EdgeX

Security Assessment

Prepared by: **RigSec**

September 2, 2024

Contents

1 Introduction

edgeX engaged RigSec to conduct a comprehensive security assessment of their smart contracts from Aug 13, 2024 to Aug 20, 2024. The purpose of this audit was to ensure the security and reliability of the smart contracts implemented by edgeX.

1.1 About RigSec

RigSec is a blockchain regulatory technology company with a strong presence across Singapore, Hong Kong, Taiwan, and Japan. We are dedicated to providing regulatory-compliant digital asset wallet solutions and security consulting services.

With a focus on ensuring the integrity and resilience of blockchain technologies, we work closely with clients to identify and mitigate potential vulnerabilities within their systems. Our team of experts combines extensive knowledge of blockchain technology with a proactive approach to security, enabling us to provide comprehensive assessments and recommendations.

Through our meticulous audits of smart contracts and implementation of regulatory-compliant wallet solutions, RigSec helps clients protect their valuable digital assets and maintain the trust of their stakeholders.

1.2 About edgeX

edgeX V1, the Minimum Viable Product (MVP) of edgeX, is a high-performance, orderbook-based perpetual decentralized exchange (Perp DEX), crafted to deliver a native trading experience. This is achieved through advanced trading infrastructure, superior performance, and trader-centric features.

2 Executive Overview

The team of 3 consultants from RigSec meticulously examined the smart contracts provided by edgeX. This process involved a thorough analysis of the codebase, including static and dynamic testing. The consultants employed both automated and manual processes to test the security of the codebase.

2.1 Scope

The security assessment was scoped to the following address:

Ethereum Mainnet: 0xC0a1a1e4AF873E9A37a0caC37F3aB81152432Cc5

Binance Smart Chain: 0x0520b0A951658Db92b8A2dd9F146bB8223638740

Arbitrum: 0xceeED84620e5eb9ab1d6Dfc316867D2cdA332E41

2.2 Risk Classification

RigSec uses a risk classification matrix to determine the risk of a found issue. The matrix is based on the likelihood of the issue occurring and the impact of the issue. The risk classification is based on the following matrix:

Table 1: Risk Classification

Risk	Impact - High	Impact - Medium	Impact - Low
Likelihood - High	Critical	High	Medium
Likelihood - Medium	High	Medium	Low
Likelihood - Low	Medium	Low	Informational

2.3 Finding Summary

Table 2: Summary of Findings

Critical	High	Medium	Low	Informational
0	0	1	2	1

3 Findings

3.1 (RS-01) Risk when swapping with 1inch router

Severity: Medium Risk
Status: Confirmed

Description

In the *MultiSigPoolV5WithPermit::deposit()`* function, the input tokens would be swapped to USDT using the 1inch router. Specifically, the *swap()* function of the 1inch router smart contract is used to swap arbitrary tokens to USDT. Since the swap is done with a low-level call with calldata provided by the user, a malicious user could potentially invoke other functions of the 1inch router, leading to unspecified risks.

```
// Swap token
(bool success, bytes memory returndata)= AGGREGATION_ROUTER_V5_ADDRESS.call{value:msg.
↪value}(exchangeData);
require(success, "exchange failed");
```

Recommendation

Validate the user input to ensure that the first 4 bytes match signature of the swap() function.

3.2 (RS-02) Signature does not follow EIP-712

Severity: Low Risk
Status: Confirmed

Description

The message signing operations in the following functions do not adhere to the EIP-712 specification: *depositWithPermit()*, *withdrawETH()*, *withdrawErc20()*, *withdrawERC20Mpc()*, and *factTransferErc20()*. Not following EIP-712 for signing data can lead to unclear and ambiguous data structures, increasing the risk of signing unintended or malicious data. It also makes signatures vulnerable to replay attacks due to the lack of context-specific information. Users may have poor experience because of cryptic and hard-to-understand signing requests. In addition, the interoperability with other tools and services that support EIP-712 would be hindered.

```
function depositWithPermit(
...
    // check MPC signature
    require(ECDSA.recover(ECDSA.toEthSignedMessageHash(keccak256(abi.
↪encodePacked(amount, starkKey, positionId, block.chainid))), mpcSignature)  ==␣
↪owner,"invalid mpc signature");

function withdrawETH(
...
    bytes32 operationHash = keccak256(abi.encodePacked("ETHER", to, amount,␣
↪expireTime, orderId, address(this), block.chainid));
    operationHash = ECDSA.toEthSignedMessageHash(operationHash);
```

Recommendation

Follow EIP-712 when signing and verifying messages.

3.3 (RS-03) USDT lacks a permit operation on both Ethereum and BSC networks, leading to the failure of depositWithPermit function

Severity: Low Risk
Status: Confirmed

Description

The *MultiSigPoolV5WithPermit.depositWithPermit()* function plans to execute the *permit()* operation on USDT tokens when deployed on Ethereum, BSC, and Arbitrum. However, implementations of USDT differ across these blockchains. Notably, USDT does not support the *permit()* operation on the BSC chain and Ethereum, causing the *depositWithPermit()* function to always fail on these blockchains.

```
// permit call
IERC20Permit(USDT_ADDRESS).permit(owner,address(this),amount,deadline,v,r,s);
```

Recommendation

Since the *depositWithPermit()* function is ineffective on Ethereum and BSC, it can be removed on the version deployed on those two blockchains.

3.4 (RS-04) Deprecate testnet used in the contract

Severity: Informational
Status: Confirmed

Description

In the *MultiSigPoolV5WithPermit.deposit()* and *MultiSigPoolV5WithPermit.depositWithPermit()* functions, the chain ID must be 1 (Ethereum mainnet), 5 (Goerli), or 11155111 (Sepolia) to proceed with depositing ERC20 tokens into the StarkEx service. However, after the Dencun upgrade, the Goerli testnet has been deprecated and is no longer in use. Therefore, checking *block.chainid==5* would be redundant here. More details are available in this resource: https://blog.ethereum.org/2023/11/30/goerli-lts-update

```
if (block.chainid == 1 || block.chainid == 5 || block.chainid == 11155111){
  // safeApprove requires unsetting the allowance first.
  IERC20(USDT_ADDRESS).safeApprove(STARKEX_ADDRESS, 0);
  IERC20(USDT_ADDRESS).safeApprove(STARKEX_ADDRESS, returnAmount);

  // deposit to starkex
  IStarkEx starkEx = IStarkEx(STARKEX_ADDRESS);
  starkEx.depositERC20(starkKey, ASSET_TYPE, positionId, returnAmount);
  return returnAmount;
}
```

Recommendation

Remove references to the Goerli testnet due to its deprecation post-Dencun upgrade.