verichains

SECURITY AUDIT OF

WINERY SMART CONTRACTS

Public Report

July 25, 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.
BSC	Binance Smart Chain or BSC is an innovative solution for introducing interoperability and programmability on Binance Chain.
DEX	Decentralized exchanges (DEX), like PancakeSwap or Uniswap, are autonomous financial protocols powered by smart contracts that enable crypto traders to convert one digital asset for another with all transactions viewable on the blockchain.
AMM	An automated market maker (AMM) is a type of decentralized exchange (DEX) protocol that relies on a mathematical formula to price assets. Instead of using an order book like a traditional exchange, assets are priced according to a pricing algorithm.

EXECUTIVE SUMMARY

This Security Audit Report prepared by Verichains Lab on July 25, 2022. We would like to thank the Winery 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 Winery Smart Contracts. 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 Winery Smart Contracts

WINERY is a decentralized cryptocurrency exchange project based on Binance Smart Chain (BEP20) that operates by connecting cryptocurrency traders in a peer-to-peer (P2P) manner using the Automated Market Maker Model (AMM). It is a platform that enables users to trade a huge variety of BEP-20 tokens via smart contracts to eliminate the risk of the counterparty.

The main functions of Winery DEX are token trading, liquidity supply, and other liquidity provided functions, farming and staking function, lotto function. Winery is connected to the Binance Smart Chain and supports a large number of Blockchain wallets like MetaMask, TrustWallet, and TokenPocket as well as a huge number of tokens. The rewards traders get from the main functions of Winery DEX will be paid with CORK.

CORK is Winery's special utility token issued on the Binance Smart Chain as a BEP-20 token. The main function of CORK is to incentivize those who add liquidity to the WINERY platform. Users can stake their tokens, deposit liquidity provider tokens, farm (lock-up) them, and be rewarded with more CORKs accordingly. CORK can be stored on your blockchain wallet or used to buy non-fungible tokens and lottery tickets with WINERY.

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of the smart contracts of Winery Smart Contracts. It was conducted on the source code provided by the Winery team.

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

SHA256 Sum	File
92cff2cb29cbc66338b0816e417753aa2ddaf8ba4ba0aa2c4a2da4237ea5f102	./WineryAuction.sol
97d55e40239a4dd74931995c9e5cacac03fb2b67bd44b41e7f54d1e47fcd1b6e	./WineryNFT.sol
310a6d9879e30344540f04d7ed1154d12d8464f271dbcea5b598c6f8f3af433f	./CORK.sol
f15c82ddf450de4498c5b388e8669182b5ac44ef74012f5a1b6e9acf0d91aa40	./WineryRouter.sol
6a7f5a76d45fe3e20348a755a747f17e4e8dc154478442513cea35f32edf07e1	./WineryMarket.sol
fed541560af6404ab93ea14e4805c992ff8b41c8483f7ce3874bd4a211108d27	./MasterChef.sol
85d12a9ee22d0dcd9ef0fa6e3cd222ecc3f48309a0e3c9277ba790c82cbcb010	./WineryPriceGetter.sol

a1a1678becbcd8af127821243ab36ecba50938c4d18ce1efaa3929d3119e393b	./WineryFactory.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.

SEVERITY LEVEL	DESCRIPTION
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 Winery Smart Contracts was written in `Solidity` language which was written based on OpenZeppelin's library and PancakeSwap contracts.

2.1.1. CORK token contract

The CORK token contract is a BEP20 token contract. Beside the default BEP20 functions, the contract implements some more functions for delegate voting so that the token can be used for governance. Furthermore, the contract owner can mint tokens at any time.

The table below lists some properties of the CORK token contract (as of the report writing time).

PROPERTY	VALUE
Name	WINERY TOKEN CORK
Symbol	CORK
Decimals	18

Table 2. The CORK token contract properties

2.1.2. MasterChef contract

Winery MasterChef contract is a modified version of PancakeSwap MasterChef contract, which uses CORK as the reward token.

2.1.3. WineryFactory and WineryRouter contracts

WineryFactory and WineryRouter are also modified versions of PancakeFactory and PancakeRouter contracts which support token trading via AMM protocol.

2.1.4. WineryNFT contract

The WineryNFT contract is an ERC721 token contract. The contract extends `ERC721EnumerableUpgradeable`, and `AccessControlUpgradeable` contracts. The `AccessControl` contract allows the inherited contract to implement role-based access control mechanisms which adds multiple roles to the contract likes `TOKEN_FREEZER`, `TOKEN_MINTER`, `LAUNCHPAD_TOKEN_MINTER` and `RB_SETTER`.

2.1.5. WineryAuction contract

WineryAuction is the contract that allows users to list their NFTs for auction.

2.1.6. WineryMarket contract

This contract provides a marketplace where users can buy/sell NFTs to each other in a decentralized manner.

2.2. Findings

During the audit process, the audit team found some vulnerabilities in the given version of Winery Smart Contracts. Winery team fixed the code, according to Verichains's draft report.

#	Issue	Severity	Status
1	Deposited tokens are mixed with reward tokens	CRITICAL	Fixed
2	The `onERC721Received` function can be called directly	MEDIUM	Fixed
3	The `bid` function is vulnerable to front-running attacks	LOW	Not fixed
4	Unexpected revert by integer underflow	LOW	Not fixed
5	Missing `robiBoost` limit check for new level	LOW	Not fixed
6	Missing level limit check	LOW	Not fixed

Audit team also suggested some possible enhancements.

#	Issue	Status
1	Redundant checks in the `WineryMarket` contract	Not fixed
2	NFT token addresses should be whitelisted instead of blacklisted	Not fixed
3	Token info should be deleted when burning	Not fixed

2.2.2. Deposited tokens are mixed with reward tokens CRITICAL

Affected files:

- MasterChef.sol

In the `MasterChef` contract, when users stake CORK tokens to pool with id `0`, both deposited CORK tokens and reward CORK tokens are held inside this contract. It means that users' deposited CORKS tokens can be used as reward tokens for other pools. This may lead to some situations where users can not unstake their CORKs since the contract CORK balance is insufficient.

```
function leaveStaking(uint256 _amount) public {
    PoolInfo storage pool = poolInfo[0];
    UserInfo storage user = userInfo[0][msg.sender];
    require(user.amount >= _amount, "withdraw: not good");
    updatePool(0);
    uint256 pending = user.amount.mul(pool.accCorkPerShare).div(1e12).sub(
        user.rewardDebt
    );
    if (pending > 0) {
        safeCorkTransfer(msg.sender, pending);
    }

    if (_amount > 0) {
        user.amount = user.amount.sub(_amount);
        pool.lpToken.safeTransfer(address(msg.sender), _amount); // BALANCE MAY BE
INSUFFICIENT
        depositedCork = depositedCork.sub(_amount);
    }
    user.rewardDebt = user.amount.mul(pool.accCorkPerShare).div(1e12);
    emit Withdraw(msg.sender, 0, _amount);
}
```

RECOMMENDATION

The reward CORK tokens should be put inside a separate reward vault contract.

UPDATES

- *July 20, 2022*: This issue has been acknowledged and fixed by Winery team, reward CORKs have been put in a separate vault at `rewardAddress` as below:

```
function leaveStaking(uint256 _amount) public {
    PoolInfo storage pool = poolInfo[0];
    UserInfo storage user = userInfo[0][msg.sender];
    require(user.amount >= _amount, "withdraw: not good");

    updatePool(0);
```

```
    uint256 pending = user.amount.mul(pool.accCorkPerShare).div(1e12).sub(
        user.rewardDebt
    );
    if (pending > 0) {
        // Transfer from reward pool
        safeCorkTransferFrom(rewardAddress, msg.sender, pending); // FIXED
    }

    if (_amount > 0) {
        user.amount = user.amount.sub(_amount);
        pool.lpToken.safeTransfer(address(msg.sender), _amount);
        depositedCork = depositedCork.sub(_amount);
    }

    user.rewardDebt = user.amount.mul(pool.accCorkPerShare).div(1e12);
    emit Withdraw(msg.sender, 0, _amount);
}
```

2.2.3. The `onERC721Received` function can be called directly MEDIUM

Affected files:

- WineryAuction.sol

The `onERC721Received` function can be called directly by any user, so we should not depend on this function for selling.

```
function onERC721Received(
    address operator,
    address from,
    uint256 tokenId,
    bytes calldata data
) external override whenNotPaused returns (bytes4) {
    if (data.length > 0) {
        require(operator == from, "caller should own the token");
        require(!nftBlacklist[IERC721(msg.sender)], "token not allowed");
        (IERC20 currency, uint256 askPrice, uint256 endTimestamp) = abi
            .decode(data, (IERC20, uint256, uint256));
        TokenPair memory pair = TokenPair({
            nft: IERC721(msg.sender),
            tokenId: tokenId
        });
        _sell(from, pair, currency, askPrice, endTimestamp);
    } else {
        require(_canReceive, "cannot transfer directly");
    }

    return this.onERC721Received.selector;
}
```

RECOMMENDATION

The NFT token address should be whitelisted before using with this auction contract.

UPDATES

- *July 20, 2022*: This issue has been acknowledged and fixed by removing the `onERC721Received` function.

2.2.4. The `bid` function is vulnerable to front-running attacks LOW

Affected files:

- WineryAuction.sol

In the `bid` function, the old bidder can receive an `incentive` fee amount which is extracted from the `offer` amount of the new bidder. Some attackers can listen for new bid which has a much large offer value than the previous offer. When there is such new bid is submitted to the blockchain, they can quickly inject their bid before the new bid (front-running attack) to collect the `incentive` fee from the new bidder. The reward amount is based on the difference between the new offer with the old offer and the `bidderIncentiveRate`.

```
function bid(uint256 id, uint256 offer)
    public
    _hasAuction(id)
    _isStOpen(id)
    nonReentrant
    whenNotPaused
    notContract
{
    Inventory storage inv = auctions[id];
    require(block.timestamp < inv.endTimestamp, "auction finished");

    // minimum increment
    require(offer >= getMinBidPrice(id), "offer not enough");

    inv.currency.safeTransferFrom(msg.sender, address(this), offer);

    // transfer some to previous bidder
    uint256 incentive = 0;
    if (inv.netBidPrice > 0 && inv.bidder != address(0)) {
        incentive = (offer * bidderIncentiveRate) / rateBase;
        _transfer(inv.currency, inv.bidder, inv.netBidPrice + incentive);
    }

    inv.bidPrice = offer;
    inv.netBidPrice = offer - incentive;
    inv.bidder = msg.sender;
```

```
    if (block.timestamp + extendEndTimestamp >= inv.endTimestamp) {
        inv.endTimestamp += prolongationTime;
    }

    emit NewBid(id, msg.sender, offer, inv.netBidPrice, inv.endTimestamp);
}
```

RECOMMENDATION

We can add a new parameter `previousOffer` to the `bid` function as below to avoid front-running attacks:

```
function bid(uint256 id, uint256 offer, uint256 previousOffer)
    public
    _hasAuction(id)
    _isStOpen(id)
    nonReentrant
    whenNotPaused
    notContract
{
    // IMPLEMENT THE CHECK FOR `previousOffer`
    // ...
}
```

UPDATES

- *July 20, 2022*: This issue has been acknowledged by Winery team.

2.2.5. Unexpected revert by integer underflow LOW

Affected files:

- WineryNFT.sol

In the `WineryNFT` contract, two functions `_sendRBToToken` and `exchangeRB` both calculate the `robiBoost` amount before increasing the `robiBoost` for the specified tokens. Consider the `_sendRBToToken` function below:

```
function _sendRBToToken(uint256[] memory tokenId, uint256[] memory amount)
    internal
{
    require(
        tokenId.length <= MAX_ARRAY_LENGTH_PER_REQUEST,
        "Array length gt max"
    );
    require(tokenId.length == amount.length, "Wrong length of arrays");
    for (uint256 i = 0; i < tokenId.length; i++) {
        require(ownerOf(tokenId[i]) == msg.sender, "Not owner of token");
        uint256 calcAmount = amount[i];
        uint256 period = _burnRBPeriod;
```

```
        uint256 currentRB;
        uint256 curDay;
        while (calcAmount > 0 || period > 0) {
            curDay = (block.timestamp - period * 1 days) / 86400;
            currentRB = _robiBoost[msg.sender][curDay];
            if (currentRB == 0) {
                period--; // POSSIBLE OF INTEGER UNDERFLOW
                continue;
            }
            if (calcAmount > currentRB) {
                calcAmount -= currentRB;
                _robiBoostTotalAmounts[curDay] -= currentRB;
                delete _robiBoost[msg.sender][curDay];
            } else {
                decreaseRobiBoost(msg.sender, curDay, calcAmount);
                calcAmount = 0;
                break;
            }
            period--; // POSSIBLE OF INTEGER UNDERFLOW
        }
        if (calcAmount == 0) {
            _gainRB(tokenId[i], amount[i]);
        } else {
            revert("Not enough RB balance"); // UNREACHABLE CODE
        }
    }
}
```

Consider the case when the RB balance is insufficient, the expected revert message should be "Not enough RB balance". However, when the `period` variable is equal to zero, the statement `period--;` will make the whole transaction reverted before reaching the final custom revert statement. So, the transaction will be reverted with an unclear message.

RECOMMENDATION

To add the custom revert message in case of insufficient RB balance, we must check the `period` variable, ensure that it is greater than zero before decreasing it.

UPDATES

- *July 20, 2022*: This issue has been acknowledged by Winery team.

2.2.6. Missing `robiBoost` limit check for new level LOW

Affected files:

- WineryNFT.sol

In the `_mintLevelUp` function, the `robiBoost` for the new token should not be greater than the maximum `robiBoost` for its level.

```
function _mintLevelUp(uint256 level, uint256[] memory tokenId) private {
    uint256 newRobiBoost = 0;
    for (uint256 i = 0; i < tokenId.length; i++) {
        require(ownerOf(tokenId[i]) == msg.sender, "Not owner of token");
        newRobiBoost += _tokens[tokenId[i]].robiBoost;
        _burn(tokenId[i]);
    }
    newRobiBoost = newRobiBoost + (newRobiBoost * _levelUpPercent) / 100;
    _lastTokenId += 1;
    uint256 newTokenId = _lastTokenId;
    _tokens[newTokenId].robiBoost = newRobiBoost;
    _tokens[newTokenId].createTimestamp = block.timestamp;
    _tokens[newTokenId].level = level;
    _safeMint(msg.sender, newTokenId);
}
```

RECOMMENDATION

Adding limit check for the `robiBoost` of the newly minted token.

```
function _mintLevelUp(uint256 level, uint256[] memory tokenId) private {
    uint256 newRobiBoost = 0;
    for (uint256 i = 0; i < tokenId.length; i++) {
        require(ownerOf(tokenId[i]) == msg.sender, "Not owner of token");
        newRobiBoost += _tokens[tokenId[i]].robiBoost;
        _burn(tokenId[i]);
    }
    newRobiBoost = newRobiBoost + (newRobiBoost * _levelUpPercent) / 100;
    require(newRobiBoost <= _rbTable[level], "RB value over limit by level"); // FIXED
    _lastTokenId += 1;
    uint256 newTokenId = _lastTokenId;
    _tokens[newTokenId].robiBoost = newRobiBoost;
    _tokens[newTokenId].createTimestamp = block.timestamp;
    _tokens[newTokenId].level = level;
    _safeMint(msg.sender, newTokenId);
}
```

UPDATES

- *July 20, 2022*: This issue has been acknowledged by Winery team.

2.2.7. Missing level limit check LOW

Affected files:

- WineryNFT.sol

The level limit check is missing in both functions `launchpadMint` and `levelUp` as below:

```
function launchpadMint(
    address to,
    uint256 level,
    uint256 robiBoost
) public onlyRole(LAUNCHPAD_TOKEN_MINTER) nonReentrant {
    require(to != address(0), "Address can not be zero");
    require(_rbTable[level] >= robiBoost, "RB Value out of limit");
    _lastTokenId += 1;
    uint256 tokenId = _lastTokenId;
    _tokens[tokenId].robiBoost = robiBoost;
    _tokens[tokenId].createTimestamp = block.timestamp;
    _tokens[tokenId].level = level;
    _safeMint(to, tokenId);
}

function levelUp(uint256[] calldata tokenId) public nonReentrant {
    require(
        tokenId.length <= MAX_ARRAY_LENGTH_PER_REQUEST,
        "Array length gt max"
    );
    uint256 currentLevel = _tokens[tokenId[0]].level;
    require(_levelTable[currentLevel] != 0, "This level not upgradable");
    uint256 numbersOfToken = _levelTable[currentLevel];
    require(
        numbersOfToken == tokenId.length,
        "Wrong numbers of tokens received"
    );
    uint256 neededRb = numbersOfToken * _rbTable[currentLevel];
    uint256 cumulatedRb = 0;

    for (uint256 i = 0; i < numbersOfToken; i++) {
        Token memory token = _tokens[tokenId[i]]; //safe gas
        require(token.level == currentLevel, "Token not from this level");
        cumulatedRb += token.robiBoost;
    }

    if (neededRb == cumulatedRb) {
        _mintLevelUp((currentLevel + 1), tokenId);
    } else {
        revert("Wrong robi boost amount");
    }
    emit LevelUp(msg.sender, (currentLevel + 1), tokenId);
}
```

UPDATES

- *July 20, 2022*: This issue has been acknowledged by Winery team.

2.3. Additional notes and recommendations

2.3.1. Redundant checks in the `WineryMarket` contract INFORMATIVE

Affected files:

- WineryMarket.sol

There are some redundant checks in the `placementRewardQualifier` and `_offerSell` functions as shown below:

```
function placementRewardQualifier(
    bool increase,
    address user,
    address nftToken
) internal {
    //Check if nft token in nftForAccrualRB list and accrue reward enable
    if (!nftForAccrualRB[nftToken] || !placementRewardEnabled) return;

    if (increase) {
        tokensCount[user]++;
    } else {
        tokensCount[user] = tokensCount[user] > 0
            ? tokensCount[user] - 1
            : 0;
    }
    if (tokensCount[user] > maxUserTokenOnSellToReward) return;

    uint256 stakedAmount = tokensCount[user] >= maxUserTokenOnSellToReward
        ? maxUserTokenOnSellToReward
        : tokensCount[user]; // REDUNDANT CHECK
    smartChefMarket.updateStakedTokens(user, stakedAmount);
}

function _offerSell(
    IERC721 nft,
    uint256 tokenId,
    uint256 price,
    address dealToken
) internal {
    require(msg.value == 0, "Seller should not pay"); // REDUNDANT CHECK
    require(price > 0, "price > 0");
    // ...
}
```

UPDATES

- *July 20, 2022*: This issue has been acknowledged by Winery team.

2.3.2. NFT token addresses should be whitelisted instead of blacklisted INFORMATIVE

Affected files:

- WineryMarket.sol
- WineryAuction.sol

Both contracts `WineryMarket` and `WineryAuction` use a mapping to save blacklisted NFT token addresses, which is not the recommended way to do that.

```
mapping(IERC721 => bool) public nftBlacklist;
```

RECOMMENDATION

We should whitelist NFT token addresses instead of blacklist them, so that we can avoid various phishing/scam NFT addresses to be used on the market or auction.

UPDATES

- *July 20, 2022*: This issue has been acknowledged by Winery team.

2.3.3. Token info should be deleted when burning INFORMATIVE

Affected files:

- WineryNFT.sol

In the `_mintLevelUp` function, the token info in `_tokens[tokenId[i]]` should be deleted when `token[i]` is burned.

```
function _mintLevelUp(uint256 level, uint256[] memory tokenId) private {
    uint256 newRobiBoost = 0;
    for (uint256 i = 0; i < tokenId.length; i++) {
        require(ownerOf(tokenId[i]) == msg.sender, "Not owner of token");
        newRobiBoost += _tokens[tokenId[i]].robiBoost;
        _burn(tokenId[i]);
        // _tokens[tokenId[i]] SHOULD BE DELETED
    }
    newRobiBoost = newRobiBoost + (newRobiBoost * _levelUpPercent) / 100;
    _lastTokenId += 1;
    uint256 newTokenId = _lastTokenId;
    _tokens[newTokenId].robiBoost = newRobiBoost;
    _tokens[newTokenId].createTimestamp = block.timestamp;
    _tokens[newTokenId].level = level;
    _safeMint(msg.sender, newTokenId);
}
```

UPDATES

- *July 20, 2022*: This issue has been acknowledged by Winery team.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Jul 20, 2022*	Public Report	Verichains Lab
1.1	*Jul 25, 2022*	Public Report	Verichains Lab

Table 3. Report versions history