BEOSIN
Blockchain Security

unizen-trade-aggregator

Smart Contract Security Audit

No. 202406211146

Jun 21th, 2024

WWW.BEOSIN.COM

Contents

Summary of Audit Results

After auditing, 1 Medium and 4 Low items were identified in the unizen-trade-aggregator project. Specific audit details will be presented in the Findings section. Users should pay attention to the following aspects when interacting with this project:

- **Risk Description:**

.Some key variable restrictions for the exchange operations, such as the project's service fee `sharePercent` and the exchange parameters `SwapCall`, are guaranteed by the project team in the off-chain system. Users should therefore be cautious when interacting directly with the on-chain contract.

Business overview:

The unizen-trade-aggregator is a cross-chain trading aggregation project that supports multiple cross-chain bridge protocols, including deBridge, Meson, and Wormhole, among others. The project's cross-chain exchange business logic is mainly divided into three parts:

Users can specify a series of exchange operations on the source chain through the `SwapCall` parameter to ultimately obtain the `srcTokenOut` token that needs to be transferred across the chain.

The project calls the third-party cross-chain bridge API to perform the cross-chain operation. The specific third-party cross-chain interface to be called is determined by the off-chain routing algorithm based on the user's requirements.

After the target chain receives the cross-chain tokens, it also follows the user's specified `SwapCall` parameters to perform a series of exchange operations on the target chain. Finally, the exchanged target tokens and the remaining cross-chain tokens are all returned to the user.

In summary, the project enables users to conduct cross-chain token exchanges by leveraging multiple cross-chain bridge protocols, with the exchange logic split into three main parts to facilitate the overall cross-chain trading process.

1 Overview

1.1 Project Overview

Project Name	unizen-trade-aggregator
Project Language	Solidity
Platform	Ethereum, Base, Arbitrum, Optimism, BSC, AVAX, Fantom, Polygon
Code base	https://github.com/unizen-io/unizen-trade-aggregator/tree/dev
Commit hash	d92933baf2ec9f03ab6a77698496e787b7e32d31 70ed4e6bd1baf6ca882b9acc2d8b61900e00565e

1.2 Audit Overview

Audit work duration: Jun 14, 2024 – Jun 21, 2024

Audit team: Beosin Security Team

1.3 Audit Method

The audit methods are as follows:

1. Formal Verification

Formal verification is a technique that uses property-based approaches for testing and verification. Property specifications define a set of rules using Beosin's library of security expert rules. These rules call into the contracts under analysis and make various assertions about their behavior. The rules of the specification play a crucial role in the analysis. If the rule is violated, a concrete test case is provided to demonstrate the violation.

2. Manual Review

Using manual auditing methods, the code is read line by line to identify potential security issues. This ensures that the contract's execution logic aligns with the client's specifications and intentions, thereby safeguarding the accuracy of the contract's business logic.

The manual audit is divided into three groups to cover the entire auditing process:

The Basic Testing Group is primarily responsible for interpreting the project's code and conducting comprehensive functional testing.

The Simulated Attack Group is responsible for analyzing the audited project based on the collected historical audit vulnerability database and security incident attack models. They identify potential attack vectors and collaborate with the Basic Testing Group to conduct simulated attack tests.

The Expert Analysis Group is responsible for analyzing the overall project design, interactions with third parties, and security risks in the on-chain operational environment. They also conduct a review of the entire audit findings.

3. Static Analysis

Static analysis is a method of examining code during compilation or static analysis to detect issues. Beosin-VaaS can detect more than 100 common smart contract vulnerabilities through static analysis, such as reentrancy and block parameter dependency. It allows early and efficient discovery of problems to improve code quality and security.

2 Findings

Index	Risk description	Severity level	Status
unizen-trade-aggregator-01	The totalDstAmount is not verified	Medium	Fixed
unizen-trade-aggregator-02	The sharePercent is not verified	Low	Acknowledged
unizen-trade-aggregator-03	The loss of the tempAmount value may lead to loss of user funds	Low	Acknowledged
unizen-trade-aggregator-04	The vault address is not being verified and not included in event	Low	Fixed
unizen-trade-aggregator-05	The missing return statement in the try-catch block	Low	Fixed

Finding Details:

[unizen-trade-aggregator-01] The totalDstAmount is not verified

Severity Level	Medium
Type	Business Security
Lines	UnizenDexAggr.sol#L523-533 UnizenDexAggrETH.sol#L510-531
Description	In the `swapTC` function, the `SwapTC` parameters specify an amountOutMin, which is the expected minimum amount out. However, the actual tokens received after the swap, totalDstAmount, are not verified against this value. Additionally, the amountOutMin is not used in the subsequent logic. This could result in users receiving an amount less than their expected amount in the actual exchange.
Recommendation	It is recommended to add a new check that the totalDstAmount must be greater than or equal to amountOutMin.
Status	**Fixed.** The project party added totalDstAmount check for this function.

```
    uint256 balanceDstAfter = dstToken.balanceOf(address(this));
    uint256 totalDstAmount = balanceDstAfter - balanceDstBefore;
    dstToken.safeApprove(tcRouter, 0);
    dstToken.safeApprove(tcRouter, totalDstAmount);
    ITcRouter(tcRouter).depositWithExpiry(
        payable(info.vault),
        info.dstToken,
        totalDstAmount,
        info.memo,
        info.deadline
    );
```

[unizen-trade-aggregator-02] The sharePercent is not verified

Severity Level	Low
Type	Business Security
Lines	UnizenDexAggrV3.sol#L598-622
Description	1) In functions like `swapWormhole` and `swapDB`, the fees are passed in as parameters. They only verify that the feePercent is greater than 0, but do not specify the sharePercent for the Unizen dividend. Users could set the sharePercent to 0, which may not be reasonable. 2) In the `_takeIntegratorFee` function, the calculation of the user's fees does not limit the maximum value of feePercent and sharePercent in the contract. When sharePercent is greater than 10000, it is possible for the user to lose more funds than the total fees.

```
function _takeIntegratorFee(
    string memory uuid,
    bool isETHTrade,
    address token,
    uint256 amount,
    uint256 feePercent,
    uint256 sharePercent
) internal returns (uint256 totalFee) {
    uint256 unizenFee;
    address integratorAddrs =
IUnizenDexAggr(address(unizenController)).integratorAddrs(uuid);
    totalFee = (amount * feePercent) / 10000;
    //Collect integrator unizen shared fee
    if (sharePercent > 0) {
        unizenFee = (totalFee * sharePercent) / 10000;
        unizenFeeEarned[address(token)] =
unizenFeeEarned[address(token)] + unizenFee;
    }
    if (isETHTrade) {
        payable(integratorAddrs).sendValue(totalFee - unizenFee);
    } else {
        IERC20(token).safeTransfer(integratorAddrs, totalFee -
unizenFee);
    }
```

```
    return totalFee;
}
```

Recommendation	It is recommended to the project's transaction fees are set by the contract to a fixed percentage, rather than being passed in by the user.
Status	**Acknowledged.** The project party stated that sharePercent could be 0 as well, and they verify it in backend system, and when conduct a trade, if the fee is super high, so amountOut will very low, trade will revert. And fee will send to their whitelisted address (their client).

[unizen-trade-aggregator-03] Invalid tempAmount value may lead to loss of user funds

Severity Level	Low
Type	Business Security
Lines	UnizenDexAggrV3Base.sol#L486-537
Description	The path parameters for the swap need to be strictly controlled, otherwise it may lead to errors. This is because the function does not record the temporary amount (tempAmount) of the buyToken after each SwapCall execution, which may result in the loss of the temporary amount data generated during the intermediate exchange. When the next round of SwapCall is executed for exchange, the token quantity may have incorrect constraints.

```
function _swap(address _srcToken, uint256 _srcAmount, SwapCall[] memory
calls) internal {
        require(calls[0].sellToken == _srcToken, "Invalid-token");
        uint256 tempAmount;
        uint256 totalSrcAmount;
        IERC20 srcToken;
        for (uint8 i = 0; i < calls.length; ) {
            require(unizenController.isWhiteListedDex(calls[i].target
Exchange), "Not-verified-dex");
            if (calls[i].sellToken == _srcToken) {
                // if trade from source token
                // if not split trade, it will be calls[0]
                // if split trade, we count total amount of souce token
we split into routes
                totalSrcAmount += calls[i].amount;
                require(totalSrcAmount <= _srcAmount,
"Invalid-amount-to-sell");
            }
            if (calls[i].sellToken == address(0)) {
                // trade Ethereum, it will be for trade from source token
as native, only trade single-chain as if trade dstChain, no native trade
                tempAmount = _executeTrade(
                    calls[i].targetExchange,
                    address(0),
                    calls[i].buyToken,
                    calls[i].amount,
```

```
                    calls[i].amount,
                    calls[i].data
                );
            } else {
                // trade ERC20
                srcToken = IERC20(calls[i].sellToken);
                srcToken.safeApprove(calls[i].targetExchange, 0);
                srcToken.safeApprove(calls[i].targetExchange,
calls[i].amount);

                tempAmount = _executeTrade(
                    calls[i].targetExchange,
                    calls[i].sellToken,
                    calls[i].buyToken,
                    calls[i].amount,
                    0,
                    calls[i].data
                );
            }
            // Here we have to check the tempAmount we got from the trade
is higher than sell amount of next, else that mean we got steal fund
            // But if there is split trade with split source token into
multi routes, we dont check because first trade of route is trade from
source token
            // And we already check totalSrcAmount is under total amount
we got
            if (i != calls.length - 1 && calls[i + 1].sellToken !=
_srcToken) {
                require(tempAmount >= calls[i + 1].amount,
"Steal-fund");
                // the next buy token must be the current sell token
                require(calls[i].buyToken == calls[i + 1].sellToken,
"Steal-funds");
            }
            unchecked {
                ++i;
            }
        }
    }
```

Recommendation	It is recommended to verify that the cross-chain amount is greater than 0 before sending the cross-chain message.
Status	**Acknowledged.** The project party stated that the limitation restricted in their off-chain system.

[unizen-trade-aggregator-04] The vault address is not within event

Severity Level	Low
Type	Business Security
Lines	UnizenDexAggr.sol#L490-523
Description	In the `swapTC` function, the vault parameter is not set to a whitelist. Attackers could pass in a vault address under their control. The project team should confirm whether there is a whitelist for the vault. Additionally, the CrossChainUTXO event lacks the vault parameter, allowing attackers to trigger this event using a vault address they control. If the off-chain portion of the project relies on this event, there may be a security vulnerability.

```
if (isETHTrade) {
    // deposit directly to ThorchainRouter
    ITcRouter(tcRouter).depositWithExpiry{value: amount}(
        payable(info.vault),
        address(0),
        amount,
        info.memo,
        info.deadline
    );
    emit CrossChainUTXO(address(0), amount, info.apiId);
    return;
}
uint256 balanceDstBefore = dstToken.balanceOf(address(this));
_swap(info.srcToken, amount, calls, false);
uint256 balanceDstAfter = dstToken.balanceOf(address(this));
uint256 totalDstAmount = balanceDstAfter - balanceDstBefore;
dstToken.safeApprove(tcRouter, 0);
dstToken.safeApprove(tcRouter, totalDstAmount);
ITcRouter(tcRouter).depositWithExpiry(
    payable(info.vault),
    info.dstToken,
    totalDstAmount,
    info.memo,
    info.deadline
);
emit CrossChainUTXO(info.dstToken, amount, info.apiId);
```

Recommendation	It is recommended adding the key parameter vault in the event CrossChainUTXO, and it is necessary to confirm that the SDK or off-chain program should not have relevant logic based on this event.
Status	**Fixed.** The project team has added a vault variable in the CrossChainUTXO event, and has stated that they will use this event reasonably in their off-chain systems.

[unizen-trade-aggregator-05] The missing return statement in the try-catch block

Severity Level	Low
Type	Business Security
Lines	UnizenDexAggr.sol#L132-135
Description	In the _onWormholeUsdcReceived function,the catch code block is missing a return statement, which means the logic after it will continue to execute,potentially leading to unnecessary gas consumption.
Recommendation	It is recommended adding the return statement to optimize gas.
Status	**Fixed.**

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

3 Appendix

3.1 Vulnerability Assessment Metrics and Status in Smart Contracts

3.1.1 Metrics

In order to objectively assess the severity level of vulnerabilities in blockchain systems, this report provides detailed assessment metrics for security vulnerabilities in smart contracts with reference to CVSS 3.1(Common Vulnerability Scoring System Ver 3.1).

According to the severity level of vulnerability, the vulnerabilities are classified into four levels: "critical", "high", "medium" and "low". It mainly relies on the degree of impact and likelihood of exploitation of the vulnerability, supplemented by other comprehensive factors to determine of the severity level.

Likelihood \ Impact	Severe	High	Medium	Low
Probable	Critical	High	Medium	Low
Possible	High	Medium	Medium	Low
Unlikely	Medium	Medium	Low	Info
Rare	Low	Low	Info	Info

3.1.2 Degree of impact

- **Severe**

Severe impact generally refers to the vulnerability can have a serious impact on the confidentiality, integrity, availability of smart contracts or their economic model, which can cause substantial economic losses to the contract business system, large-scale data disruption, loss of authority management, failure of key functions, loss of credibility, or indirectly affect the operation of other smart contracts associated with it and cause substantial losses, as well as other severe and mostly irreversible harm.

- **High**

High impact generally refers to the vulnerability can have a relatively serious impact on the confidentiality, integrity, availability of the smart contract or its economic model, which can cause a greater economic loss, local functional unavailability, loss of credibility and other impact to the contract business system.

- **Medium**

Medium impact generally refers to the vulnerability can have a relatively minor impact on the confidentiality, integrity, availability of the smart contract or its economic model, which can cause a small amount of economic loss to the contract business system, individual business unavailability and other impact.

- **Low**

Low impact generally refers to the vulnerability can have a minor impact on the smart contract, which can pose certain security threat to the contract business system and needs to be improved.

3.1.3 Likelihood of Exploitation

- **Probable**

Probable likelihood generally means that the cost required to exploit the vulnerability is low, with no special exploitation threshold, and the vulnerability can be triggered consistently.

- **Possible**

Possible likelihood generally means that exploiting such vulnerability requires a certain cost, or there are certain conditions for exploitation, and the vulnerability is not easily and consistently triggered.

- **Unlikely**

Unlikely likelihood generally means that the vulnerability requires a high cost, or the exploitation conditions are very demanding and the vulnerability is highly difficult to trigger.

- **Rare**

Rare likelihood generally means that the vulnerability requires an extremely high cost or the conditions for exploitation are extremely difficult to achieve.

3.1.4 Fix Results Status

Status	Description
Fixed	The project party fully fixes a vulnerability.
Partially Fixed	The project party did not fully fix the issue, but only mitigated the issue.
Acknowledged	The project party confirms and chooses to ignore the issue.

3.2 Audit Categories

No.	Categories	Subitems
1	Coding Conventions	Deprecated Items
		Redundant Code
		require/assert Usage
		Default Values
2	General Vulnerability	Insufficient Address Validation
		Lack Of Address Normalization
		Variable Override
		DoS (Denial Of Service)
		Function Call Permissions
		Call/Delegatecall Security
		Tx.origin Usage
		Returned Value Security
		Mathematical Risk
		Overriding Variables
3	Business Security	Business Logics
		Business Implementations
		Manipulable Token Price
		Centralized Asset Control
		Arbitrage Attack
		Access Control

Beosin classified the security issues of smart contracts into three categories: Coding Conventions, General Vulnerability, Business Security. Their specific definitions are as follows:

- **Coding Conventions**

Audit whether smart contracts follow recommended language security coding practices. For example, smart contracts developed in Rust language should fix the compiler version and do not use deprecated keywords.

- **General Vulnerability**

General Vulnerability include some common vulnerabilities that may appear in smart contract projects. These vulnerabilities are mainly related to the characteristics of the smart contract itself, such as integer overflow/underflow and denial of service attacks.

- **Business Security**

Business security is mainly related to some issues related to the business realized by each project, and has a relatively strong pertinence. For example, whether the lock-up plan in the code match the white paper, or the flash loan attack caused by the incorrect setting of the price acquisition oracle.

3.3 Disclaimer

The Audit Report issued by Beosin is related to the services agreed in the relevant service agreement. The Project Party or the Served Party (hereinafter referred to as the "Served Party") can only be used within the conditions and scope agreed in the service agreement. Other third parties shall not transmit, disclose, quote, rely on or tamper with the Audit Report issued for any purpose.

The Audit Report issued by Beosin is made solely for the code, and any description, expression or wording contained therein shall not be interpreted as affirmation or confirmation of the project, nor shall any warranty or guarantee be given as to the absolute flawlessness of the code analyzed, the code team, the business model or legal compliance.

The Audit Report issued by Beosin is only based on the code provided by the Served Party and the technology currently available to Beosin. However, due to the technical limitations of any organization, and in the event that the code provided by the Served Party is missing information, tampered with, deleted, hidden or subsequently altered, the audit report may still fail to fully enumerate all the risks.

The Audit Report issued by Beosin in no way provides investment advice on any project, nor should it be utilized as investment suggestions of any type. This report represents an extensive evaluation process designed to help our customers improve code quality while mitigating the high risks in blockchain.

3.4 About Beosin

Beosin is the first institution in the world specializing in the construction of blockchain security ecosystem. The core team members are all professors, postdocs, PhDs, and Internet elites from world-renowned academic institutions. Beosin has more than 20 years of research in formal verification technology, trusted computing, mobile security and kernel security, with overseas experience in studying and collaborating in project research at well-known universities. Through the security audit and defense deployment of more than 2,000 smart contracts, over 50 public blockchains and wallets, and nearly 100 exchanges worldwide, Beosin has accumulated rich experience in security attack and defense of the blockchain field, and has developed several security products specifically for blockchain.

BEOSIN
Blockchain Security

Official Website
https://www.beosin.com

Telegram
https://t.me/beosin

Twitter
https://twitter.com/Beosin_com

Email
service@beosin.com