verichains

SECURITY AUDIT OF

UNIZEN SMART CONTRACT

Public Report

Dec 29, 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 was prepared by Verichains Lab on Dec 29, 2022. We would like to thank the Unizen 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 Unizen 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 2 vulnerability issues in the contract code.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About Unizen Smart Contract

The Unizen Ecosystem is housing and aggregating trades across trusted first- and third-party exchange modules to enable a ZEN state of mind for traders. Unizen is currently powered by the below liquidity providers but is continously adding on more sources of liquidity.

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of the Unizen Smart Contract. It was conducted on commit `e3774396556f1f2f480ce96f7c7fa657787d45d9` from git repository *https://github.com/unizen-io/unizen-trade-aggregator*.

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

SHA256 Sum	File
04bba11cb43a93511e6a995b56194d78144c8a5dfdb6a06fd6fae9146a63aa69	helpers/PermitApprove.sol
7131cf3478e3ad5aab51aa50e322bb6ad81530cecb64ebd15b4989d368e2339f	helpers/EthReceiver.sol
a749ffe55f432b6790f5e6be977df4e8b07df05f11854768039865a86b0c07ba	UnizenDexAggr.sol
69ca2aefcce1141ab3a166748003770bc8eb6c945b272bd79707ffea79305c58	dependencies/Controller.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 were 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 Unizen Smart Contract was written in `Solidity` language, with the required version to be `^0.8.12`. The source code was written based on OpenZeppelin's library.

2.1.1. PermitApprove.sol

This contract allows a user to sign an approve transaction off-chain producing a signature that anyone could use and submits to the blockchain. It's a fundamental first step towards solving the gas payment issue and also removes the user-unfriendly 2-step process of sending `approve` and later `transferFrom`.

2.1.2. EthReceiver.sol

This contract's primary purpose is to only receive native token from other contracts.

2.1.3. Controller.sol

This contract extends `OwnableUpgradeable` and `PausableUpgradeable`. With `OwnableUpgradeable`, the contract owner is contract deployer with the logic in the `initialize` function, he can transfer ownership to another address at any time. He can pause/unpause contract by using `PausableUpgradeable` contract.

Whitelist decentralized exchanges(DEX) can be enabled or disabled by the contract owner.

2.1.4. UnizenDexAggr.sol

This contract extends `PermitApprove`, `EthReceiver`, `Controller` and `ReentrancyGuardUpgradeable` contracts.

The contract includes the following main functions:

- `swapLZ`: make a bridge between tokens on different chains. The contract will use LayerZero to bridge to another chain, swap the user's desired token to the stable coin of this chain, and then swap the stable coin of the new chain to the user's desired token.
- `swapSTG`: make a bridge between the pools of two chains, with the contract taking the transaction fee and sending it to the treasury address.
- `swap`, `swapExactOut`: make a swap between two tokens, the contract will take the transaction fee and send it to treasury address. User can only use both of these functions when the contract is not paused.
- The contract owner can deposit stable coin to contract, he can also withdraw all amount of any token.

All swap orders are processed on whitelisted exchanges, and VIP users, who are defined in the VipOracle contract, may pay a different fee than other users. Any remaining tokens (including native tokens and native fees) will be returned to the user.

2.2. Findings

During the audit process, the audit team found 2 vulnerabilities in the given version of Unizen Smart Contract.

2.2.1. UnizenDexAggr.sol - Underflow in when bridge with `swapLZ()` and `sgReceive()` function CRITICAL

Because contract is not check `dstCalls` paramater, attacker can using `unchecked` keyword to make a underflow and steal stable token.

Process:

- Step1: attacker bridges stable token from chain A to other token in chain B (that mean `calls.length` == 0), and enters any number of `dstCalls`
- Step2: if the `safeApprove()` function only approves an amount of tokens equal to the `amount` variable, then `targetExchange` can only swap less than or equal to `amount`, but instead, the `safeApprove()` function approves an amount equal to `dstCalls[i].amount`, a number that the attacker can controllable, he can swap as much as he wants.(ex: `amount` = 100, total `dstCalls[i].amount` = 1.000.000)
- Step3: the value of the variable `diff` will now be:`diff` = balanceStableAfter + amount - balanceStableBefore= (balanceStableBefore - 1.000.000) + 100 - balanceStableBefore= -999.900
- Step 4: with `unchecked` keyword, the negative number -999.900 will be equal to (2^256 - 999.900)

```
function swapLZ(
        CrossChainSwapLz memory swapInfo,
        SwapCall[] memory calls,
        SwapCall[] memory dstCalls
    ) external payable nonReentrant {
        require(chainStableDecimal[swapInfo.dstChain] != 0, "Not-enable-yet");
        require(destAddr[swapInfo.dstChain] != address(0), "Invalid-address");
        uint256 balanceStableBefore = IERC20(stable).balanceOf(address(this));
        if (!swapInfo.isFromNative) {
            IERC20(swapInfo.srcToken).safeTransferFrom(
                msg.sender,
                address(this),
                swapInfo.amount
            );
        } else {
            require(
```

```
            msg.value >= swapInfo.amount + swapInfo.nativeFee,
            "Invalid-amount"
        );
    }

    for (uint8 i = 0; i < calls.length; i++) {
        require(calls[i].amount != 0, "Invalid-trade-amount");
        require(
            isWhiteListedDex(calls[i].targetExchange),
            "Not-verified-dex"
        );
        swapInfo.amount = swapInfo.amount - calls[i].amount;
        if (!swapInfo.isFromNative) {
            IERC20(swapInfo.srcToken).safeApprove(
                calls[i].targetExchange,
                0
            );
            IERC20(swapInfo.srcToken).safeApprove(
                calls[i].targetExchange,
                calls[i].amount
            );
        }
        {
            bool success;
            if (swapInfo.isFromNative) {
                success = _executeTrade(
                    calls[i].targetExchange,
                    calls[i].amount,
                    calls[i].data
                );
            } else {
                success = _executeTrade(
                    calls[i].targetExchange,
                    0,
                    calls[i].data
                );
            }
            require(success, "Call-Failed");
        }
    }

    if (swapInfo.srcToken != stable && swapInfo.amount > 0) {
        if (swapInfo.isFromNative) {
            swapInfo.nativeFee += swapInfo.amount;
        } else {
            // return diff amount
            IERC20(swapInfo.srcToken).safeTransfer(
                msg.sender,
                swapInfo.amount
            );
        }
```

```
        }

        uint256 bridgeAmount = IERC20(stable).balanceOf(address(this)) -
            balanceStableBefore;
        require(bridgeAmount > 0, "Something-went-wrong");
        bytes memory payload = abi.encode(
            (bridgeAmount * 10**chainStableDecimal[swapInfo.dstChain]) /
                10**stableDecimal,
            msg.sender,
            dstCalls
        );
        ILayerZeroEndpoint(layerZeroEndpoint).send{value: swapInfo.nativeFee}(
            swapInfo.dstChain,
            abi.encodePacked(destAddr[swapInfo.dstChain], address(this)),
            payload,
            payable(msg.sender),
            address(0),
            swapInfo.adapterParams
        );
        emit CrossChainSwapped(swapInfo.dstChain, msg.sender, bridgeAmount);
    }

    function lzReceive(
        uint16 _srcChainId,
        bytes memory _fromAddress,
        uint64, /*_nonce*/
        bytes memory _payload
    ) external override {
        require(msg.sender == address(layerZeroEndpoint), "Only-lz-endpoint"); //
boilerplate! lzReceive must be called by the endpoint for security
        bytes memory trustedRemote = trustedRemoteLookup[_srcChainId];
        require(
            _fromAddress.length == trustedRemote.length &&
                keccak256(_fromAddress) == keccak256(trustedRemote),
            "Only-trusted-remote"
        );
        //    bytes memory payload = abi.encode(
        //         (bridgeAmount * 10**chainStableDecimal[swapInfo.dstChain]) /
        //             10**stableDecimal,
        //         msg.sender,
        //         dstCalls
        //     );
        (uint256 amount, address user, SwapCall[] memory dstCalls) = abi.decode(
            _payload,
            (uint256, address, SwapCall[])
        );

        if (dstCalls.length == 0) {
            // user doesnt want to swap, want to take stable
            IERC20(stable).safeTransfer(user, amount);
            return;
```

```
        }
        uint256 balanceStableBefore = IERC20(stable).balanceOf(address(this));
        for (uint8 i = 0; i < dstCalls.length; i++) {
            require(dstCalls[i].amount != 0, "Invalid-trade-amount");
            require(
                isWhiteListedDex(dstCalls[i].targetExchange),
                "Not-verified-dex"
            );
            IERC20(stable).safeApprove(dstCalls[i].targetExchange, 0);
            IERC20(stable).safeApprove(
                dstCalls[i].targetExchange,
                dstCalls[i].amount
            );
            _executeTrade(dstCalls[i].targetExchange, 0, dstCalls[i].data);
        }

        unchecked {
            uint256 diff = IERC20(stable).balanceOf(address(this)) +
                amount -
                balanceStableBefore;
            require(diff >= 0, "Lost-token");
            if (diff > 0) {
                IERC20(stable).safeTransfer(user, diff);
            }
        }

        emit CrossChainSwapped(_srcChainId, user, amount);
    }
```

RECOMMENDATION

Remove unchecked:

```
function lzReceive(
        uint16 _srcChainId,
        bytes memory _fromAddress,
        uint64, /*_nonce*/
        bytes memory _payload
    ) external override {
        ...
        (uint256 amount, address user, SwapCall[] memory dstCalls) = abi.decode(
            _payload,
            (uint256, address, SwapCall[])
        );

        if (dstCalls.length == 0) {
            // user doesnt want to swap, want to take stable
            IERC20(stable).safeTransfer(user, amount);
            return;
        }
        uint256 balanceStableBefore = IERC20(stable).balanceOf(address(this));
```

```
        ...

        uint256 diff = IERC20(stable).balanceOf(address(this)) +
            amount -
            balanceStableBefore;
        if (diff > 0) {
            IERC20(stable).safeTransfer(user, diff);
        }
        emit CrossChainSwapped(_srcChainId, user, amount);
    }
//-------------------------------------------------------------------------------------------
-----------------------------
    // sgReceive() - the destination contract must implement this function to receive the
tokens and payload
    function sgReceive(
        uint16 _chainId,
        bytes memory _srcAddress,
        uint256 _nonce,
        address _token,
        uint256 amountLD,
        bytes memory payload
    ) external override {
        require(msg.sender == address(stargateRouter), "Only-Stargate-Router");
        (address user, SwapCall[] memory dstCalls, bool isAmountOut) = abi
            .decode(payload, (address, SwapCall[], bool));
        if (dstCalls.length == 0) {
            // user doesnt want to swap, want to take stable
            IERC20(_token).safeTransfer(user, amountLD);
            return;
        }
        uint256 balanceStableBefore = IERC20(_token).balanceOf(address(this));
        ...

        uint256 diff = IERC20(_token).balanceOf(address(this)) +
            amountLD -
            balanceStableBefore;
        if (diff > 0) {
            IERC20(stable).safeTransfer(user, diff);
        }

        emit CrossChainSwapped(_chainId, user, amountLD);
    }
```

UPDATES

- *2022-12-29*: This issue has been acknowledged and fixed by the Unizen team.

2.2.2. UnizenDexAggr.sol - Centralization Related Risks HIGH

The `DEFAULT_ADMIN_ROLE` role has admin permission so any compromise to the `admin` account may allow hackers to exploit the system.

RECOMMENDATION

We should use Multi sign or DAO mechanism to manage the admin account. In additional, we should add a delay mechanism (eg 48 hours latency) for awareness on privileged operations.

UPDATES

- *2022-12-29*: This issue has been acknowledged and fixed by the Unizen team.

2.2.3. UnizenDexAggr.sol - Missing event in `lzReceive()` and `sgReceive()` functions INFORMATIVE

In both functions, when length of `dstCalls[]` equal to 0, functions will return without emit event `CrossChainSwapped()`.

RECOMMENDATION

Add event `CrossChainSwapped()` to `lzReceive()` and `sgReceive()` functions.

UPDATES

- *2022-12-29*: This issue has been acknowledged and fixed by the Unizen team.

2.2.4. UnizenDexAggr.sol - Lack of necessary events INFORMATIVE

Events are used to inform external users that something happened on the blockchain. Smart contracts themselves cannot listen to any events.

All information in the blockchain is public and any actions can be found by looking into the transactions close enough but events are a shortcut to ease the development of outside systems in cooperation with smart contracts.

RECOMMENDATION

Add events for `set` functions like: `setDstGas()`, `setDestAddr()`... and emit them.

UPDATES

- *2022-12-29*: This issue has been acknowledged by the Unizen team.

2.2.5. UnizenDexAggr.sol - Condition inside `require()` function always returns true INFORMATIVE

Because integer value in solidity are always positive, if there is a subtraction that produces negative numbers, they will be reversed unless they are set in the unchecked keyword (and underflow).

Positions: Line 636 and 768.

RECOMMENDATION

Remove the `require()` function in the above positions.

UPDATES

- *2022-12-29*: This issue has been acknowledged by the Unizen team.

APPENDIX

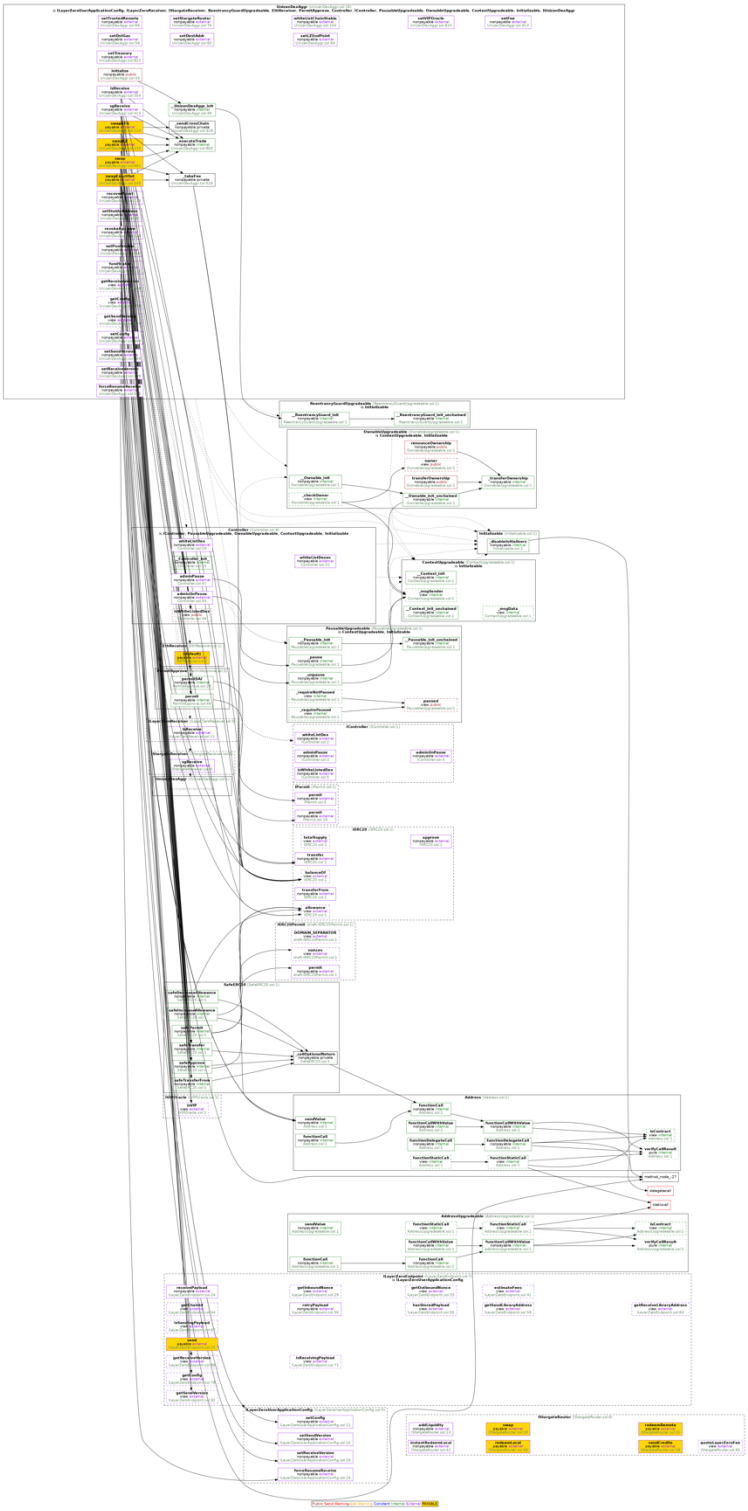

Image 1. Unizen Smart Contract call graph

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Dec 23, 2022*	Private Report	Verichains Lab
1.1	*Dec 29, 2022*	Public Report	Verichains Lab

Table 2. Report versions history