정적 분석 툴의 비교: Lexical Analysis and Semantic Analysis

장성수*, 최영현*, 임헌정*, 엄정호*, 정태명**
*성균관대학교 전자전기컴퓨터공학부
**성균관대학교 정보통신공학부
e-mail: *{ssjang, yhchoi, hjlim99, jheom}@imtl.skku.ac.kr,
**tmchung@ece.skku.ac.kr

Comparison of Tools for Static Analysis: Lexical Analysis and Semantic Analysis

Seongsoo Jang*, Young-Hyun Choi*, Hun-Jung Lim*, Jung-Ho Eom*
and Tai-Myoung Chung**
*Dept. of Electrical and Computer Engineering, Sungkyunkwan Univ.
**School of Information Communication Engineering, Sungkyunkwan Univ.

요 약

오늘날 소프트웨어를 대상으로 하는 악성코드로부터의 공격이 잦아지면서, 소프트웨어 개발 프로세스에서부터의 보안 취약성 점검이 중요시되고 있다. 본 논문에서는 소프트웨어 보안 취약점 분석 기법 중 하나인 정적 분석에 사용되는 도구들을 살펴보고 비교하여 그 구조 및 특성을 분석·파악한다. 그리하여 우리의 궁극적 목표인 향상된 성능의 새로운 정적 분석 툴 개발의 기반을 마련하고자 한다.

1. 서론

소프트웨어를 대상으로 하는 악의적인 공격자 혹은 악성코드로부터의 공격이 증가하고 있다. 특히, 최근 모바일 어플리케이션 시장이 활성화되었지만, 아직 보안 및 소프트웨어 검증 절차가 허술하기 때문에 위험 요소를 내포한 소프트웨어가 유통되어 보안 사고를 일으킬 가능성이 있다. 따라서 소프트웨어 개발 프로세스에서부터 보안 취약점을 점검하여 안전한 소프트웨어를 개발해야할 필요성이 대두되고 있다.

소프트웨어의 보안 취약점을 점검하는 테스팅 방법은 블랙박스 테스트와(black-box test) 화이트박스 테스트(white-box test)의 두 가지로 구분될 수 있다. 블랙박스 테스트는 소프트웨어 내부 구조와 무관하게 소프트웨어가 목적으로 하는 동작만을 테스트하는 방법이고, 화이트박스 테스트는 소스코드를 기반으로 소프트웨어 내부 구조 이해를 통해 테스트를 수행하는 방법이다. 본 논문에서 우리는 화이트박스 테스트 기법 중 하나인 정적 분석에 사용되는 도구들을 비교·분석하여 그 특성을 파악함으로써 새로운 정적 분석 툴 개발의 기반을 마련하고자 한다.

본 논문의 구성은 다음과 같다. 2장에서 관련 연구로써 정적 분석 기법에 대해 간략히 살펴보고, 3장에서 정적 분석 툴의 특성을 비교, 분석한다. 마지막으로 4장에서 결론과 함께 향후 연구 방향에 대해 기술한다.

본 논문은 중소기업청에서 지원하는 2010년도 산학연공동기술개발사업(No. 00044301)의 연구수행으로 인한 결과물임을 밝힙니다.

2. 정적 분석 기법

정적 프로그램 분석(static analysis)은 프로그램을 실행시키지 않고 소스 코드 혹은 바이너리 코드를 조사함으로써 보안 취약점을 찾아내는 프로그램 분석 기법이다. 정적 분석 기법은 대상 프로그램을 가상 머신에서 실제로 실행시키며 그 결과를 관찰하는 동적 프로그램 분석 기법(dynamic analysis)과 비교해 다음과 같은 특징이 있다.

동적 분석 기법은 소스 코드가 주어지지 않더라도 프로그램의 분석이 가능하며, 입력 값을 실제로 대입하며 프로그램을 실행하기 때문에 긍정 오류(false positive)와 부정 오류(false negative)의 발생이 적다는 특징이 있다. 그러나 프로그램 실행 비용(execution overhead)이 크며, 모든 테스트 케이스의 수집이 사실상 어렵다는 단점이 있다.

이에 반해, 정적 분석 기법은 오버헤드가 적고, 대상 프로그램의 실행 중에 발생할 수 있는 모든 경우에 대해 점검할 수 있다. 또한, 다양한 자동화 도구가 제공되기 때문에 최근 소프트웨어 테스트 공정에서 널리 사용된다.

2.1 어휘 분석

어휘 분석(lexical analysis)[2]은 변수명, 예약어 등 프로그램 본문의 일련의 연속된 문자열 단위인 토큰(token)을 분석하는 정적 분석 방법이다. 대표적인 기법인 코딩 룰 검사(coding rule check)는 scanf(), sprintf(), strcpy()

<표 1> Comparison of Static Analysis Tools

Name	Year	Target	Platform	Features	Check list
ITS4	2000	C/C++ source code	Unix, Windows	lexical analysis	buffer overflows, race conditions
Coverity Prevent	2004	C/C++, Java, C# source code	Linux, Mac OSX, Solaris, Windows, etc.	statistical analysis, interprocedural analysis, false path pruning, incremental analysis	memory leak, buffer overflows, format string vulnerabilities, runtime exceptions, etc.
Polyspace C Verifier	2004	C/C++ source code	Windows, Linux, Solaris	abstract interpretation, MISRA-C, MISRA-C++	overflow, divide-by-zero, out-of-bounds array access, run-time errors
Splint	1994	C source code	Unix, Linux, FreeBSD, OS/2, Solaris, Win32	lightweight analysis, annotation-assist	memory leak, buffer overflows, format bugs, etc.

등 잠재적인 위험 요소를 가진 함수들의 호출 여부를 판단하는 기법으로 실행 속도가 빠르다는 장점이 있다.

2.2 프로시저 간 분석

프로시저 간 분석(interprocedural analysis)[2]은 의미 분석(semantic analysis) 방법 중 한 가지로, 프로그램 분석 시 실행되는 함수들 간의 앞 뒤 문맥을 고려하는 문맥 인식(context-sensitive) 분석 기법이다. 정적 분석 툴로 하여금 프로그램의 가능한 모든 실행 경로를 고려하여 더욱 정확한 분석을 할 수 있게 해준다. 프로시저 간 분석은 함수 호출자(caller)로부터 피호출자(callee)에게로, 혹은 그 역방향으로 전달되는 데이터의 흐름을 따라가면서 프로그램 전체 코드를 고려하여 분석함으로써 어휘 분석보다 안전한 분석 결과를 보장할 수 있다.

2.3 요약 해석

요약 해석(abstract interpretation)[1]은 의미 분석의 이론적 방법으로, 프로그램 실행의 실제 도메인(domain) 대신 프로그램 실행 시 나올 수 있는 모든 결과 값이 반영된 요약 도메인(abstract domain)을 계산하는 기법이다.

쉬운 예로, "4×24+16×(−8)=?"이라는 정수 식 계산 문제가 주어졌을 때, 홍길동 학생이 "정답은 짝수"라고 대답하는 것을 틀렸다고 할 수 없다는 것이다. 문제의 정확한 답은 "-32"이지만, "짝수×짝수+짝수×짝수=짝수"라는 계산을 통해 나온 "짝수"라는 결과 값의 도메인이 정확한 답 "-32"를 포함하고 있기 때문이다.

이렇듯, 요약 해석은 주어진 프로그램을 실행시키지 않고도 요약 도메인을 통해 모든 실행 결과 값을 고려하게 되어 안전한 분석을 가능하게 한다. 요약 해석에서는 긍정 오류 및 부정 오류의 발생과 밀접한 연관이 있는 요약 도메인 계산의 정확성의 정도가 주요 고려 사항이다.

3. 정적 분석 툴 비교

본 논문에서는 현존하는 다양한 정적 분석 툴 중 어휘 분석을 사용하는 분석 툴과 의미 분석을 사용하는 분석 툴을 살펴보고 비교함으로써 그 구조 및 특성을 분석·파악하고자 한다. 어휘 분석 툴 중 ITS4[6][9]를, 의미 분석 툴 중 Coverity Prevent[3], Polyspace C Verifier[5][7], 그리고 Splint[8]를 분석 대상으로 선정하였다. <표 1>에서 보듯, 이들 분석 툴은 서로 다른 다양한 특성을 지니고 있어, 우리의 궁극적 목표인 향상된 성능의 새로운 정적 분석 툴을 제안하기 위한 추후 연구에 많은 도움이 될 것으로 예상한다.

3.1 ITS4

ITS4(It's the Software Stupid! Security Scanner)는 FlawFinder, RATS[10] 등과 함께 널리 알려진 어휘 분석 기반 정적 분석 툴이다. 2000년에 개발된 ITS4는 Unix와 Windows 기반에서 C/C++ 소스 코드를 분석한다. ITS4 이전의 분석 툴은 모호하지 않은 문법의 일부에 대해서만 파싱을 하는 전통적인 파서를 사용하기 때문에 많은 부정 오류가 발생한다. 그러나 ITS4는 전통적인 파서를 사용하지 않고 전체 코드를 단순히 탐색하여 생성된 토큰을 취약성을 내포하는 함수들이 저장된 내장 데이터베이스와 비교하는 방식을 사용한다. 따라서 버퍼 오버플로(buffer overflow) 및 경쟁 상태(race condition)의 취약성 등을 탐색할 수 있지만, 데이터베이스에 의존하기 때문에 알려지지 않은 패턴에 대해서는 검사가 어려운 단점이 있다.

3.2 Coverity Prevent

미 Coverity사의 Prevent는 삼성, 인텔 등 전 세계의 대기업에서 치명적인 버그 및 보안 취약점 검사에 널리 쓰이는 시장 점유율 1위의 상용 정적 분석 툴이다. Linux,

Windows, Solaris, Mac 등 다양한 플랫폼에서 C/C++, C# .NET, Java 등의 다양한 소스 코드를 분석할 수 있다. Prevent는 다양한 분석 엔진을 사용하여 정확하고 정밀한 분석을 제공하고, 분석시간을 단축하는 등 우수한 성능을 보인다. 제어 흐름(control flow)를 따라 프로시저 간 분석을 하여 모든 경로와 데이터 값을 추적하고, false path pruning 엔진이 실행되지 않는 경로를 분석에서 제외시켜 긍정 오류 발생의 가능성을 낮춘다. 제외된 부분은 소프트웨어 DNA 지도에서 통계적 분석(statistical analysis)을 통해 소스 코드 상에서 직접 호출되는 API의 취약성을 검사하는 방식으로 분석이 이루어진다. Incremental analysis 엔진은 한 번 분석이 끝난 프로그램을 두 번째 분석부터는 버그 및 취약점이 수정된 부분만 분석하여 분석 시간을 단축시킨다. 2004년 처음 릴리즈(release) 된 Coverity Prevent는 현재 버전 5까지 출시되었다.

3.3 Polyspace C Verifier

Polyspace C Verifier는 2004년에 처음 릴리즈 되어 유럽 항공사와 우주 계획에서 런타임 오류 검사에 사용되었다. Windows, Linux, 그리고 Solaris 플랫폼을 지원하는 C Verifier는 요약 해석을 기반으로 하여 C/C++ 소스 코드를 검증한다. 특히, 자동차, 우주 항공, 통신, 의료 등 제조 분야에서 사용되고 있는 "MISRA(Motor Industry Software Reliability Association)-C/C++" 산업 표준에 의거하여 overflow, divide-by-zero, out-of-bound 등의 취약점을 검사한다. Polyspace C Verifier는 현재 Polyspace Server for C/C++라는 이름으로 변경되어 버전 8.0까지 출시되었다. Polyspace C Verifier는 요약 해석을 사용함으로써 정확성 높은 안전한 분석을 가능하게 한다. 그러나 요약 해석의 특성상 속도가 매우 느리고, ANSI-C 언어를 완벽하게 지원하지는 못한다는 단점이 있다.

3.4 Splint

1994년, 논문 [4]에서 David Evans에 의해 소개된 LCLint는 2002년 Splint(Secure Programming Lint)로 개명되어 현재 3.1.2 버전까지 출시되고 있다. Splint는 Unix 시스템에는 소스 코드 형태로 제공되고, 그 밖의 여러 플랫폼에는 바이너리의 형태로 제공되어 C로 작성된 소스 코드 분석에 사용된다. Splint는 오버헤드가 큰 프로시저 간 분석 대신 각각의 프로시저에 대해 독립적으로 경량 분석(lightweight analysis)을 사용한다. 프로시저의 엔트리 포인트(entry point)에서 전역 변수와 입력 인자들의 값을 참이라 가정하고, 프로시저 종료 시점의 반환 값과 주석을 참고하여 함수 본문을 검사한다. 메모리 누수(memory leak), 버퍼 오버플로(buffer overflow) 등의 다양한 취약점에 대해 빠르고 안전한 분석을 제공하지만, 제어 흐름이 닿지 않는 부분에 대해서는 점검을 하지 못하고 긍정 오류를 발생할 가능성이 있다.

4. 결론 및 향후 연구

본 논문에서는 몇 가지 어휘 분석 툴과 의미 분석 툴의 구조, 분석 방식, 장·단점 등을 비교함으로써, 각각의 특징을 알 수 있었다. 어휘 분석 기법을 사용하는 툴은 수행 속도가 빠르지만 취약점 분석에 있어 많은 한계가 있고, 의미 분석 기법을 사용하는 툴은 정확하고 정밀한 분석이 가능하지만 속도가 느리고 실행 오버헤드가 크다는 단점이 있다. 뿐만 아니라, 분석 메커니즘의 비교를 통해 정적 분석 툴의 성능을 좌우하는 요소에 대해서도 알 수 있었다. 제어 흐름을 어떻게 참고할 것인지, 또 그에 따른 후속 조치를 어떻게 할 것인지에 따라 서로 상충 관계에 있는 긍정 오류와 부정 오류 발생의 조절이 가능하다는 점은 추후 연구에 많은 도움이 될 것이다.

본 논문에서 파악한 정적 분석 툴들의 특징을 통해 우리의 궁극적 목표인 향상된 성능의 새로운 정적 분석 툴 개발의 기반을 마련할 수 있을 것으로 기대된다.

추후, 강력한 성능을 보이는 프로시저 간 분석 방법과 요약 해석 기법에 대한 연구를 계속하여 알고리즘을 명확히 파악하고, 그 한계점을 보완할 수 있는 방안을 마련할 예정이다. 그래서 현재 문제가 되는 수행 속도 향상 및 오버헤드를 줄이고, 더 나은 성능을 발휘할 수 있는 기법을 제시하기 위한 노력을 할 것이다.

참고문헌

[1] 정영범, 김재황, 신재호, 이광근, "자동 오류 검출을 위한 프로그램 분석기 - 아이락", 마이크로 소프트웨어, 마소인터렉티브, pp. 178-186, Jun. 2005

[2] Alfred V. Aho, Monica S. Lam, Ravi Sethi and Jeffrey D. Ullman, "Compilers - Principles, Techniques, and Tools", 2nd Edition, Pearson Education, Inc., 2007

[3] Coverity, http://www.coverity.com/

[4] David Evans, John Guttag, Jim Horning and Yang Meng Tan, "LCLint: A tool for using specifications to check code", SIGSOFT Symposium on the Foundations of Software Engineering, Dec. 1994

[5] Hote C., "Extension of Static Verification Techiniques by Semantic Analysis", Digital Avionics Systems Conference, 2005. DASC 2005. The 24th, 2005

[6] ITS4, http://www.cigital.com/its4/

[7] Polyspace, http://www.mathworks.com/products/polyspace/

[8] Splint, http://www.splint.org/

[9] Viega J., Bloch J.T., Kohno T. and McGraw G., "ITS4: A Static Vulnerability Scanner for C and C++ Code", Proceedings of the 16th Annual Computer Security Applications Conference, Dec. 2000

[10] Zitser M., Lippmann R., and Leek T., "Testing static analysis tools using exploitable buffer overflows from open source code", SIGSOFT'04/FSE- 12, 2004