룰 시그니처를 이용한 소프트웨어 보안성 검증 시스템 개발

장희진[0*] 김완경* 소우영*
*한남대학교 컴퓨터공학과
*{zzang, wankk12, wsoh}@neuro.hannam.ac.kr

Development of a Software Security Verification System Using Rule Signatures

Huijin Jang[0*], Wankyoung Kim*, Wooyoung Soh*
*Dept of Computer Engineering, Hannam University

요 약

프로그래밍 기술과 인터넷 통신의 발달로 인하여 보안성이 검증되지 않은 다양한 프로그램들이 생성되고 쉽게 유포되어 보안 취약성으로 인해 야기되는 다양한 문제의 심각성이 더해가고 있다. 따라서 사용자가 보안상 안전하게 사용할 수 있는 소프트웨어 인증절차가 필수적으로 요구되고 있는데, 이를 해결하기 위해 소프트웨어 안전성 평가에 대한 연구가 진행 중이지만, 기존의 방법들은 특정 영역에 한정적이어서 일반적인 소프트웨어의 보안성 평가(security evaluation) 방법으로써 부적합하다. 뿐만 아니라 기존의 시스템들은 단순 패턴매칭에 기반을 두고 있어 오용탐지가 크고 정확성이 떨어진다는 문제점을 가지고 있다. 따라서 본 논문에서는 이러한 문제점들을 해결하기 위해 악성프로그램 코드의 구조와 흐름을 분석하여 규칙으로 정의하고 그 규칙에 따라 검사 대상 프로그램 코드에서 악성코드와 취약점 흐름을 탐지하는 규칙 기반의 소프트웨어 보안성 검증 시스템 프로토타입을 제안한다. 제안한 검증 시스템의 프로토타입은 악성코드와 소프트웨어 취약성을 동시에 탐지하여 보안성을 평가함으로써 범용적인 소프트웨어 평가에 활용 가능할 것이다.

1. 서 론

컴퓨팅 환경에서 보안성이 검증되지 않은 다양한 프로그램들이 생성되고 쉽게 유포되어 소프트웨어에 대한 보안 취약성으로 인하여 야기되는 다양한 문제의 심각성이 더해가고 있다. 악의의 목적을 가지고 정보 및 소프트웨어를 배포할 경우 엄청난 피해를 초래하게 된다. 따라서 소프트웨어에 대한 보안성 및 신뢰성을 검증할 수 있는 시스템이 필요하다. 또한 개발자의 안일한 보안 의식으로 인해 발생될 수 있는 소프트웨어 취약성을 사전에 검사함으로써 해킹의 위험을 줄일 수 있는 자동화된 검증 방법 필요하다. 기존의 안전성 검증 시스템들은 악성코드(malicious code) 탐지, 바이러스 탐지, 소프트웨어 취약성(software vulnerability) 탐지가 각각 분리되어 존재하고 있어 실제 소프트웨어 내부에 포함되어 있는 악의적인 코드를 검사할 수 없으며, 이에 대한 보안성을 제대로 검사할 수 있는 환경을 제공하지 못하고 있다. 또한 기존의 탐지 기법들은 대부분 패턴 매칭에 의존하고 있어 새로운 악성코드에 대한 시그니처를 업데이트하기 전에는 악성코드를 탐지 할 수 없게 된다. 그리고 실제 소스코드 내부에 악의의 목적을 가지고 포함되어 있는 악성코드나 개발자의 실수로 인해 내포 되어 있는 소프트웨어 취약성을 검사할 수도 없다.

따라서 본 논문에서는 소스코드 내부에 심어져 있는 악성코드 및 소프트웨어 취약성을 탐지하여 소프트웨어에 대한 보안성을 사전에 평가할 수 있는 검증 시스템을 제안한다. 또한 제안 시스템의 프로토타입은 탐지 대상의 구조를 분석하고 이를 규칙으로 정의함으로써 기존의 패턴 매칭 기법 시그니처 기반에서 탐지하지 못했던 새로운 악성코드를 효과적으로 탐지할 수 있을 것이다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구에 대해 이야기 하고, 3장에서는 본 논문에서 제안한 검증 시스템의 구조와 규칙 기반의 탐지 기법에 대해 기술하고, 4장에서는 기존 탐지 도구와 제안 시스템을 비교 분석한다. 마지막으로 5장에서 결론을 맺는다.

2. 관련 연구

2.1 ITS4

C와 C++의 정적 취약점 점검 스캐너인 ITS4는 C와 C++ 소스코드의 보안 취약점을 찾아주는 커맨드 라인의 도구로 전체 소스코드를 스캔하여 잠재적인 위험성이 있는 코드를 찾아주는 도구로써, 문제점 데이터베이스를 자체적으로 관리하여 토큰단위로 패턴 매칭하여, 문제점에 관한 간단한 설명과 등급별로 문제점을 분류하여 알려준다[1]. 최초 ITS4는 프로그래머들이 보안에 대해서 신경 쓰지 않고 개발할 수 있는 환경을 제공하기 위해 만들어지기 시작하였다. 즉, 프로그램을 작성한 후 취약성 분석에 투자하는 시간을 줄이기 위해 만들어지기 시작한 것이다.

2.2 MOPS

본 연구는 산업자원부 지역혁신센터사업
(R12-2003-004-01002-0) 지원으로 수행되었음

MOPS는 C 프로그램에서 보안에 대한 버그들이나 결함들을 찾아 안전성을 검증하기 위한 시스템이다. 보안에 대한 버그들은 root권한을 획득하거나, 버퍼오버플로우 공격 등을 일으켜 일시적 안전성들을 위반하게 된다. 프로그램에서 모든 코드경로에 대해 이러한 안전성 위반사항들을 직접 일일이 검사한다는 것은 복잡하고 어려운 일이다. 따라서 MOPS는 이러한 검사과정을 모델 검사와 정적 분석 기술을 이용하여 자동화한다. 먼저 사용자가 보안에 대한 안전한 속성들을 기술한다. 그 다음, 대상 C 소스코드에서 이러한 속성들을 위반하는지 검사하기 위해 MOPS를 실행한다. 대상 프로그램이 이러한 속성들을 위반한다면, MOPS는 그 대상 프로그램에서 안전성을 위반하는 코드경로를 출력해준다[2,3,4].

3. 소프트웨어 보안성 검증 시스템

본 논문에서 제안하고 있는 소프트웨어 안정성 검증 시스템은 악의적인 목적을 가진 악성코드가 소스코드에 포함되어 시스템에 설치됨으로써 정보의 유출 및 해킹의 우려가 있는 소프트웨어에 대한 보안성을 평가한다. 기존의 악성코드의 구조를 분석하여 규칙을 생성하고 이를 바탕을 악성코드를 탐지함으로써 새로운 악성코드를 효과적으로 탐지 가능하며, 더불어 개발자의 부주의로 발생되는 소프트웨어 취약성을 검증해줌으로써 신뢰할 수 있는 소프트웨어를 사용할 수 있는 환경을 제공한다.

3.1 소프트웨어 보안성 검증 시스템의 구성

제안한 검증 시스템은 소스코드 분석 모듈과 이를 활용하여 규칙을 생성하는 악성코드 규칙 생성 모듈, 소스코드 분석 모듈에서 생성된 정규형 표현을 통하여 규칙과 취약성 함수 리스트를 비교하여 검증하는 검사 모듈로 구성된다. 소프트웨어 보안성 검증 시스템의 전체 구조는 그림 1.과 같다.

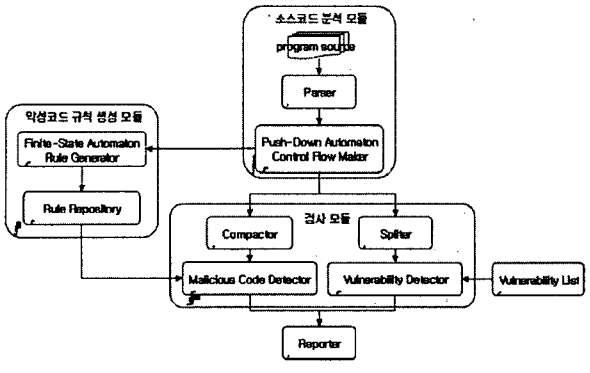

그림 1. 소프트웨어 보안성 검증 시스템의 구조

3.2 악성코드 규칙 생성 모듈

대상코드에서 악성코드가 포함되어 있는지를 검사하기 위해서는 검사 기준으로 제시되어야 할 악성코드에 대한 규칙이 필요하다. 즉 악성프로그램의 코드에서 악성행위를 하는 기능에 대한 구문의 규칙들이 생성되어져야 한다. 이 구문의 규칙에 따라 생성된 악성 코드의 규칙은 악성행위를 하는 코드가 어떤 이름으로 명명되는지 그리고 어떤 실행 흐름을 가지는지에 대한 것이다. 즉, 악성코드가 수행되는데 필요한 함수들의 리스트와 함수가 변경하는 상태 변수에 대한 정보이다. 여기서 함수의 리스트는 단순한 함수의 나열이 아닌 처리 흐름을 반영하고 유지할 수 있도록 구성되어 있어야 한다. 왜냐하면 처리 흐름이 반영되지 않을 경우는 단순한 사용함수에 대한 이름 밖에 없기 때문에 악성 코드 탐지에 활용될 수 없다. 이는 악성 코드 검사에서 요구하는 처리 흐름 정보를 제공하지 못하기 때문이다.

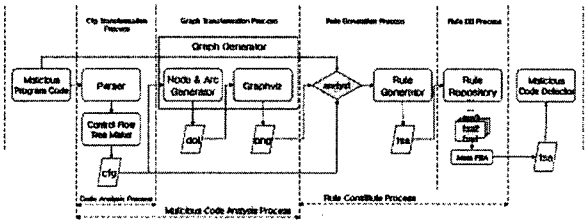

그림 2. 악성 코드 분석 및 규칙 생성 과정

악성 코드 분석 모듈의 처리 흐름에 따라 프로세스별로 나누면 그림 2.와 같이 크게 규칙을 생성하기 전에 수행해야 할 악성 코드 분석 프로세스와 실질적으로 규칙을 만들고 저장·관리하는 규칙 형성 프로세스로 나누어진다.

악성프로그램 코드는 Parser를 통해 컴파일된 후, Control Flow Maker가 분석된 정보를 토대로 표현식, 문장, 선언, 블록, 프로시저 흐름으로 기술한다. 생성된 중간 코드(cfg)는 dot 파일로 변환 후, 실행 처리 흐름을 표현한 그래프 구조로 변환한다. 이때, 규칙 생성은 기존 악성 프로그램에 대한 여러 가지 정보를 바탕으로 검수자 혹은 악성 프로그램에 대한 전문적 지식을 가지고 있는 분석가가 악성 프로그램의 코드, 중간 코드, 실행 처리 흐름을 가진 그래프를 분석하여 FSA(Finite State Automata)로 정규화하여 생성한다.

3.3 소스코드 분석 및 검사 모듈

소스코드 분석 모듈은 프로그램 안의 표현식(expression), 문장(statements), 선언문(declarations), 블록(blocks) 그리고 프로시저(procedures) 등을 각기 구별하고 파싱한다. 파싱 과정에서 생성되는 각각의 구문들은 Control Flow Maker에 전달되어 PDA(push-down automaton)에 의해 그 처리 흐름을 정규형 코드로 표현하여 기술하게 된다. 각 토큰을 표현식, 문장, 선언, 블록, 프로시저 흐름으로 구분한다. 토큰과 토큰의 정보를 기반으로 Control Flow Maker가 토큰 유형에 해당되는 지시자를 붙이고 이를 이용하여 프로그램의 흐름을 기술한다. Parser와 Control Flow Maker는 대상코드뿐만 아니라 규칙 생성을 위한 악성코드의 분석에도 동일하게 적용된다.

검사 모듈은 그림 1.에서 보는 바와 같이 소스코드 내부에 포함되어 있을지 모를 악성코드를 탐지하는 Malicious Code Detector 부분과 소스코드 내부에 존재

하는 취약성을 탐지하는 Vulnerability Detector 부분으로 구성된다. 두 개의 세부 모듈들은 소스코드 분석 모듈에 의해 생성된 정규형 코드(cfg)를 활용하여 검사를 위한 정규형 코드(fsa)로 변경 후, Rule Repository에 저장되어 악성코드의 규칙과 Vulnerability List에 저장되어 있는 취약성 리스트를 매칭함으로써 악성코드와 취약성을 탐지하고 결과를 보고하게 된다.

그림 3.은 소스코드 분석 및 검사 모듈의 처리 흐름을 나타낸다.

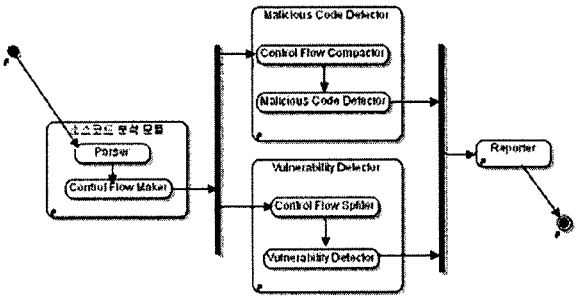

그림 3. 소스코드 분석 및 검사 모듈의 처리 흐름

제안한 소프트웨어 검증 시스템에서 검사를 수행하면 규칙에 위반한 코드의 경로를 탐지하고 결과를 통하여 실제 코드에서 위반 경로를 그림 4.와 같은 GUI를 통해 출력하여 준다.

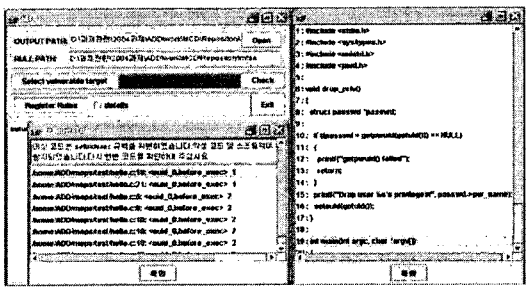

그림 4. 보안성 검증 시스템의 GUI 실행 화면

4. 결론

기존의 ITS4, MOPS는 특정 부분에 한정되어 사용되는 시스템이지만, 제안된 시스템은 악성코드 및 소프트웨어 취약성 전반에 걸쳐 탐지가 가능하기 때문에 다른 시스템에 비해 소프트웨어의 보안성을 효과적으로 평가할 수 있다. 뿐만 아니라 단순 패턴 매칭 시스템과 달리 악성코드의 규칙에 기반을 두고 있어 새로운 악성코드의 탐지가 용이하다. 새로운 형태의 악성코드가 출현하더라도 규칙 프로세스를 이용하여 규칙을 생성함으로써 효율적인 감사 도구로 활용 가능하다. <표 1>은 제안한 시스템과 기존의 취약성 탐지 시스템을 비교한 결과이다.

표 1. ITS4, MOPS, 제안시스템의 비교

구분	ITS4	MOPS	제안시스템
탐지 대상	소프트웨어 취약성	security property	악성코드 및 소프트웨어 취약성
탐지 기법	패턴 매칭 기반 탐지	Model 기반 탐지	규칙 기반 탐지
탐지 시점	컴파일 이전 탐지	컴파일 이후 탐지	컴파일 이후 탐지

본 논문에서는 소스코드 내부에 악의의 목적으로 포함될 수 있는 악성코드 및 소프트웨어 취약성을 사전에 탐지하여 보안성을 평가할 수 있는 소프트웨어 보안성 검증 시스템을 제안하였다. 제안된 시스템 대상코드의 프로그램 구조를 분석하여 규칙을 생성함으로써 단순 패턴 매칭으로 인해 새로운 악성코드에 대처할 수 없었던 기존 탐지 시스템의 문제점을 해결하고 소프트웨어 취약성 탐지 모듈을 통합함으로써 보다 효과적으로 소프트웨어의 보안성을 평가할 수 있게 되었다.

향후 연구 과제로는 다양한 악성코드에 따라 소프트웨어를 어떻게 체계적으로 인증 평가를 수행할 것인지에 대한 방법론에 대한 연구가 이루어져야 할 것이다

참고문헌

[1] John Viega, J.T. Bloch, Tadayoshi Kohno, Gary McGraw Reliable Software Technologies, "ITS4: A Static Vulnerability Scanner for C and C++ Code", http://www.rstcorp.com

[2] Hao Chen, David Wagner and David Schultz at Computer Science Division, UC Berkeley, "MOPS User's Manual"

[3] Hao Chen and David Wagner University of California at Berkeley "MOPS: an Infrastructure for Examining Security Properties of Software"

[4] Hao Chen, David Wagner, and Robert Johnson, "Model Checking Software for Security Violations", Short talk at 2001, IEEE Symposium on Security and Privacy.

[5] Thomas Ball, Sriram K. Rajamani, "Automatically Validating Temporal Safety Properties of Interfaces", SPIN 2001, Workshop on Model Checking of Software, LNCS 2057, May 2001, pp. 103-122.

[6] Thomas Ball, Sriram K. Rajamani, "Boolean Programs: A Model and Process for Software Analysis", MSR Technical Report 2000-14.