verichains

SECURITY AUDIT OF

VERA RENT SMART CONTRACT

vera

Public Report

May 17, 2022

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

ABBREVIATIONS

Name	Description
Ethereum	An open source platform based on blockchain technology to create and distribute smart contracts and decentralized applications.
Ether (ETH)	A cryptocurrency whose blockchain is generated by the Ethereum platform. Ether is used for payment of transactions and computing services in the Ethereum network.
Smart contract	A computer protocol intended to digitally facilitate, verify or enforce the negotiation or performance of a contract.
Solidity	A contract-oriented, high-level language for implementing smart contracts for the Ethereum platform.
Solc	A compiler for Solidity.
ERC20	ERC20 (BEP20 in Binance Smart Chain or *x*RP20 in other chains) tokens are blockchain-based assets that have value and can be sent and received. The primary difference with the primary coin is that instead of running on their own blockchain, ERC20 tokens are issued on a network that supports smart contracts such as Ethereum or Binance Smart Chain.

EXECUTIVE SUMMARY

This Security Audit Report prepared by Verichains Lab on May 17, 2022. We would like to thank the Vera Labs Inc 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 Vera Rent Smart Contract. 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 smart contracts code.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About Vera Rent Smart Contract

Vera is a decentralized protocol built on top of major blockchains that allows essential financial services for NFTs such as renting, lending, and mortgages. Decentralized protocols are non-custodial, meaning digital assets and NFTs are not owned by any middleman or intermediary.

VERA is the native utility token for the platform used for governance and fee collections purposes.

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of the Vera Rent Smart Contract.

It was conducted on commit 9b86dead4e525b203c39d3a24b011d98b32e69c0 from git repository *https://github.com/istepofficial/token-smartcontract*.

The latest version of the following files were made available in the course of the review:

SHA256 Sum	File
39822b2c8e36154e29080112115ad2c5b67da123636f669ebbbfb6dab1af8ea0	**IvRent.sol**
b85705212a30c9425600e237c3fd489bb559cd1e52da57b7a1fba14aca10e18e	**vRent.sol**

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 RK87, 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
- Timestamp Dependence
- Race Conditions
- Transaction-Ordering Dependence

- DoS with (Unexpected) revert
- DoS with Block Gas Limit
- Gas Usage, Gas Limit and Loops
- Redundant fallback function
- Unsafe type Inference
- Reentrancy
- Explicit visibility of functions state variables (external, internal, private and public)
- 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 Vera Rent Smart Contract was written in Solidity language, with the required version to be 0.8.6. The source code was written based on OpenZeppelin's library.

2.1.1. vRent contract

vRent is a lending and renting contract which extends ERC721Holder, ERC1155Receiver and ERC1155Holder contracts. This contract allows users to list their ERC721 and ERC1155 tokens on the marketplace with their desired rental terms. Collateralized renting requires the borrower to deposit collateral to rent NFT while non-collateralized renting does not require any collateral.

2.2. Findings

During the audit process, the audit team had identified some vulnerable issues in the smart contracts code.

2.2.1. vRent.sol - Leaser can not stop leasing if renter keep renting NFT immediately after returning [HIGH]

In manageStopLeasing function, the contract checks for _isNull(item.renting); so if renter keep renting NFT immediately after returning again and again, the leaser will not be able to stop leasing and get back the NFT until the renter stop.

```
function manageStopLeasing(CallData memory _cd) private {
    uint256[] memory leaseAmounts = new uint256[](_cd.right - _cd.left);

    for (uint256 i = _cd.left; i < _cd.right; i++) {
        LeasingRenting storage item =
        leasingRenting[
        keccak256(
            abi.encodePacked(
                _cd.nfts[_cd.left],
                _cd.tokenIds[i],
                _cd.leasingIds[i]
            )
        )
        ];

        _isNotNull(item.Leasing);
        _isNull(item.renting);
```

```
        _isStoppable(item.Leasing, msg.sender);

        leaseAmounts[i - _cd.left] = item.Leasing.leaseAmount;

        emit LeasingStopped(_cd.leasingIds[i], uint32(block.timestamp));

        delete item.Leasing;
    }

    _safeTransfer(
        _cd,
        address(this),
        msg.sender,
        _sliceArr(_cd.tokenIds, _cd.left, _cd.right, 0),
        _sliceArr(leaseAmounts, _cd.left, _cd.right, _cd.left)
    );
}
```

RECOMMENDATION

Adding enabled variable to Leasing and check in manageLease to stop new rental.

UPDATES

- *May 17, 2022*: This issue has been acknowledged and fixed by the Vera Labs Inc team.

2.2.2. vRent.sol - Pausing contract can make renter lose his collateral [HIGH]

Renter can only endRent and get back collateral when notPaused so if admin pauses contract, it makes renter unable to endRent and after passing return date, the renter will lose his collateral unwillingly.

```
function endRent(
    address[] memory _nfts,
    uint256[] memory _tokenIds,
    uint256[] memory _leasingIds
) external override notPaused {
    bundleCall(
        manageReturn,
        _createActionCallData(_nfts, _tokenIds, _leasingIds)
    );
}
```

```
function _isReturnable(
    Renting memory _renting,
    address _msgSender,
    uint256 _blockTimestamp
) private pure {
    require(_renting.renterAddress == _msgSender, "vRent::not renter");
    require(
        !_isPastReturnDate(_renting, _blockTimestamp),
        "vRent::past return date"
    );
}

function _isPastReturnDate(Renting memory _renting, uint256 _now)
    private
    pure
    returns (bool)
{
    require(_now > _renting.rentedAt, "vRent::now before rented");
    return
    _now - _renting.rentedAt > _renting.rentDuration * SECONDS_IN_DAY;
}
```

RECOMMENDATION

Removing notPaused from endRent.

UPDATES

- *May 17, 2022*: This issue has been acknowledged and fixed by the Vera Labs Inc team.

2.2.3. vRent.sol - Wrong require variable [LOW]

In distributeClaimPayment, there is a require statement for collateral plus rent is zero but the condition check for maxRentPayment instead of finalAmt.

```
function distributeClaimPayment(LeasingRenting memory _LeasingRenting)
private
{
    uint8 paymentTokenIx = uint8(_LeasingRenting.Leasing.paymentToken);
    tokenNotSentinel(paymentTokenIx);
    ERC20 paymentToken = ERC20(payment.getPaymentToken(paymentTokenIx));

    uint256 decimals = ERC20(paymentToken).decimals();
    uint256 scale = 10**decimals;
```

```
    uint256 collateralAmount =
    _LeasingRenting.Leasing.leaseAmount *
    _unwrapPrice(_LeasingRenting.Leasing.collateralAmount, scale);
    uint256 rentPrice =
    _unwrapPrice(_LeasingRenting.Leasing.dailyLeasePrice, scale);
    uint256 maxRentPayment =
    rentPrice * _LeasingRenting.renting.rentDuration;
    uint256 takenFee =
    takeFee(maxRentPayment, IPayment.PaymentToken(paymentTokenIx));
    uint256 finalAmt = maxRentPayment + collateralAmount;

    require(maxRentPayment > 0, "vRent::collateral plus rent is zero");

    paymentToken.safeTransfer(
        _LeasingRenting.Leasing.leaserAddress,
        finalAmt - takenFee
    );
}
```

RECOMMENDATION

Changing the require condition to finalAmt.

```
function distributeClaimPayment(LeasingRenting memory _LeasingRenting)
private
{
    uint8 paymentTokenIx = uint8(_LeasingRenting.Leasing.paymentToken);
    tokenNotSentinel(paymentTokenIx);
    ERC20 paymentToken = ERC20(payment.getPaymentToken(paymentTokenIx));

    uint256 decimals = ERC20(paymentToken).decimals();
    uint256 scale = 10**decimals;
    uint256 collateralAmount =
    _LeasingRenting.Leasing.leaseAmount *
    _unwrapPrice(_LeasingRenting.Leasing.collateralAmount, scale);
    uint256 rentPrice =
    _unwrapPrice(_LeasingRenting.Leasing.dailyLeasePrice, scale);
    uint256 maxRentPayment =
    rentPrice * _LeasingRenting.renting.rentDuration;
    uint256 takenFee =
    takeFee(maxRentPayment, IPayment.PaymentToken(paymentTokenIx));
    uint256 finalAmt = maxRentPayment + collateralAmount;
```

```
    require(finalAmt > 0, "vRent::collateral plus rent is zero");

    paymentToken.safeTransfer(
        _LeasingRenting.Leasing.leaserAddress,
        finalAmt - takenFee
    );
}
```

UPDATES

- *May 17, 2022*: This issue has been acknowledged and fixed by the Vera Labs Inc team.

2.2.4. vRent.sol - Wrong documentation comment [INFO]

In setRentFee, 100% is 10000 but the comment in rentFee variable declaration said that 1000 => 1% which must be 100 => 1%

```
// in bps. so 1000 => 1%
uint256 public rentFee = 0;

/**
 * @dev only Admin can call this function
     * set the platform `rentFee`
     * `+_rentFee` should be less than 100%
    */
function setRentFee(uint256 _rentFee) external onlyAdmin {
    require(_rentFee < 10000, "vRent::fee exceeds 100pct");
    rentFee = _rentFee;
}
```

RECOMMENDATION

Fix the comment.

UPDATES

- *May 17, 2022*: This issue has been acknowledged and fixed by the Vera Labs Inc team.

2.2.5. vRent.sol - Wrong documentation comment [INFO]

_isReturnable function only reverts when the NFT is not returnable or _msgSender is not renter, not returns anything like in the documentation comment.

```
/**
 * @dev compare the timestamp and return time and returns
```

```
 * whether the NFT is returnable or not
*/
function _isReturnable(
    Renting memory _renting,
    address _msgSender,
    uint256 _blockTimestamp
) private pure {
    require(_renting.renterAddress == _msgSender, "vRent::not renter");
    require(
        !_isPastReturnDate(_renting, _blockTimestamp),
        "vRent::past return date"
    );
}
```

RECOMMENDATION

Fix the comment.

UPDATES

- *May 17, 2022*: This issue has been acknowledged by the Vera Labs Inc team.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*May 13, 2022*	Public Report	Verichains Lab
1.1	*May 17, 2022*	Public Report	Verichains Lab

Table 2. Report versions history