Vol.5 No.2

Advanced Engineering and ICT-Convergence Proceedings (AEICP)

9th International Conference on Advanced Engineering and ICT-Convergence (ICAEIC-2022)

July 13-15, 2022

Organized by

ICT - Advanced Engineering Society, Seoul, Korea (ICT-AES)

Bima Build. #525, 20 Kwangwoon-ro, Nowon-gu, Seoul, Korea

Email: info@ictaes.org, Tel.: +82-2-940-8626 / 8637

Sponsored by

TRACOM

BonC nnovators

Applying Code Visualization into Solidity for Auditing of Smart Contract

Chansol Park[1], Bokyung Park[2], Soyoung Moon[3], and R. Young Chul Kim*

[1,3,4,*]Dept. of Software and Communication Engineering, SE Lab., Hongik University
[2]Dept. of Computer Education, Chinju National University of Education, South Korea

c2193102@g.hongik.ac.kr[1], parkse@cue.ac.kr[2], {msy[3], bob*}@hongik.ac.kr

Abstract

In near future, it will expect to rapidly be increasing Audit demand for Smart Contracts containing the core logic of DApp, but still manual or partially automated. This audit process has a limit to meet all demands. Therefore, we propose Code Visualization for the automatic audit process. Through this, it is expected to be able to produce an audit report rich in evidence and contents by melting the audit elements into the call graph.

Keywords: *Code Visualization, Smart Contract, Solidarity, Audit process, DApp.*

1. Introduction

As DApp services on blockchain networks is increasing, there are also increasing demand for Audit services for Smart Contracts containing Core Logic. However, most of the Audit processes currently in service are either manual or limitedly automated. In addition, it detects only vulnerabilities such as Smart Contract Weakness Classification (SWC) during Audit for source code. Auditing and generating reports through these non-automated processes is time consuming and inevitably increases the cost of auditing services. In addition, when only analyzing vulnerabilities, it is difficult to analyze the overall flow and content of the source code through an audit report. To solve this problem, we are conducting research on automating the audit process by applying the Our 'Software Architecture Visualization for Object-Oriented Programming' research. Among them, this paper mentions a study to derive code complexity by extracting Call Graphs for Solidity. With applying code visualization to Solidity, we can visualize the complexity inside the Solidity code and guide it to the problematic areas. Chapter 2 describes code visualization as a related study. Chapter 3 discusses code visualization of solidarity such as call graphs. Chapter 4 describes the output of the Call Graph Visualization tool. Finally, Chapter 5 describes the conclusion and future research.

Advanced Engineering and ICT-Convergence Proceedings (AEICP)
ISSN : 2635-4586

2. Related Works

With our code visualization technique, we extract information about source code through reverse engineering on software, and then visualize the design extraction and complexity of source code.[1] We can extract designs such as Class Diagram, Sequence Diagram, and Call Graph, and measure Complexity of various indicators such as Coupling, Cohesion, and MCC Metrics.

Regarding the quality of Solidity, the most representative indicator and most used indicator by Audit is SWC. SWC implements the vulnerability classification scheme proposed in Ethereum Improvement Proposal (EIP)-1470.[3] Each SWC item is associated with a Common Weakness Enumeration (CWE) items. The SWC includes a description of the vulnerability and an example code and an improved code.[4]

3. Our Code Visualization for Solidity

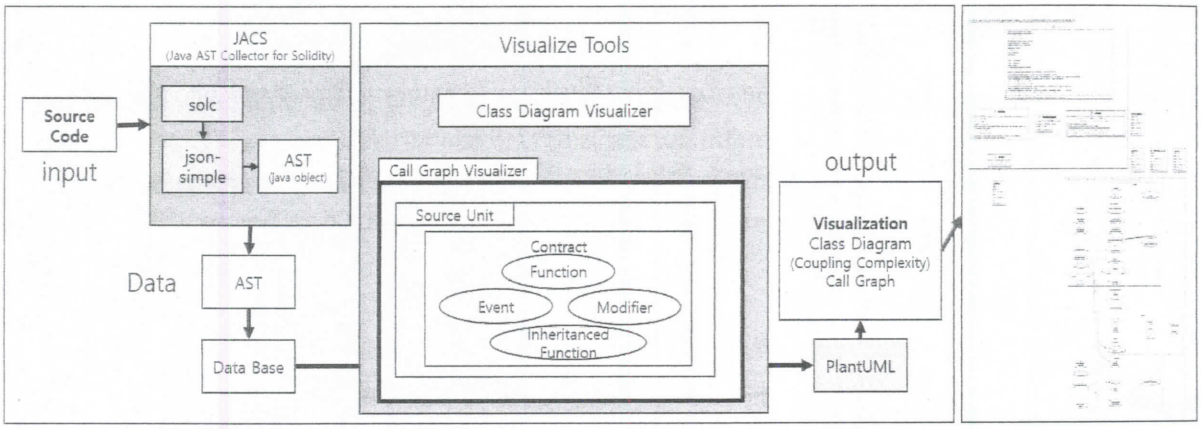


Figure 1. Solidity Visualization Toolchain.

The Solidity Visualization Toolchain receives the Solidity source code as inputs, extracts the AST into the JAVA object structure through the JACS(Java AST Collector for Solidity), and stores it in the local DB. Each visualization tool extracts and processes information DB information, and creates diagrams and graphs as outputs via Plant UML. Currently, the Solidity Visualization Toolchain can be automatically visualized for 'object-oriented complexities and class diagram' and Call Graph. We describe Call Graph among them.

The Call Graph Visualizer is a tool that extracts and processes information from DB, and generates PlantUML scripts as an output. The Call Graph Visualizer groups each function type in units of contract. At this time, the function type includes not only modifier and event, but also functions that included in the parent contract. The Contracts are then grouped in units of source code, and arrows are connected for all function calls.

4. Case Study

We explain our approach wirh the case study. Figure 2 is a part of the Extracted Call Graph for Sample code. An elliptical object in number 1 is a function. If the type of function is Modifier or Event, we state the

of function above the function name. Also, we state 'inheritanced' above the function name for function re inherited from the parent. Number 2 is Contract. Number 3 is Source Code. Finally, arrows in number function calls. In number 4, the left function (Caller Function) calls the right function (Callee Function). umber of calls is specified at the top of the arrow, and the arrow line becomes thicker as the number of increases. Through the Call Graph, various complexities such as counting for simple function calls or can be applied in the future

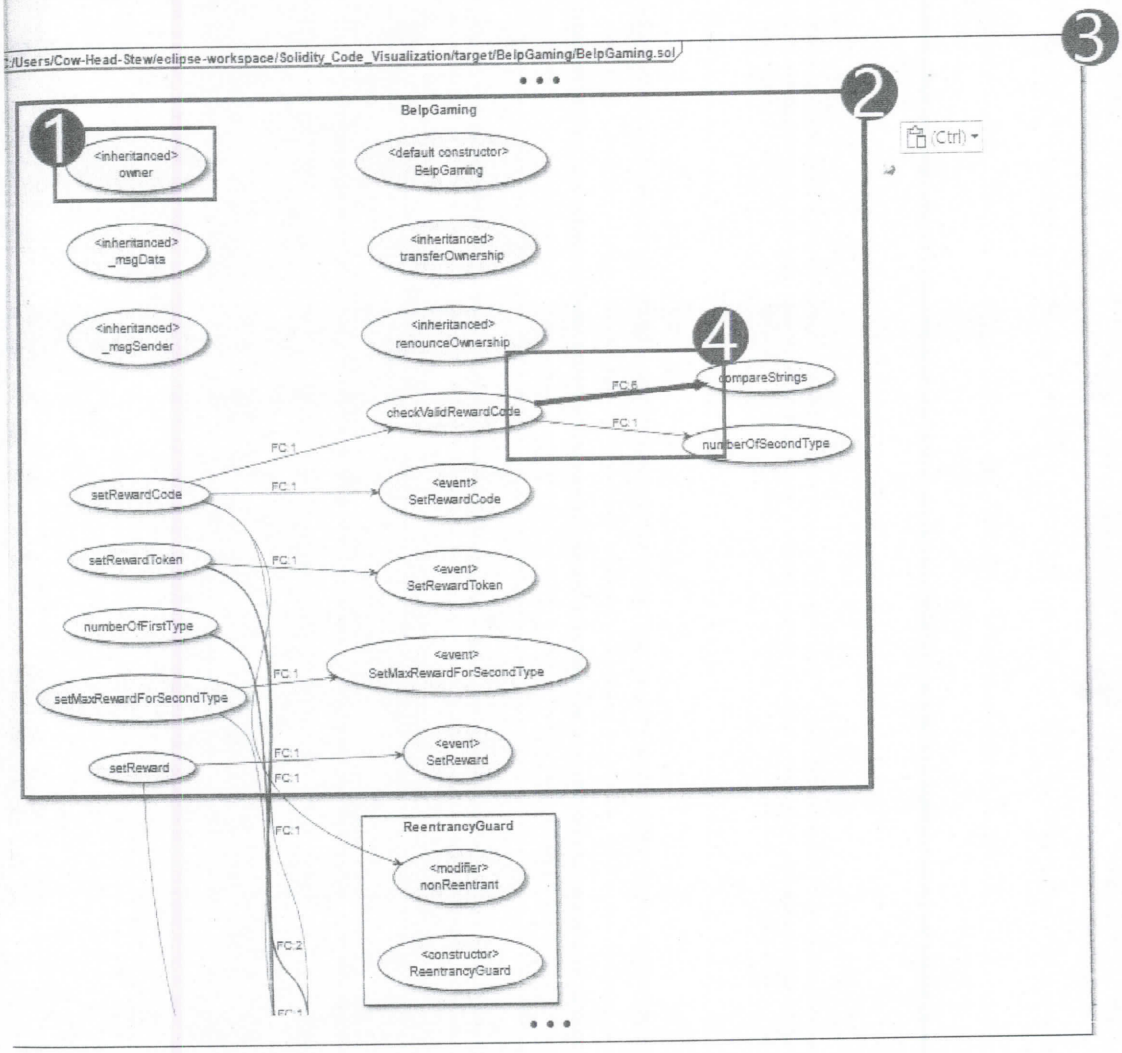


Figure 2. Part of Extracted Call Graph for Sample code.

onclusion

his paper, we add the Solidity Call Graph Visualizer to the Solidity Visualization Toolchain. In the future, ill analyze the correlation between Solidity and Smart Contract Audit elements (SWC, Gas Consumption, and the complexity we extracted. After this correlation is proven, we will study the improvement of ity Code Quality through complexities and our Software Architecture Visualization study.

Acknowledgement

This work was supported by the Ministry of Education and the Korea Research Foundation (F21YY8102068) and the government (Ministry of Education) with the support of the Korea Research Foundation (No. 2021R1I1A305040711, No. 2021R1I1A1A01044060) in 2022.

References

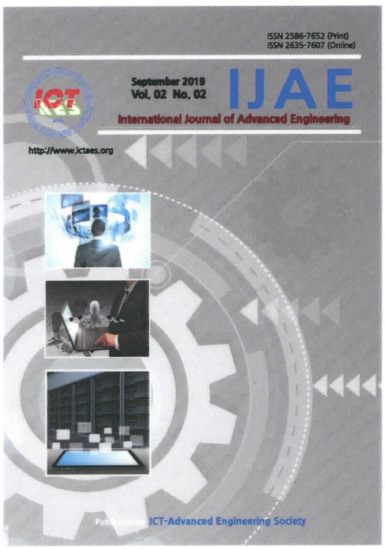
[1] Park, Bo Kyung, et al, (2020) Code Visualization for Performance Improvement of Java Code for Controlling Smart Traffic System in the Smart City, *Applied Sciences*, 10(8)

[2] Jung, Se Jun, et al, (2021) Automatic UML Design Extraction with Software Visualization based on Reverse Engineering. International journal of advanced smart convergence, 10(3), 89-96.

[3] Smart Contract Weakness Classification and Test Cases. https://swcregistry.io.

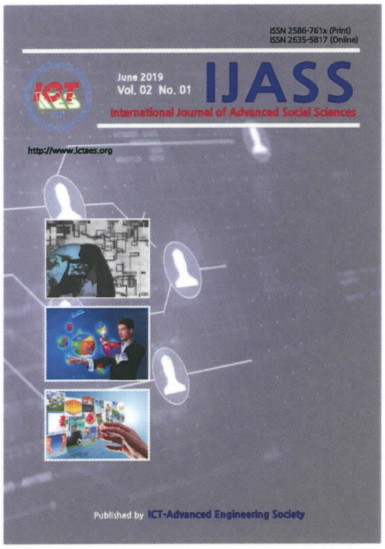
[4] CWE - Common Weakness Enumeration. https://cwe.mitre.org.

ICT-AES Publications



International Journal of Advanced Engineering (IJAE) is a scholarly open access, a peer-reviewed, and half-yearly journal focusing on theories, methods, and applications in Engineering and Technology. It covers all areas of Engineering and Technology, publishing original research articles and technical notes. All manuscript that are submitted must report unpublished work and cannot be under consideration for publication elsewhere.

Print ISSN: 2586-7652 | Online ISSN: 2635-7607



International Journal of Advanced Social Sciences (IJASS) is a scholarly open access, Peer-reviewed, and half-yearly journal focusing on theories, methods, and applications in Social Sciences. It publishes both theoretical and empirical articles and case studies relating to sociology, political science, history, a law in society and related disciplines. Published articles use scientific research methods, including statistical analysis, case studies, field research and historical analysis. All manuscript that are submitted must report unpublished work and cannot be under consideration for publication elsewhere.

Print ISSN:2586-761x | Online ISSN: 2635-5817

All submitted articles should report original, previously unpublished research results, experimental or theoretical and will be peer-reviewed.