verichains

SECURITY AUDIT OF

YIELDLY MARKETPLACE SMARTCONTRACT

yieldly

Public Report

Sep 05, 2022

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

EXECUTIVE SUMMARY

This Security Audit Report prepared by Verichains Lab on Sep 05, 2022. We would like to thank the Yieldly Finance for trusting Verichains Lab in auditing smart contracts. Delivering high-quality audits is always our top priority.

This audit focused on identifying security flaws in code and the design of the Yieldly Marketplace Smartcontract. The scope of the audit is limited to the source code files provided to Verichains. Verichains Lab completed the assessment using manual, static, and dynamic analysis techniques.

During the audit process, the audit team had identified some vulnerable issues in the application, along with some recommendations.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About Yieldly Marketplace Smartcontract

yNFT is Yieldly's one stop marketplace for all Algorand NFTs. yNFT incorporates a unique focus on Game-Fi; driving access for Algorand's P2E community to list in-game NFTs in front of the largest Algorand community – Yieldly. yNFT sets itself apart by boasting the most user-friendly minting, listing and trading features across the entire ecosystem, improving the whole NFT experience for Algorand.

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of the Yieldly Marketplace Smartcontract.

It was conducted on the source code provided by Yieldly Finance team. The latest version of the following repositories were made available in the course of the review:

Repository	Commit
https://github.com/yieldly-finance/ynft-contract-service	41f8d272a2b48fab79b3f71eb9581c6e267b11ce

1.3. Audit methodology

Our security audit process for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using public and our in-house smart contract security analysis tool.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

- Integer Overflow and Underflow
- Improper Token Precision Handling
- Race Conditions
- Transaction-Ordering Dependence
- DoS with (Unexpected) revert
- Unsafe Memory Handling
- Reentrancy
- Action Authorizations
- Unsafe Random Number Generator

- Logic Flaws

For vulnerabilities, we categorize the findings into categories as listed in table below, depending on their severity level:

SEVERITY LEVEL	DESCRIPTION
CRITICAL	A vulnerability that can disrupt the contract functioning; creates a critical risk to the contract; required to be fixed immediately.
HIGH	A vulnerability that could affect the desired outcome of executing the contract with high impact; needs to be fixed with high priority.
MEDIUM	A vulnerability that could affect the desired outcome of executing the contract with medium impact in a specific scenario; needs to be fixed.
LOW	An issue that does not have a significant impact, can be considered as less important.

Table 1. Severity levels

1.4. Disclaimer

Please note that security auditing cannot uncover all existing vulnerabilities, and even an audit in which no vulnerabilities are found is not a guarantee for a 100% secure smart contract. However, auditing allows discovering vulnerabilities that were unobserved, overlooked during development and areas where additional security measures are necessary.

2. AUDIT RESULT

2.1. Overview

The Yieldly Marketplace Smartcontract was written in `Reach` language which requires version to be `0.1`. These contracts are used to deploy to the Algorand blockchain. Below is the summary of the repositories in the audit scope:

2.1.1. Auction contract

Auction contract allows users to create NFT auctions and place a bid. The platform will charge 5% platform fee, creator fund fee and royalty fee when users make a transaction in the market.

2.1.2. Buynow contract

Buynow contract allows users to buy, sell, make offers and accept offers for NFT. The platform will charge 5% platform fee, creator fund fee and royalty fee when users make a transaction in the market.

2.2. Findings

During the audit process, the audit team found some vulnerabilities in the given version of Yieldly Marketplace Smartcontract. Yieldly Finance fixed the code, according to Verichains's private reports.

2.2.1. Nonsense/Wrong auction continuation condition CRITICAL

Look at the current continuation logic in `auction.rsh`:

```
const [keepGoing, highestBidder, lastPrice, isFirstBid, dlSecs] =
    parallelReduce([true, AuctionCreator, minBid, true, end])
    // ...
      .while(keepGoing)
      .api(
        // ...
        (bid, notify) => {
          //...
          return [
            // keepGoing,
            lastConsensusSecs() < dlSecs,
            who,
            bid,
            false,
            nextDlSecs(lastConsensusSecs(), dlSecs),
          ];
        }
```

```
      )
      .timeout(false);
```

Above snippet be summarized as below:

```
dlSecs = end

action bid():
    if keepGoing:
        # bid logic ...
        keepGoing = lastTxTime < dlSecs
        dlSecs = calcNextDlSecs(lastTxTime, nextDlSecs)
    lastTxTime = now
```

or in another way:

```
dlSecs = end

action bid():
    if not finished:
        # bid logic ...
        finished = lastTxTime >= lastDlSecs
    lastTxTime = now
    lastDlSecs = dlSecs
```

So in the current code, anyone just need to wait until someone bid after `dlSecs` and perform the bid, the auction will be finished with him as the final winner.

We can confirm this in generated solidity contract by running `./reach compile --intermediate-files ynft-reach-contracts/auction.rsh`.

The generated `bid` function for `Bidder`:

```
function Bidder_bid(uint256 _a0)  external payable returns (T11  memory) {
    T14 memory _t;
    _t.msg.v355.elem0 = _a0;
    {
      ApiRng memory _r;
      _reach_m4(_t, _r);
      return _r.Bidder_bid;
      }
    }
```

It call internal function `_reach_m4` with relating error logic summary as below:

```
function _reach_m4(T14 memory _a, ApiRng memory _apiRet)  internal  {
    reachRequire((current_step == uint256(6)), uint256(31) /*'state step check at ./ynft-reach-contracts/auction.rsh:100:19:dot'*/);
    reachRequire(((_a.time == uint256(0)) || (current_time == _a.time)), uint256(32) /*'state time check at ./ynft-reach-contracts/auction.rsh:100:19:dot'*/);
    current_step = 0x0;
    (T3 memory _svs) = abi.decode(current_svbs, (T3));
```

```
    _F4 memory _f;

    emit _reach_e4(msg.sender, _a);
    _f.v361 = safeAdd(_svs.v328, (_a.msg.v355.elem0));
    reachRequire((msg.value == (_a.msg.v355.elem0)), uint256(29) /*'(./ynft-reach-
contracts/auction.rsh:100:19:dot,[],"verify network token pay amount")'*/);
    reachRequire((((_a.msg.v355.elem0) > _svs.v320)), uint256(30) /*'(./ynft-reach-
contracts/auction.rsh:120:18:application,[at ./ynft-reach-
contracts/auction.rsh:119:23:application call to [unknown function] (defined at: ./ynft-
reach-contracts/auction.rsh:119:23:function exp)],Just "bid is too low")'*/);
    _f.v364.elem0 = _svs.v317;
    _f.v364.elem1 = _svs.v320;

    emit _reach_oe_v364( _f.v364);
    _apiRet.Bidder_bid = _f.v364;

    if (_svs.v318) { // _svs.v318 is isFirstBid variable in rsh
        // ...
    } else {
      _svs.v317.transfer(_svs.v320);
      T8 memory la;
      la.svs.v285 = _svs.v285;
      la.svs.v286 = _svs.v286;
      la.svs.v289 = _svs.v289;
      la.svs.v290 = _svs.v290;
      la.svs.v291 = _svs.v291;
      la.svs.v292 = _svs.v292;
      la.svs.v293 = _svs.v293;
      la.svs.v315 = _svs.v315;
      la.msg.v316 = ((_svs.v324 > (safeSub(_svs.v316, uint256(60)))) ? (safeAdd(_svs.v316,
uint256(120))) : _svs.v316);
      la.msg.v317 = payable(msg.sender);
      la.msg.v318 = false;
      la.msg.v319 = (_svs.v324 < _svs.v316); // last bid's time < last bid's dlSecs
      la.msg.v320 = (_a.msg.v355.elem0);
      la.msg.v321 = uint256(block.number);
      la.msg.v324 = uint256(block.timestamp);
      la.msg.v328 = (safeSub(_f.v361, _svs.v320));
      l2(la);
    }
```

In above uglify generated code, `_svs` stores the last state, `la.msg` stores current state. We also need to look at the state transition function `l2`:

```
function l2(T8 memory _a)  internal {
    _F2 memory _f;

    if (_a.msg.v319) {
      // ...
      }
```

```
    else {
      // ...
      current_step = uint256(3);
      current_time = uint256(block.number);
      current_svbs = abi.encode(nsvs);
      }

    }
```

So the state machine switch to `3` (exit state) when `_a.msg.v319` is `false`, i.e. the above highlighted code:

```
la.msg.v319 = (_svs.v324 < _svs.v316); // last bid's time < last bid's dlSecs
```

2.2.1.1. Recommends

Check Reach's example auction with basic deadline: *https://github.com/reach-sh/reach-lang/blob/47d09e54faaa43bce495e5c1a2ea432b5bcbb7f4/examples/nft-auction/index.rsh*. Extendable deadline can be implemented in similar way.

UPDATES

- *Sep 01, 2022*: This issue has been acknowledged and fixed by Yieldly Finance team.

2.2.2. Round-off in fee calculations INFORMATIVE

Fee calculations is performed by divide-and-then-multiple, which will result in rounding off in the amount that royalty, platform and creator receive.

```
  const bal = balance() / 100;  const royaltyAmount = royaltyPercentageToUse * bal;  const
platformAmount = PLATFORM_FEE * bal;  const creatorFundAmount = CREATOR_FUND * bal;  const
recvAmount =    balance() -    (royaltyAmount + platformAmount + creatorFundAmount);
```

Corresponding generated code:

```
      _f.v398 = safeAdd(_a.msg.v320, (safeMul((safeDiv(_a.msg.v320, uint256(100))),
_a.svs.v293)));      _f.v402 = safeDiv(_a.msg.v328, uint256(100));      _f.v403 =
safeMul(((_a.svs.v289 > uint256(25)) ? _a.svs.v289 : uint256(25)), _f.v402);      _f.v404 =
safeMul(uint256(5), _f.v402);      _f.v409 = safeSub(_a.msg.v328,
(safeAdd((safeAdd(_f.v403, _f.v404)), _f.v404)));
```

UPDATES

- *Sep 01, 2022*: This issue has been acknowledged by Yieldly Finance team.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Sep 05, 2022*	Public Report	Verichains Lab

Table 2. Report versions history