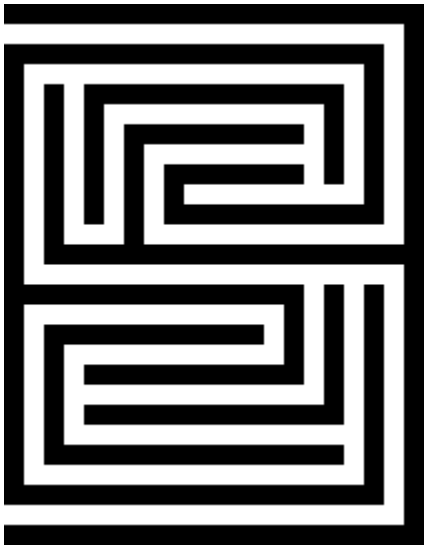

EdgeX Token Audit Public Report

PROJECT: EdgeX ERC20 Audit

December 2020

Prepared For:

Mahesh Sashital | Edge196, LLC.

mahesh@edge196.com

Prepared By:

Jonathan Haas | Bramah Systems, LLC.

jonathan@bramah.systems

Table of Contents

EdgeX ERC20 Token Review

Executive Summary

Scope of Engagement

Bramah Systems, LLC was engaged in December of 2020 to perform a comprehensive security review of the EdgeX token smart contracts (specific contracts denoted within the appendix). Our review was conducted over a period of one business day by a member of the Bramah Systems, LLC. executive staff.

Bramah Systems completed the assessment using manual, static and dynamic analysis techniques.

Timeline

Review Commencement: December 8th, 2020

Report Delivery: December 9th, 2020

Engagement Goals

The primary scope of the engagement was to evaluate and establish the overall security of the EdgeX token, with a specific focus on trading actions. In specific, the engagement sought to answer the following questions:

- Is it possible for an **attacker** to steal or freeze tokens?
- Does the Solidity code match the specification as provided?
- Is there a way to interfere with the contract mechanisms?
- Are the arithmetic calculations trustworthy?

Token Specification

A basic specification document was compiled by the review team based upon review of the EdgeX token smart contract code and discussion with the team. As EdgeX's token contracts are based upon the OpenZeppelin standards, extensive test documentation exists for their construction and modification.

Overall Assessment

Bramah Systems was engaged to evaluate and identify any potential security concerns within the codebase of the EdgeX token. During the course of our engagement, Bramah Systems noted only minor instances wherein the team deviated materially from established best practices and procedures of secure software development within DLT.

These aside, the team otherwise used thoroughly reviewed and vetted components, going as far as to establish role-based access control for user permissions. This methodology provides for expanded granular control over permissions, should EdgeX choose to change these assignments in the future. This future leaning focus with an emphasis on security shows the steps the organization has made to ensure the security of their token.

Disclaimer

As of the date of publication, the information provided in this report reflects the presently held, commercially reasonable understanding of Bramah Systems, LLC.'s knowledge of security patterns as they relate to the EdgeX Protocol, with the understanding that distributed ledger technologies ("DLT") remain under frequent and continual development, and resultantly carry with them unknown technical risks and flaws. The scope of the review provided herein is limited solely to items denoted within "Scope of Engagement" and contained within "Directory Structure". The report does NOT cover, review, or opine upon security considerations unique to the Solidity compiler, tools used in the development of the protocol, or distributed ledger technologies themselves, or to any other matters not specifically covered in this report.

The contents of this report must NOT be construed as investment advice or advice of any other kind. This report does NOT have any bearing upon the potential economics of the EdgeX protocol or any other relevant product, service or asset of EdgeX or otherwise. This report is not and should not be relied upon by EdgeX or any reader of this report as any form of financial, tax, legal, regulatory, or other advice.

To the full extent permissible by applicable law, Bramah Systems, LLC. disclaims all warranties, express or implied. The information in this report is provided "as is" without warranty, representation, or guarantee of any kind, including the accuracy of the information provided. Bramah Systems, LLC. makes no warranties, representations, or guarantees about the EdgeX Protocol. Use of this report and/or any of the information provided herein is at the users sole risk, and Bramah Systems, LLC. hereby disclaims, and each user of this report hereby waives, releases, and holds Bramah Systems, LLC. harmless from, any and all liability, damage, expense, or harm (actual, threatened, or claimed) from such use.

Timeliness of Content

All content within this report is presented only as of the date published or indicated, to the commercially reasonable knowledge of Bramah Systems, LLC. as of such date, and may be superseded by subsequent events or for other reasons. The content contained within this report is subject to change without notice. Bramah Systems, LLC. does not guarantee or warrant the accuracy or timeliness of any of the content contained within this report, whether accessed through digital means or otherwise.

Bramah Systems, LLC. is not responsible for setting individual browser cache settings nor can it ensure any parties beyond those individuals directly listed within this report are receiving the most recent content as reasonably understood by Bramah Systems, LLC. as of the date this report is provided to such individuals.

General Recommendations

Best Practices & Solidity Development Guidelines

Usage of version control systems & CI/CD

We recommend the usage of a formal version control system such as Git alongside the usage of CI/CD pipelines. While these contracts are of relative simplicity (due to the robustly defined specification upon which they are built), more complex contracts will benefit from the ability to record code changes over time.

Similarly, by adding continuous deployment / continuous development, automated test suites may be run on any development work. This speeds up any work streams which require repeated testing, often common in Solidity development.

Resolution: The team will present code in a publicly accessible Git repository.

Out of date Solidity Version

The current pragma version of the contract (0.6.2) is out of date. As the Solidity compiler frequently adds new checks to security and performance, we recommend that unless explicitly required (e.g. a function no longer exists in a newer Solidity version), that the contracts be deployed with a more recent version (e.g. 0.6.10 or 0.6.11).

Resolution: The team will update the smart contracts accordingly.

Specific Recommendations

Unique to the EdgeX Protocol

Pre-Flattened Solidity Files

The Solidity files come flattened, containing all contracts needed for deployment of the contract. However, by deploying this way, comparing changes between files (especially those considered to be template code) becomes increasingly difficult, which may present confusion. This is exacerbated by the lack of version control within the contracts.

Resolution: This noted, the team has also provided a non-flattened version of the contracts, mitigating concern.

Highly Permissive Owner Role

The token code features minting, burning, pausing and unpausing functionality, all of which are exclusively assigned to the control of the **msg.sender** address, which will be assigned to the individual who deploys the contract. As this account is highly permissive in nature, it is suggested that the private keys for this address be held in a hardware storage wallet, ideally in a tamper evident / resistant location.

Where possible, such functionality should be divided amongst a large number of trusted parties, requiring multiple individuals to approve irreversible actions such as token burning.

As with any highly permissive key, principles of operational security should be followed, and careful recordings should be made of all usage.

Resolution: The team is aware of operational security best practices.

Toolset Warnings

Unique to the EdgeX Token

Overview

In addition to our manual review, our process involves utilizing static analysis and formal methods in order to perform additional verification of the presence of security vulnerabilities (or lack thereof). An additional part of this review phase consists of reviewing any automated unit testing frameworks that exist.

The following sections detail warnings generated by the automated tools and confirmation of false positives where applicable.

Compilation Warnings

No compilation warnings are generated by the contract as of file SHA256 sum **bd800d0797a5f768f08897b447c0e18b87bb3ae488cac29556b3e877984da911** of **EdgeXTokenSingleFile.sol**.

Test Coverage

The contract repository possesses substantial unit test coverage throughout. This testing traditionally provides a variety of unit tests which encompass the various operational stages of the contract, largely from the usage of OpenZeppelin token logic.

Static Analysis Coverage

The contract repository underwent heavy scrutiny with multiple static analysis agents, including:

- Securify
- MAIAN
- Mythril
- Oyente
- Slither

In each case, the team had either mitigated relevant concerns raised by each of these tools or

provided adequate justification for the risk (such as adhering to the ERC-20 token standard).

Slither in particular was used to validate matching signatures, matching return types, the lack of existence of custom modifiers, proper event emissions, and appropriate usage of allowance (avoiding the ERC20 API Approve / TransferFrom attack). Each function was then manually inspected for any unintended or unclear behaviour which could pose a material impact to end users.

We found no deviation from the intended ERC20 function definitions and modifiers during the scope of our analysis.

== ERC20 functions definition ==

[✓] transfer (address, uint256) -> (bool)

[✓] approve (address, uint256) -> (bool)

[✓] transferFrom (address, address, uint256) -> (bool)

[✓] allowance (address, address) -> (uint256)

[✓] balanceOf (address) -> (uint256)

== Custom modifiers ==

[✓] No custom modifiers in ERC20 functions

== ERC20 events ==

[✓] Transfer (address, address, uint256)

[✓] Approval (address, address, uint256)

[✓] transfer must emit Transfer (address, address, uint256)

[✓] approve must emit Approval (address, address, uint256)

[✓] transferFrom must emit Transfer (address, address, uint256)

== ERC20 getters ==

[✓] totalSupply () -> (uint256)

[✓] decimals () -> (uint8)

[✓] symbol () -> (string)

[✓] name () -> (string)

== Allowance frontrunning mitigation ==

[✓] increaseAllowance (address, uint256) -> (bool)

[✓] decreaseAllowance (address, uint256) -> (bool)

[✓] increaseAllowance emits Approval (address, address, uint256)

[✓] decreaseAllowance emits Approval (address, address, uint256)

== Balance check in approve function ==

[✓] approve function should not check for sender's balance

Directory Structure

At time of review, the directory structure of the EdgeX ERC20 token contract repository appeared as it does below. Our review, at request of Edge196, covers the Solidity code (*.sol) as of file SHA256 sum **bd800d0797a5f768f08897b447c0e18b87bb3ae488cac29556b3e877984da911** of **EdgeXTokenSingleFile.sol**, the flattened version of the contracts.

```
├── README.md
├── clean_code
│   └── EdgeXToken.sol
├── contracts
│   ├── EdgeXTokenSingleFile.sol
│   └── Migrations.sol
├── install_nvm.sh
├── migrations
│   ├── 1_initial_migration.js
│   └── 2_edgex_token_migration.js
├── package-lock.json
├── package.json
├── test
│   ├── EdgeXToken.test.js
│   └── util.js
└── truffle-config.js

4 directories, 12 files
```