Journal of The Korea Society of Computer and Information
Vol. 23 No. 12, pp. 89-94, December 2018
www.ksci.re.kr
https://doi.org/10.9708/jksci.2018.23.12.089

Implement Static Analysis Tool using JavaCC

Byeongcheol Kim*, Changjin Kim*, Seongcheol Yun*, Kyungsook Han*

Abstract

In this paper, we implemented a static analysis tool for weakness. We implemented on JavaCC using syntax information and control flow information among various information. We also tested the performance of the tool using Juliet-test suite on Eclipse. We were classified using information necessary for diagnosis and diagnostic methods were studied and implemented. By mapping the information obtained at each compiler phase the security weakness, we expected to link the diagnostic method with the program analysis information to the security weakness.

In the future, we will extend to implement diagnostic tools using other analysis information.

▸Keyword: Security, Weakness, Static Analysis, Syntax Information, Flow Information

I. Introduction

생활 전반에 IT 활용도가 높아짐에 따라 보안에 대한 관심이 증가하고 있으며, 이로 인해 개발자들의 개발보안에 대한 관심이 증가하고 있는 상황이다. 소프트웨어 개발자들은 시큐어 코딩 규칙을 준수하여 개발해야하는 인식이 확산되었고, 자신들이 개발한 소프트웨어를 검수하기 위한 도구의 사용이 확대되었다.

이에 따라 소프트웨어의 보안약점(Weakness)을 찾기 위한 방법에 대한 연구가 많이 진행되고 있는데, 그 중 한 가지 방법이 정적 분석(Static Analysis)[1]이다. 정적 분석이란 프로그램의 실행 없이 소프트웨어를 분석하는 것을 말한다. 보안약점은 공격자가 원하는 정보를 유출, 탈취 가능하거나 제어 흐름을 바꿀 수 있도록 하는 프로그램의 불완전한 부분을 의미하고, 보안약점을 공격자가 이용하여 실제 보안 사고가 발생하면 그 보안약점을 취약점(Vulnerability)이라고 한다.

본 논문에서는 CWE[2]에서 Java와 관련된 보안약점을 분석한 정보를 바탕으로 CERT[3]의 시큐어 코딩 규칙을 적용하여 예방할 수 있는 보안약점을 분류하였다. 또한 분류에 따라 컴파일러의 각 단계에서 수행할 수 있는 진단 방법 중 구문 정보(Syntax Information)[4][5]와 흐름 정보(Flow Information)[4][5]를 이용하여 진단할 수 있는 보안약점에 대해 JavaCC[6]로 정적 분석 도구를 구현하였다. 또한 SAMATE에서 제공하는 정적 분석 도구 벤치마크인 Juliet 테스트 모음[7]을 사용하여 구현한 정적 분석 도구의 성능을 검증하였다.

본 논문의 구성은 다음과 같다. 2절에는 보안약점을 진단하기 위한 관련 연구에 대해 기술하고, 3절에서 각 컴파일러 단계별 얻을 수 있는 정보를 분류하고 그에 따른 진단 방법을 서술한다. 4절에서 JavaCC로 구현된 내용을 설명하고, 5절에서 Juliet 테스트 모음을 이용하여 성능을 검증한다. 마지막으로 6절에서 결론을 맺는다.

II. Related Works

보안약점 진단 방법에 대한 연구는 주로 정적 분석 도구 개발과 관련하여 이루어지고 있다. 따라서 본 논문에서는 정적 분석 도구 구현을 위해 사용한 JavaCC에 대한 분석과 정적 분석 도구를 검증하기 위한 벤치마크로 사용되는 Juliet 테스트 모음에 대하여 분석하였다. 또한 현재 상용중인 정적 분석 도구인

• First Author: Byeongcheol Kim, Corresponding Author: Kyungsook Han
*Byeongcheol Kim (altorious153@gmail.com), Dept. of Computer Engineering, Korea Polytechnic University
*Changjin Kim (dhgoak45@naver.com), Dept. of Computer Engineering, Korea Polytechnic University
*Sungcheol Yun (sungchul09@naver.com), Dept. of Computer Engineering, Korea Polytechnic University
*Kyungsook Han (khan@kpu.ac.kr), Dept. of Computer Engineering, Korea Polytechnic University
• Received: 2018. 10. 18, Revised: 2018. 11. 17, Accepted: 2018. 11. 20.
• This work was supported by the National Research Foundation of Korea(NRF-2017R1A2B1011354)

fortify SCA[8]와 sparrow[9], Codemind CSI[10]에 대하여 분석하였다.

2.1. JavaCC

JavaCC는 자바 프로그래밍 언어로 작성된 오픈 소스 구문 생성기이자 어휘 생성기이다. JavaCC는 EBNF 표기법으로 작성된 형식 문법의 파서를 생성한다는 점에서 yacc와 비슷하지만, 상향식 파서를 생성하는 yacc와 달리 JavaCC는 하향식 파서를 생성한다. JavaCC는 다음 k 입력 토큰에 기반한 선택을 해결할 수 있으므로 LL(k) 문법을 자동으로 처리할 수 있다.

JavaCC는 options, PARSER_BEGIN ~ PARSER_END, TOKEN, 실제 구문 부분으로 나뉘어져있다. options 부분에는 구현하는 문법을 파싱하는데 필요한 옵션을 정의하고, PARSER_BEGIN과 PARSER_END 사이에는 파싱을 할 문자열 또는 파일을 읽어 실제 파싱을 진행하며, TOKEN은 파싱을 진행하면서 예약어 또는 리터럴 상수를 정의한다. 이후에 작성하는 부분은 작성자가 원하는 파싱 작업을 할 문법을 작성하였다.[11]

본 논문에서는 이러한 구조를 가지는 JavaCC를 이용하여 TOKEN 부분에는 어휘 정보를 얻을 수 있도록 작성한 후, 구문 기술 부분에 Java 문법을 사용하여 Java 언어로 작성된 코드에 대한 분석 도구를 구현하였다.

2.2 Juliet Test Suite

Juliet 테스트 모음은 미국 국가안보국의 CAS(Center for Assured Software)에서 개발되었다.[7] 미국 국립표준기술연구소에서 정적 분석 도구의 성능을 평가하기 위한 시험코드로 Juliet 테스트 모음을 채택했다. Juliet 테스트 모음은 CWE를 기반으로 Java와 C/C++로 나뉘어져 있으며, CWE 항목 별로 분류되어 있으며, 한 가지의 CWE 항목 내에서도 여러 유형의 테스트 코드로 나뉘어져 있어 유형 별로 성능을 평가할 수 있다.

본 논문에서는 Java 정적 분석 도구를 구현하였으므로 C/C++을 제외한 Java와 관련된 보안약점 항목을 추출하고, 그에 관련된 Juliet 코드를 이용하였다.

2.3 Static Analysis Tool

소프트웨어 보안약점을 찾기 위해 여러 가지 정적 분석 도구들이 개발되고 있다. 해외에는 fortify SCA, 국내에는 sparrow, Codemind CSI 등의 도구가 있다.

fortify SCA는 Micro Focus에서 개발한 정적 분석 도구로 OWASP TOP 10, CWE를 포함하는 430여 보안 및 품질 관점의 점검 범주를 지원하며 특히 Java, C#, C/C++, Swift, PHP를 포함하여 25개 언어에 대한 소스 및 관련 라이브러리들의 보안약점을 점검한다. 또한 개발 초기 단계부터 개발 프로세스와 연동되어 개발자에 의한 보안약점에 대한 조기 발견 및 조치가 가능하도록 함으로써 보안이 확보된 응용 프로그램 개발을 가능하도록 하고 있다.[8]

sparrow는 파수닷컴에서 개발한 정적 분석 도구로 행정안전부 47개 보안약점, 국정원 8대 취약점, CWE외 다수의 점검 항목들을 지원하며 Java, Python, JavaScript외 다수의 언어에 대하여 점검하며 전자정부표준프레임워크의 점검 또한 지원한다. 또한 인공지능 기반의 기술을 적용해 검출된 취약점에 대해 자동으로 안전하게 수정된 코드를 제안하고 파일 간의 연관성까지 확인 가능한 영향도 분석을 지원한다.[9]

Codemind CSI는 코드마인드에서 개발한 정적 분석 도구이며 국정원 8대 취약점 검사와 OWASP TOP 10, CWE외 다양한 점검 항목들을 지원하며 Java, JSP, HTML, Android, iOS 등 다수의 언어 점검을 지원한다. 질의기반으로 소스코드를 분석하며 분석한 코드에 대한 보고서를 자동으로 생성한다. 또한 웹기반 분석관리시스템을 사용하여 별도 클라이언트나 빌드환경 없이 분석이 가능하다.[10]

III. Classification by Diagnostic Method

본 절에서는 각 컴파일러 단계별 얻을 수 있는 정보를 구문 정보, 흐름 정보, 타입 정보, 값 정보로 분류한다. Table 1.은 CWE의 보안약점을 정보별로 분류한 것이다. 또한 이러한 정보를 이용하여 보안약점을 진단할 때, 어떠한 방식으로 진단할 것인가에 대한 진단 방법을 기술한다.

Table 1. Classification by Information from Compiler Phase

Syntax Information	CWE-248 : Uncaught Exception CWE-500 : Public static Field Not Marked Final CWE-582 : Array Declared Public, Final, and Static
Flow Information	CWE-89 : Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') CWE-609 : Double-Checked Locking CWE-833 : Deadlock
Type Information	CWE-192 : Integer Truncation Error CWE-597 : Use of Wrong Operator in String Comparison CWE-681 : Incorrect Conversion between Numeric Types
Value Information	CWE-134 : Use of Externally-Controlled Format String CWE-135 : Incorrect Calculation of Multi-Byte String Length CWE-252 : Unchecked Return Value

3.1 Diagnostic Method by Syntax Informations

구문 정보는 소스 코드를 파싱하는 단계에서 추출 가능한 정보를 의미한다. 어휘 정보는 스캐너 단계에서 분석 가능한 정보로, 분석 도구에서는 어떤 함수를 사용하였는가에 대한 여부를 확인할 수 있는 정보이다. Table 2.는 구문 정보를 이용하여 진단할 수 있는 보안약점과 진단 방법의 예시이다.

구문 정보는 어휘 정보인 토큰을 이용하여 생성된 구문 트리(Syntax Tree) 상에서 트리 노드의 존재 유무나 위치를 의미한다. 예를 들어, 구문 트리 상에서 catch 절에 해당하는 노드의 하위 노드로 어떤 종류의 exception이 포함되어 있는지 여

Table 2. Diagnostic Method using Syntax Information

CWE List	CWE-396 : Declaration of Catch for Generic Exeption	CWE-337 : Same Seed in Pseudo-Random Number Generator(PRNG)
CERT Rule	ERR07-J. Do not throw RuntimeException, Exception, or Throwable	MSC02-J. Generate strong random numbers
Diagnostic Method	1) Find try-catch block. 2) Check the catch clause. 3) If catch clause is RuntimeException or Exception, diagnose security weakness.	1) Find random() function. 2) Check the function is safe. 3) If the function is Math class or Util class, diagnose security weakness.

Table 3. Diagnostic Method using Flow Information

CWE List	CWE-89 : Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	CWE-321 : Use of Hard-coded Cryptographic Key
CERT Rule	IDS00-J. Prevent SQL injection	MSC03-J. Never hard code sensitive information
Diagnostic Method	1) Find weakness function like executeQuery(). 2) If weakness function is used, 1st Diagnose security Weakness. 3) Else Check SQL statement is written as Statement class 4) If it has received external input and not checked that input, 2nd Diagnose security Weakness.	1) Find weakness function like getConnection(). 2) Check getConnection() using String argument. then, diagnose security Weakness. 3) If variable is used not String argument, check flow information 4) That variable is not input stream. then, diagnose security Weakness.

부를 검사한다. 이를 이용하여 “Declaration of Catch for Generic Exception(CWE-396)”에 해당하는 보안약점을 진단할 수 있다.[5]

3.2 Diagnostic Method by Flow Informations

흐름 정보는 제어 흐름(Control Flow)과 데이터 흐름(Data Flow) 두 가지 정보가 있다. 제어 흐름은 문장 사이의 실행 순서를 의미하고, 데이터 흐름은 제어 흐름을 기반으로 데이터가 직접적 또는 간접적으로 영향을 미치는 관계를 의미한다.[5] Table 3.은 흐름 정보를 이용하여 진단할 수 있는 보안약점과 진단 방법의 예시이다.

예를 들어, executeQuery()를 이용해 SQL문을 실행하는 과정에서 문자열을 필터링 없이 사용하려는 경우 공격자가 임의의 SQL문을 삽입하여 원하는 정보를 탈취하거나 인증을 우회하는 등의 공격을 할 수 있는 “Improper Neutralization of Special Elements used in an SQL Command(‘SQL Injection’)(CWE-89)” 보안약점으로 진단할 수 있다. 이 보안약점은 Java 언어의 경우 executeQuery() 함수 호출에서 시작하여 제어 흐름상에서 PrepareStatement 클래스 메서드를 사용하는 도미네이터(Dominator) 블록이 존재하는지 여부를 이용하여 진단할 수 있다.

다른 예로, “Use of Hard-coded Cryptographic Key(CWE-321)“의 경우 암호 변수를 사용하는 함수 호출 이전에 제어 흐름상에서 외부 입력을 받는 도미네이터 블록이 존재하지 않으면 보안약점으로 진단할 수 있다.

3.3 Diagnostic Method by Type Informations

타입 정보로 진단할 수 있는 보안약점은 의미 분석 단계 중에서 얻어지는 타입 정보를 사용하여 분석할 수 있는 보안약점을 의미한다. 타입 정보는 컴파일 과정에서 심볼 테이블 정보를 이용하여 구문 트리에 정보를 추가하여 진단 가능하다.

단순한 구문 정보에 타입 정보를 추가해서 분석이 가능한 보안약점의 예로. “Numeric Truncation Error(CWE-197)” 보안약점을 들 수 있다. 대입문의 구문 트리를 볼 때, 저장될 값에 대한 수식 부분의 타입과 값이 저장될 변수의 타입을 비교함으로써 이러한 보안약점이 발생할 가능성이 있는지 확인할 수 있다.

3.4 Diagnostic Method using Value Analysis Informations

값 추적 분석은 어떤 지점에서 변수 또는 수식이 가질 수 있는 값의 범위를 유추하는 작업을 의미한다. 값의 범위를 분석하기 위한 방법이 여러 가지 있는데, 그 중 대표적인 분석 방법으로는 기호 기반 실행(Symbolic Execution)[12], 요약 해석(Abstract Interpretation)기법[13] 등이 있다.

기호 기반 실행 기법은 프로그램에서 사용하는 변수를 심볼(Symbol)로 표현하고, if문 같은 조건문을 만나면 기호에 대한 제약식을 추가해 나가면서 심볼이 흘러갈 수 있는 가능한 모든 경로에 대해 분석하면서 값의 범위를 추론하는 방법이다. 기호 기반 실행 기법을 이용할 때, 프로그램의 크기가 커질수록 실행 가능한 경로의 수가 기하급수적으로 늘어나 시간과 메모리 소모가 극심해지는 경로 폭발(Path Explosion)[14]문제로 부정확한 분석이 되는 경우가 발생할 수 있다.

요약 해석 기법은 실제 값을 요약 값으로 추상화하고 프로그램을 요약 영역(Abstract Domain) 상에서 해석하는 방식이다. 값을 추상화하여 사용하기 때문에 실제 실행과 가장 유사한 정보를 얻을 수 있으므로 실제 실행과 가까운 정보를 얻을 수 있어 동적 분석을 실행한 것과 유사한 효과를 기대할 수 있다.

값 정보 분석을 이용하면 허용 가능한 값의 범위를 지정하

고, 그 범위에서 벗어난 변수로 인해 발생하는 보안약점을 진단할 수 있다. 허용 가능한 값의 범위를 상수만 허용할 것인지, 기호 기반 실행이나 요약 해석 기법으로 변수를 사용한 범위를 허용할 것인지에 따라 복잡도가 달라질 것이다.

IV. Implementation

본 절에서는 JavaCC를 기반으로 CWE 보안약점을 CERT 코딩 규칙을 이용하여 진단하는 도구를 구현하였다. 컴파일러 단계에서 추출할 수 있는 정보에 따라 분류한 보안약점 중 구문 정보와 흐름 정보에 대한 진단 방법 중 일부를 구현하였다.

4.1 Implementation List

Table 4. Implementation List(using Syntax Information)

Coding Standard
ERR07-J. Do not throw RuntimeException, Exception, or Throwable
ERR08-J. Do not catch NullPointerException or any of its ancestors
OBJ01-J. Limit accessibility of fields
MSC02-J. Generate strong random numbers

Table 4.는 본 논문이 구현한 도구가 검출할 수 있는 CERT의 코딩 규칙 중 구문 정보를 이용하는 규칙을 나열한 것이다.

구문 정보를 이용하여 진단할 수 있는 코딩 규칙은 주로 구문 트리에서 어휘 정보를 함께 이용하는 경우가 많다. 예를 들면 catch 절에서 Exception 또는 NullPointerException을 찾는 경우로 해당 어휘 정보의 존재 여부에 따라 보안약점으로 진단한다. 또한 “Limit accessibility of fields”는 변수의 구문트리 상에서 변수이름 앞에 붙는 접근지정자가 없거나 public, protected 또는 private인지에 따라 보안약점의 여부를 확인한다.

Table 5. Implementation List(using Flow Information)

Coding Standard
IDS00-J. Prevent SQL Injection
IDS01-J. Normalize strings before validating them
FIO05-J. Do not expose buffers created using the wrap() or duplicate() methods to untrusted code
FIO08-J. Distinguish between characters or bytes read from a stream and -1
SER05-J. Do not serialize instances of inner classes
MSC03-J. Never hard code sensitive information

Table 5.는 흐름 정보를 이용하여 검출할 수 있는 코딩 규칙을 나열하였다. 흐름 정보를 이용하는 경우에도 어휘 정보를 먼저 이용하여 찾은 어휘 정보에서부터 이전 또는 이후 흐름에서 다른 어휘 정보나 구문 정보를 찾아 보안약점이 존재하는지 여부를 확인하여 진단한다. 예를 들어, “Normalize strings before validating them”의 경우 정규화를 거치지 않고 사용하게 될 경우 위험한 함수가 있을 때, 그 이전 흐름에서 정규화를 하는 함수가 존재하는지 여부를 확인하여 그렇지 않다면 보안약점으로 검출하는 식이다.

4.2 Diagnosis using Syntax Information

구문 정보를 이용하는 경우 JavaCC에서 추상 구문 트리(Abstract Syntax Tree)상의 구조를 분석하는 형태이다. Fig. 1.은 구문 정보를 이용한 진단 방법을 간단히 보여준다.

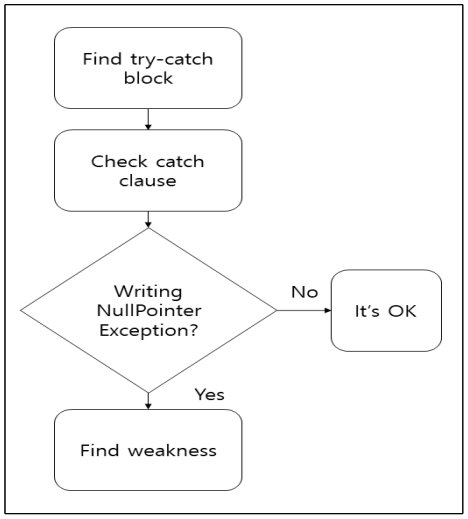

Fig. 1. Diagnostic Algorithm using Syntax Information

예를 들어 “Do not catch NullPointerException or any of its ancestors”에 대한 진단 방법으로 JavaCC의 파서를 작성하는 부분의 TryStatement를 찾아 catch 절의 자식 노드 값과 NullPointerException을 비교하여, 해당 예외와 같다면 코딩 규칙 위반으로 진단하고 코딩 규칙의 이름과 작성된 코드의 위치 정보를 출력한다. 이 외에도 “Do not throw RuntimeException, Exception, or Throwable”의 경우도 같은 TryStatement에서 catch 절의 자식 노드 값을 비교하여 코딩 규칙의 이름과 위치 정보를 출력한다.

4.3 Diagnosis using Flow Information

흐름 정보를 이용하는 경우 구문 분석 과정의 명세 부분에서 JavaCC를 통해 얻은 구문 정보와 라인에 대한 정보를 Java 배열리스트로 저장하고 보안약점이 될 수 있는 함수들의 리스트를 저장하며, 배열 리스트를 탐색하면서 해당 함수 이전과 이후에 보안약점이 되는 함수 또는 구문을 찾는다. Fig. 2.는 흐름 정보를 이용한 진단 방법을 간단히 보여준다.

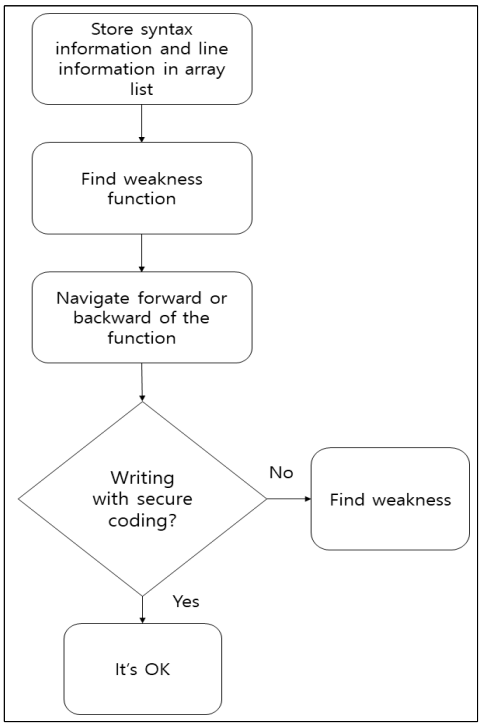

Fig. 2. Diagnostic Algorithm using Flow Information

예를 들어 "Prevent SQL Injection"은 executeQuery 함수의 존재 여부를 확인 후 배열리스트에 저장된 해당 라인의 구문 정보를 이용하여 자식 노드에 있는 값이 일반적인 SQL문으로 쓰여 있으면 1차적으로 보안약점으로 진단한다. 변수로 쓰여 있는 경우 저장되어 있는 라인 정보를 이용하여 해당 변수를 찾는다. 그 변수가 Statement 클래스로 선언되었다면 외부의 변수에 따라 SQL문이 생성되므로 보안약점으로 진단하고 구문 정보를 이용한 진단과 마찬가지로 코딩 규칙의 이름과 코드의 위치 정보를 출력한다.

V. Result of Benchmarks

본 절에서는 Juliet 테스트 모음을 이용하여 구현한 정적 분석 도구에 대한 성능 검증 결과를 기술한다.

Juliet 테스트 모음은 C/C++와 Java 언어 기반으로 작성되어 있으며, 테스트에 필요한 Java의 경우 112개의 CWE 항목, 25,000여개의 테스트 코드로 구성되어 있다. 구현한 도구를 검증하기 위한 CWE 항목을 Table 6.과 같이 정리하였다. 본 논문에서 CERT를 기준으로 구현한 항목은 10개이지만, Juliet 테스트 모음은 CWE를 기준으로 하였기 때문에 정확하게 일치하는 코딩 규칙이 없어 일치하는 코딩 규칙 있는 5개의 항목만을 테스트하였으며, 테스트 환경은 Eclipse에서 구현한 정적 분석 도구를 실행 후 Juliet 테스트 모음에 작성되어있는 코드를 명령행 인자로 사용하여 테스트를 진행하였다.

Table 6. Result using Juliet Test Suite

CWE List	Juliet Test-suit
CWE-89 : SQL Injection	183 / 3660
CWE-259 : Use of Hard-coded Password CWE-321 : Use of Hard-coded Cryptographic Key	125 / 240
CWE-338 : Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG)	34 / 34
CWE-395 : Use of NullPointerException Catch to Detect NULL Pointer Dereference	17 / 17
CWE-396 : Declaration of Catch for Generic Exception	34 / 34

Table 6.은 본 논문에서 구현한 정적 분석 도구가 CWE를 기준으로 Juliet 테스트 모음의 코드를 얼마나 검출하였는가에 대하여 표로 나타낸 것이다. 먼저 "Prevent SQL Injection"과 매칭되는 CWE-89의 Juliet 코드는 3660개가 있으며 183개의 코드만 검출하였다. 이와 같이 진단율이 현저하게 낮은 결과가 나타난 것은 본 논문에서 구현한 정적 분석 도구가 executeQuery 함수의 존재 여부를 확인 후 진단을 진행하는데 대부분의 Juliet 코드에서는 executeQuery 외에 다른 함수를 사용하여 SQL문을 실행하는 경우가 많았기 때문이다. "Never Hard code sensitive information"은 CWE-259. CWE-321 두 개의 CWE와 매칭되는데 총 240개의 코드 가운데 125개의 코드를 검출하였다. CWE-259와 CWE-321도 CWE-89와 같이 진단율이 낮은 현상이 있으며, 보안에 취약한 함수를 찾은 후 제어 흐름 정보를 이용하여 보안약점을 진단하는데 구현한 도구가 확인하는 함수의 양이 적기 때문에 이와 같은 결과를 보였다. 이는 구현한 도구에 검출할 수 있는 함수의 종류를 추가적으로 작성하여 진단율을 높일 수 있다. CWE-338은 "Generate strong random number"에 해당하는 보안약점으로, SecureRandom 클래스에 속한 메서드를 사용하지 않으면 보안약점으로 검출하였다. Util과 Math 클래스에 속한 random 메서드의 사용은 구문 정보를 이용하여 모든 Juliet 코드에 대해 보안약점을 검출하는 결과를 보인다. CWE-395, CWE-396에 해당하는 보안약점은 NullPointerException과 Exception만을 사용하여 catch 절에 작성한 경우 구문 정보를 이용하여 진단하는 보안약점이고 Juliet 코드에 대해 보안약점을 검출하였다.

VI. Conclusions

본 논문에서는 보안약점을 진단할 때 사용하는 여러 가지 정보 중 구문 정보와 제어 흐름 정보를 이용하여 JavaCC로 보안약점을 진단하는 도구를 구현하였으며, Eclipse 상에서 Juliet 테스트 모음을 이용하여 구현한 도구에 대한 성능 검증을 하였고 구현한 도구의 진단율을 보면 CWE-89의 진단율은 5%, CWE-259, CWE-321의 진단율은 50%, 그 외 구문 정보를 이용하여 진단하는 경우 100%의

진단율을 보였다. JavaCC는 파서 생성기이지만 개별적으로 배열 리스트에 저장하여 제어 흐름 정보를 이용할 수 있도록 구현하였다. 이를 통해 각 컴파일러 단계에서 얻을 수 있는 정보와 보안약점을 연계함으로써 보안약점에 대한 진단 방법과 프로그램 분석 정보의 연계를 기대할 수 있다. 또한 프로그래밍 언어 측면에서 보안약점에 대한 진단 방법을 접근함으로써 언어의 특성과 관련된 진단 기법 연구의 기반을 마련할 수 있다.

향후 컴파일러 단계에 따라 추출 가능한 정보 중 타입 정보와 값 정보를 이용하여 진단하는 도구를 구현하는 방법을 연구하는 방향으로 확장할 것이다. 또한 LLVM[15]을 사용하여 컴파일러 단계에서 얻을 수 있는 정보를 활용한 Clang의 확장 또한 기대할 수 있다.

REFERENCES

[1] SungMoon Hong, Seungcheol Shin, Kyung-Goo Doh, Detection of Security Vulnerability From the Knowledge-Base Representation of Source Code, Journal of The Korea Information Science Society pp.1618-1620, June 2014.

[2] CWE, Common Weakness Enumeration, *https://cwe.mitre.org/*

[3] CERT, Computer Emergency Response Team, *https://wiki.sei.cmu.edu/confluence/*

[4] Kyungsook Han, Damho Lee, Changwoo Pyo, Classificati on of Diagnostic Information and Analysis Methods for Weaknesses in C/C++ Programs, Journal of The Korea Society of Computer and Information Vol. 22 No. 3, pp. 81-88. March 2017.

[5] Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman, "*Compilers: Principled, Techniques, and Tools*", Addison Wesley, 1986

[6] JavaCC, *https://javacc.org/*

[7] Juliet test-suite, *https://samate.nist.gov/SRD/testsuite.php/*

[8] MICRO FOCUS Inc., *https://software.microfocus.com/*

[9] Sparrow Co., *https://sparrowpasso.com/*

[10] CODEMIND, *https://www.codemind.co.kr/*

[11] Minero Aoki, "*Compiler structure and principle : Language processing system learned by the compiler*", 2009

[12] C. Cadar, and K. Sen, "Symbolic execution for software testing: three decades late," Communications of the ACM, 56.2 pp.82-90, July 2013.

[13] P. Cousot, and R. Cousot, "Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints, "Proceedings of the 4th ACM SIGACT- SIGPLAN symposium on Principles of programming languages, pp238-252, ACM, January 1977.

[14] Kuznetsov, Volodymyr, Kinder, Johannes, Bucur, Stefan, Candea, George, "Efficient State Merging in Symbolic Execution", 2012

[15] LLVM, http://llvm.org/

Authors

Byeongcheol Kim will receive the B.S. degree in Computer Engineering from Korea Polytechnic University, in 2019. He is currently a student in the Department of Computer Engineering Korea Polytechnic University. He is interested in software security, static analysis.

Changjin Kim will receive the B.S. degree in Computer Engineering from Korea Polytechnic University, in 2019. He is currently a student in the Department of Computer Engineering Korea Polytechnic University. He is interested in software security, program development.

Seongcheol Yun will receive the B.S. degree in Computer Engineering from Korea Polytechnic University, in 2019. He is currently a student in the Department of Computer Engineering Korea Polytechnic University. He is interested in software security, program development.

Kyungsook Han received the B.S., M.S. and Ph.D. degrees in Computer Engineering from Hongik University, Korea, in 1993, 1995 and 2002, respectively Dr. Han joined the faculty of the Department of Computer Engineering at Korea Polytechnic University, Gyeonggi-Do, Korea, in 2003. She is currently an Professor in the Department of Computer Engineering, Korea Polytechnic University. She is interested in compiler optimization and software security,