low Open

Division before multiplication can, in some rare cases, cause calculation errors in integer math. In this contract, division before multiplication is performed in L468, L1704, L1706.

```
#[payable]
pub fn buy(...) {
  ...
  price = price - ((price - end_price) / duration as u128) * time_since_start as u128;
  ...
```

```
}

#[payable]
pub fn add_bid(...) {
  ...
  assert!(amount.0 >= current_bid.price.0 + (current_bid.price.0 / 100 * 5),
            "Paras: Can't pay less than or equal to current bid price + 5% : {:?}",
            current_bid.price.0 + (current_bid.price.0 / 100 * 5));
  ...
}
```

Recommendation

It is recommended to change the order of execution of arithmetic operations so that it does not violate the logic of the function but also gets rid of a possible error in integer mathematics.

C1-08 Transaction fee view method

Low Open

The function `get_market_data_transaction_fee()` may return the old transaction fee. This can happen if timestamp is greater than `transaction_fee.start_time`, so the next call of `calculate_current_transaction_fee()` will update the fee value.

```
pub fn get_transaction_fee(&self) -> &TransactionFee {
  &self.transaction_fee
}

pub fn calculate_current_transaction_fee(&mut self) -> u128 {
  ...
  if to_sec(env::block_timestamp()) >= transaction_fee.start_time.unwrap() {
    self.transaction_fee.current_fee = transaction_fee.next_fee.unwrap();
    ...
  }
  ...
}
```

Recommendation

It is recommended in the `get_market_data_transaction_fee()` function to check whether a new fee has started using condition constructions, if it has started then return it, if it has not started then return the current fee. Since the `get_market_data_transaction_fee()` function is oriented towards lower gas consumption, it is not necessary to call the `calculate_market_data_transaction_fee()` function in the getter function, as this will significantly increase gas consumption.

C1-09 Accept bids for ended auctions

● Info ⓘ Open

In the `accept_bid()` function, the owner of the contract can accept bids for finished auctions.

```
if env::predecessor_account_id() == self.owner_id && market_data.ended_at.is_some() {
  assert!(
    current_time >= market_data.ended_at.unwrap(),
    "Paras: Auction has not ended yet"
  );
}
```

C1-10 Doesn’t check length of payout

● Info ⓘ Open

The contract doesn't check the length of a payout that is received from NFT in L609, L1092. This can result in wasting gas to perform a function. According to NEP-199 financial contracts MUST validate several invariants on the returned

Payout: the returned Payout MUST be no longer than the given maximum length (`max_len_payout` parameter) if provided.

C1-11 No checked math used

Info Open

In the `storage_withdraw()` function, the `amount` variable is of type unsigned int, if checked math is not used, then the `amount` variable may overflow.

C1-12 Typos and comments

Info Open

Typos reduce the code's readability. In L62 'DELIMETER' should be replaced with 'DELIMITER'.

Inconsistent comment in L1736 'Remove first element if bids.length > 50'. The adjacent code checks if `bids.len() >= 100`.

C1-13 No checks for promise results with transfer native tokens

Info Open

Results of promises that transfer native tokens aren't checked. In order to make sure that the promise is executed correctly, it's required to check the results of the promises.

C1-14 Remainder in NFT contracts

Info Open

NEP-199 states that "Financial contracts MAY take a cut of the NFT sale price as commission, subtracting their cut from the total token sale price, and calling nft_transfer_payout with the remainder", but in this contract, this isn't done.

C2. nft_callbacks.rs

Overview

This is a set of approval callbacks for NFT contracts. No issues were found.

5. Conclusion

2 high and 2 medium severity issues were found.

The reviewed contract is highly dependent on the owner’s account. Users using the project have to trust the owner and that the owner's account is properly secured.

This audit includes recommendations on code improvement and the prevention of potential attacks.

Appendix. Issues' severity classification

Critical. Issues that may cause an unlimited loss of funds or entirely break the contract workflow. Malicious code (including malicious modification of libraries) is also treated as a critical severity issue. These issues must be fixed before deployments or fixed in already running projects as soon as possible.

High. Issues that may lead to a limited loss of funds, break interaction with users, or other contracts under specific conditions. Also, issues in a smart contract, that allow a privileged account the ability to steal or block other users' funds.

Medium. Issues that do not lead to a loss of funds directly, but break the contract logic. May lead to failures in contracts operation.

Low. Issues that are of a non-optimal code character, for instance, gas optimization tips, unused variables, errors in messages.

Informational. Issues that do not impact the contract operation. Usually, informational severity issues are related to code best practices, e.g. style guide.

 contact@hashex.org

 @hashex_manager

 blog.hashex.org

 linkedin

 github

 twitter

#HashEx
BLOCKCHAIN SECURITY