verichains

SECURITY AUDIT OF

UNIZEN DEX AGGREGATOR UPDATES

Public Report

Jun 27, 2024

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

ABBREVIATIONS

Name	Description
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.
Ethereum	An opensource platform based on blockchain technology to create and distribute smart contracts and decentralized applications.
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.
Stargate	Stargate is a community-driven organization building the first fully composable native asset bridge, and the first dApp built on LayerZero. Stargate's vision is to make cross-chain liquidity transfer a seamless, single transaction process.
Layerzero	LayerZero is an interoperability protocol that connects blockchains, allowing developers to build seamless omnichain applications, tokens, and experiences. The protocol relies on immutable on-chain endpoints, a configurable Security Stack, and a permissionless set of Executors to transfer censorship-resistant messages between chains.
Wormhole	Wormhole is a generic message passing protocol that enables communication between blockchains.

Name	Description
Meson	Meson is the faster and safer way to execute low-cost, zero-slippage universal cross-chain swaps across all leading blockchains and layer-2 rollups.
DeBridge	deBridge is a high-performance and secure interoperability layer for Web3 that enables decentralized transfers of arbitrary messages and value between blockchains.
ThorChain	THORChain is a decentralized cross-chain liquidity protocol that utilizes the Tendermint consensus engine, Cosmos-SDK state machine, and GG20 Threshold Signature Scheme (TSS). It operates as an independent Layer 1 cross-chain decentralized exchange (DEX) built on the Cosmos SDK.

EXECUTIVE SUMMARY

This Security Audit Report was prepared by Verichains Lab on Jun 27, 2024. 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 Dex Aggregator Updates. 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 identified some vulnerable issues in the contract code.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About Unizen Dex Aggregator Updates

The Unizen Ecosystem is housing and aggregating trades across trusted first and third-party exchange modules. Unizen is currently powered by the below liquidity providers but is continuously 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 Dex Aggregator Updates. It was conducted on commit `d92933baf2ec9f03ab6a77698496e787b7e32d31` from git repository link: *https://github.com/unizen-io/unizen-trade-aggregator*

The patch was applied to commit `70ed4e6bd1baf6ca882b9acc2d8b61900e00565e` from the above git repository.

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

Unizen acknowledges that the security services provided by Verichains, are conducted to the best of their professional abilities but cannot guarantee 100% coverage of all security vulnerabilities. Unizen understands and accepts that despite rigorous auditing, certain vulnerabilities may remain undetected. Therefore, Unizen agrees that Verichains shall not be held responsible or liable, and shall not be charged for any hacking incidents that occur due to security vulnerabilities not identified during the audit process.

1.5. Acceptance Minute

This final report served by Verichains to the Unizen will be considered an Acceptance Minute. Within 7 days, if no any further responses or reports is received from the Unizen, the final report will be considered fully accepted by the Unizen without the signature.

2. AUDIT RESULT

2.1. Overview

The Unizen Dex Aggregator Updates is developed in the `Solidity` language, adhering to version `^0.8.12` requirements, and built upon the OpenZeppelin library.

The updated contracts version contains `UnizenDexAggr`, `UnizenDexAggrETH`, `UnizenDexAggrV3Base` and `UnizenDexAggrV3` share identical primary functionalities. These include aggregating swaps, facilitating cross-chain swaps, and enabling integrators to earn fees.

This update version that enhances flexibility and functionality by adding support for ThorChain, Meson, and Wormhole Routers. The contracts are designed to be upgradable, allowing for future improvements and updates.

2.1.1. UnizenDexAggr and UnizenDexAggrETH Contracts

In the `UnizenDexAggr` and `UnizenDexAggrETH` contracts, a notable addition is the integration of the `ThorChain` Router. This involves a setter function that allows only the contract owner to set the ThorChain Router address. Additionally, the `swapTC` function has been introduced to facilitate swaps via the ThorChain Router, enabling decentralized cross-chain swaps.

Furthermore, the StarGate Router integration has been enhanced by allowing users to pass a fee number as a parameter in the `swapSTG` and `swapTC` functions, providing greater flexibility and optimizing transaction costs.

2.1.2. UnizenDexAggrV3Base Contract

The `UnizenDexAggrV3Base` contract extends several important modules, including `OwnableUpgradeable`, `PausableUpgradeable`, `IExternalCallExecutor`, `EthReceiver3`, and `ReentrancyGuardUpgradeable`. These extensions provide essential features such as ownership management, the ability to pause the contract in emergencies, external contract call execution, ETH transfer handling, and protection against reentrancy attacks.

Additionally, this contract supports cross-chain asset swaps via `deBridge` and `Meson`, enhancing its interoperability across different blockchain networks.

2.1.3. UnizenDexAggrV3 Contract

The `UnizenDexAggrV3` contract supports the Wormhole Router, enabling robust cross-chain swaps between different blockchain networks. In this contract, tokens are swapped to stable tokens on the source chain and then back to the desired tokens on the destination chain. This functionality enhances interoperability and ensures seamless asset transfers across different chains.

2.2. Findings

During the audit process, the audit team identified some vulnerable issues in the contract code.

#	Issue	Severity	Status
1	Does not return when executing swap at destination chain fails	HIGH	Fixed
2	Does not restrict the range of fee BPS	INFORMATIVE	Acknowledged
3	Missing check for length of parameters as array and emit event	INFORMATIVE	Acknowledged

2.2.1. [HIGH] Does not return when executing swap at destination chain fails

Affected file: UnizenDexAggrV3Wormhole.sol:107-155

The `_onWormholeUsdcReceived` function is responsible for receiving USDC tokens from the Wormhole bridge, executing a swap on the destination chain, and then sending the swapped tokens to the receiver. The workflow of the function is as follows:

- Attempt to swap `stableToken` into `tokenOut` using the `executeSwapDstChain` function.
- If the swap fails:
  - Transfer the `stableToken` to the receiver.
  - Emit a `WormholeDstChainSwapFailed` event.
- If the swap succeeds:
  - Calculate the amount of `tokenOut` received.
  - Send the `tokenOut` to the receiver.

The issue arises in step 2. When the swap fails, the function does not immediately return and exit. Instead, the transaction continues, potentially leading to unintended behavior or loss of funds. This is an unexpected behavior and can result in significant security and financial risks.

```
function _onWormholeUsdcReceived(
    bytes memory _payload,
    uint256 _stableAmount,
    bytes32 _srcAddress,
    uint16 _srcChain,
    bytes32 // deliveryHash
) internal override onlyWormholeRelayer {
    if (registeredSenders[_srcChain] != _srcAddress) {
        revert NotRegisteredContract();
    }
    (
        address receiver,
        address tokenOut,
        uint256 minTokenOutQuote,
        string memory uuid,
        SwapCall[] memory calls
    ) = abi.decode(_payload, (address, address, uint256, string, SwapCall[]));

    require(receiver != address(0), "Invalid-receiver");

    IERC20 stableToken = IERC20(WormholeUSDC);
    uint256 stableBalanceBefore = stableToken.balanceOf(address(this));
    uint256 tokenOutBalanceBefore = _getBalance(tokenOut);

    try this.executeSwapDstChain(address(stableToken), _stableAmount, calls) {} catch {
        stableToken.safeTransfer(receiver, _stableAmount);
        emit WormholeDstChainSwapFailed(address(stableToken), _stableAmount, receiver);
    }

    uint256 tokenOutAmt = _getBalance(tokenOut) - tokenOutBalanceBefore;
    if (tokenOutAmt > minTokenOutQuote) {
        tokenOutAmt = tokenOutAmt - _takePSFee(tokenOut, (tokenOutAmt - minTokenOutQuote),
uuid, 0);
    }
    if (tokenOut == address(0)) {
        payable(receiver).sendValue(tokenOutAmt);
    } else {
        IERC20(tokenOut).safeTransfer(receiver, tokenOutAmt);
    }
    emit WormholeDstChainSwapSuccess(tokenOut, tokenOutAmt, receiver);

    tokenOutAmt = stableBalanceBefore - stableToken.balanceOf(address(this));
    if (_stableAmount > tokenOutAmt) {
        stableToken.safeTransfer(receiver, _stableAmount - tokenOutAmt);
    }
}
```

RECOMMENDATION

To prevent the function from continuing execution after a failed swap, an explicit return statement should be added immediately after handling the failed swap scenario. This ensures that the function exits and no further actions are taken.

UPDATES

27 Jun 2024, The Unizen team has acknowledged the issue and has implemented the recommended changes.

2.2.2. [INFORMATIVE] Does not restrict the range of fee BPS

Affected file: UnizenDexAggrV3.sol:230-260

The `_calculateTakeAmount` function calculates the amount of tokens to be taken after applying a fee in basis points (bps). The function currently does not restrict the range of the fee bps, which can lead to unexpected results. For instance, if the fee bps is greater than `BPS_DENOMINATOR`, the result will be negative and revert the transaction. Similarly, if the fee bps is equal to `BPS_DENOMINATOR`, the result will be zero, effectively nullifying the transaction.

```
function _calculateTakeAmount(
    uint256 giveAmount,
    bool useNativeCrossChain,
    uint256 srcTokenOutDecimals,
    uint256 dstTokenInDecimals,
    uint256 dlnProtocolFeeBps,
    uint256 dlnTakerFeeBps,
    uint256 operatingFee
) internal pure returns (uint256 takeAmt) {
    // src chain: minus DlnProtocolFee 0.04%
    if (dlnProtocolFeeBps > 0) {
        takeAmt = (giveAmount * (BPS_DENOMINATOR - dlnProtocolFeeBps)) / BPS_DENOMINATOR;
    }

    // cross chain: change token decimals when move from src chain to dst chain
    if (!useNativeCrossChain && srcTokenOutDecimals != dstTokenInDecimals) {
        if (dstTokenInDecimals > srcTokenOutDecimals) {
            takeAmt = takeAmt * 10 ** (dstTokenInDecimals - srcTokenOutDecimals);
        } else {
            takeAmt = takeAmt / 10 ** (srcTokenOutDecimals - dstTokenInDecimals);
        }
    }
    // dst chain: minus TakerMargin 0.04%
    if (dlnTakerFeeBps > 0) {
        takeAmt = (takeAmt * (BPS_DENOMINATOR - dlnTakerFeeBps)) / BPS_DENOMINATOR;
    }
    // dst chain: minus EstimatedOperatingExpenses
    takeAmt -= operatingFee;
}
```

RECOMMENDATION

To prevent unexpected results, it is recommended to add a check that ensures the fee bps is within a valid range (0 to BPS_DENOMINATOR - 1). This will ensure that the fee bps does not exceed the total basis points denominator, preventing negative or zero results.

UPDATES

27 Jun 2024, The Unizen team has acknowledged the issues.

2.2.3. [INFORMATIVE] Missing check for length of parameters as array and emit event

Affected file: UnizenDexAggrV3.sol:86-113

The function `setDlnStable` and `setMesonStable` do not check the length of the input arrays, which could lead to out-of-bound errors. Additionally, these setter functions do not emit events, which is essential for monitoring and logging changes.

```
function setUnizenController(address _controller) external onlyOwner {
    unizenController = Controller(_controller);
}

function setFeeClaimer(address feeClaimerAddr) external onlyOwner {
    feeClaimer = feeClaimerAddr;
}

function setDLNConfigution(IDlnSource _dlnSource, address _dlnAdapter) external onlyOwner {
    dlnSource = _dlnSource;
    dlnAdapter = _dlnAdapter;
}

function setDlnStable(address[] calldata stableTokens, bool[] calldata isActive) external
onlyOwner {
    for (uint8 i = 0; i < stableTokens.length; i++) {
        isDlnStable[stableTokens[i]] = isActive[i];
    }
}

function setMesonConfigution(IMeson _meson) external onlyOwner {
    meson = _meson;
}

function setMesonStable(address[] calldata stableTokens, bool[] calldata isActive) external
onlyOwner {
    for (uint8 i = 0; i < stableTokens.length; i++) {
        isMesonStable[stableTokens[i]] = isActive[i];
    }
}
```

RECOMMENDATION

We suggest adding a check to ensure the lengths of the input arrays are equal to prevent out-of-bound errors. Emitting events in the setter functions to facilitate monitoring and logging.

UPDATES

27 Jun 2024, The Unizen team has acknowledged the issues.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Jun 24, 2024*	Public Report	Verichains Lab
1.1	*Jun 27, 2024*	Public Report	Verichains Lab

Table 2. Report versions history