보안 취약점 검사를 위한 AOP 기반의 정적/동적 분석의 통합

서광익○ 최은만
동국대학교 컴퓨터공학 전공
{bradseo, emchoi}@dongguk.edu

Integration of Static and Dynamic Analysis based AOP for Security Vulnerability Analysis

Kwang Ik Seo○, Eun Man Choi
Dongguk University, Department of Computer Engineering and Science

요 약

최근 DDoS와 같은 악의적 인터넷 공격의 횟수가 증가하고 그에 따른 피해도 커지고 있다. 이를 방지하기 위해 개발 초기 단계에서부터 프로그램 코드를 분석하여 보안의 취약점을 제거하려는 노력을 하고 있다. 하지만 대부분은 개발 초기의 빠른 적용과 분석 효율성을 강조하여 정적 분석에만 집중하고 있어 실행 시점에서 발생할 수 있는 취약점은 정확히 예측하기 어렵다. 따라서 본 연구는 이러한 제약사항을 해결하기 위해 AOP를 이용하여 정적 분석과 동적 분석의 통합이 가능하면서 동시에 분석 모듈의 확장성을 높일 수 있는 방법을 제안한다.

1. 서 론

네트워크와 인터넷의 발달로 인해 모든 단말기와 소프트웨어가 통신하는 환경에서 소프트웨어 취약점을 이용한 공격이 급증하고 있다. 최근 Gartner의 사이버 위협의 동향 보고는 어플리케이션 취약점을 이용한 공격이 75%에 이르고 있다고 전한다[1]. 이러한 보고는 소프트웨어의 취약점을 점검하고 개선하여 안전한 시스템을 구축하는 것이 필요하다는 것을 말한다.

보안을 위한 취약점을 점검하는 방법은 크게 정적 분석 방법과 동적 분석 방법이 있다. 정적 분석 방법은 애플리케이션을 실행시키지 않고 분석하는 방법을 말한다. 반면 동적 분석 방법은 프로그램을 실행하면서 분석하는 방법이다. 주로 실행 중인 프로그램을 모니터링하거나 성능을 평가하기 위해 사용한다. 동적 분석을 하기 위해서는 실행이 가능한 프로그램 코드가 필요하다. 또한 프로그램을 동작시키면서 분석해야 하기 때문에 많은 비용이 요구된다. 따라서 개발 초기에 적용이 가능하면서 상대적으로 적은 비용으로 분석할 수 있는 정적 분석이 주로 사용된다.

취약점 검사를 하는데 있어 코드 분석을 일일이 수동으로 하는 것은 한계가 있기 때문에 도구를 사용할 수밖에 없다. 그러나 대부분의 도구는 고가(高價)의 외산이 시장을 선점하고 있다. 또한 이러한 도구는 개발하고자 하는 특정 도메인에 응용하기 위한 확장성에 많은 제한이 있다. 예를 들어 분석하고자 하는 언어에 따라 분석 방법이나 규칙이 달라져야 한다. 따라서 보안을 위한 취약점 분석을 위해 언어와 시스템의 특성에 맞도록 유연하면서 확장성이 좋은 분석 방법이 필요하다. 그러나 직접 분석 도구를 개발하는 것은 많은 시간과 노력이 필요하기 때문에 개발하고 있는 코드를 대상으로 정적 분석하기 위한 도구를 따로 개발하는 것은 본 업무에 많은 지장을 초래하게 된다. 정적 분석 도구의 단점은 이 뿐만이 아니다. 정적 분석 방법은 실행 중인 프로그램에 대한 예측을 전혀 할 수가 없다는 것이다. 이는 정적 분석 방법의 가장 큰 단점이자 한계점이라 할 수 있다. 따라서 이를 보완하기 위해서는 후속적으로 반드시 동적 분석 방법이 필요하다. 즉 정적 분석 방법으로 검사하지 못한 부분에 대해서는 프로그램을 실행하면서 동적인 분석을 해야 한다.

본 논문은 보안을 위한 취약점을 분석하는데 있어 확장성을 높이면서 동시에 정적 분석 도구에 대한 문제점을 해결할 수 있는 대안으로 AOP(Aspect-Oriented Programming)를 이용한 정적/동적 분석 통합 방법을 제안한다. AOP의 이용은 분석 모듈을 소스 코드와는 별개로 쉽게 추가, 수정, 삭제할 수 있어 확장성을 높일 수 있다. 또한 AOP 언어의 문법적인 이해만 있으면 프로그램을 개발하는 과정에서 직접 분석 코드를 추가할 수 있어서 개발 환경에 맞도록 수정이 가능하다. 뿐만 아니라 프로그램 실행 시점에서의 감시와 접근이 가능하므로 동적 분석이 모두 가능하다. 이로 인해 정적 분석의 한계점을 보완한다.

2. 관련 연구

2.1 정적 분석과 동적 분석

정적 분석은 프로그램을 실행하지 않고 그 자체를 분석하여 소스 코드에 내재한 논리적이고 문법적인 속성을 분석한다. 즉 컴파일 시점에서 프로그램을 분석하고, 그 결과를 확인하는 방법이다[2]. 반면 동적 분석은 프로그램을 실행하면서 프로그램의 속성을 분석하는 것이다[3]. 표 1은 두 분석 방법의 차이점과 장단점을 나타낸다.

표 1 정적 분석과 동적 분석 비교

	정적 분석	동적 분석
장점	높은 커버리지	정확성
단점	추측성(Conservative)	입력 의존성
분석범위	프로그램 코드 전체	실행 시점의 부분
정확도	근사치	실행에 근거
분석방법	일회 코드 분석	반복 실행과 분석
사용기술	소스/바이너리 코드 분석	분석 코드 삽입
분석수준	코드 분석	실행 프로그램 분석
분석시점	컴파일 시점	운용 시점

2.2 AOP(Aspect-Oriented Programming)

AOP 핵심 관심에 산재되어 있는 횡단 관심을 분리하여 다룰 수 있는 방법을 제공한다[4]. 그림 1은 핵심 관심과 횡단 관심의 예이다. 핵심 관심은 계좌이체와 입출금 그리고 이자계산과 같은 주요 핵심 기능이나 모듈이다. 횡단 관심은 모든 핵심 관심에 필요한 로깅, 보안, 트랜잭션이 각 핵심 모듈에 횡단으로 흩어져 있다. 이러한 경우 AOP는 계좌이체, 입출금, 이자계산을 중심으로 모듈화 되어 있는 시스템을 로깅과 보안 그리고 트랜잭션을 구현하고 있는 코드부분을 횡단으로 분리할 수 있다.

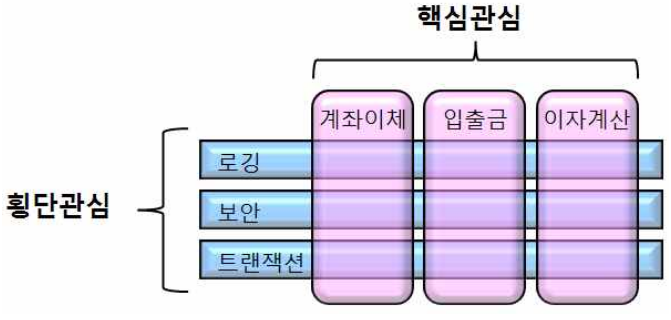

그림 1 횡단 관심사(cross-cutting concern)

2.3 취약점 검사를 위한 정적 분석

최근 프로그램 코드에 잠재 가능한 취약점에 대한 연구에 대해 관심이 많아지고 있다.[5, 6, 7]. 취약점 분석을 위해 정적 분석을 이용하여 코드를 분석하되, 외부의 공격으로부터 안전성을 확보하고 있는지 검사하는 시큐어(Secure) 정적 분석 기법이 있다[8, 9, 10].

시큐어 코드 정적 분석은 주로 그림 2와 같은 방법을 적용한다. 소스코드나 바이너리 코드를 시큐어 점검 규칙(Rule)에 따라 분석하고 점검 규칙을 위배하는 코드를 취약점으로 분류한다. 즉 분석하기 위한 규칙을 시큐어 관점에서 재정의하여 정적 분석을 하는 것이다. 하지만 정적 분석은 실행 시점의 정확한 취약점을 발견하는데 한계가 있다. 따라서 정적 분석의 단점을 보완하는 동적 분석이 필요하다.

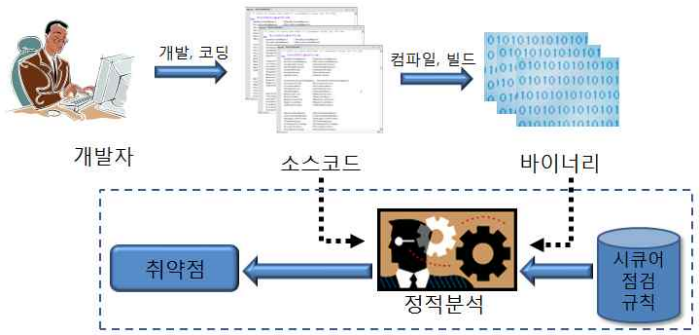

그림 2 시큐어 정적 분석

3. 정적과 동적 분석 방법

본 논문에서 제안하는 동적 분석 방법은 CERT의 10. IDS에서 제시한 취약성 발생 가능 위치인 핫 스팟(hot spot)을 파악하여 컴파일 시점에서 개발자에게 알려준다. 그리고 핫 스팟에 AOP로 정의한 인스투르먼트(Instrument)가 삽입되고, 실행 시점에서 핫 스팟을 모니터링 한다.

3.1 접근 방법

3.1.1 핫 스팟 검사

컴파일 시점에서 소스 코드의 취약점 발생이 가능한 위치와 종류와 함께 경고(warning) 메시지를 보여준다. 이를 통해 코드에 있는 분석 대상의 리스트와 라인을 미리 파악할 수 있다. 그리고 프로그램을 실행하기 전에 위치를 추적하여 취약점에 대한 대처 여부를 확인할 수 있게 한다.

3.1.2 인스트루먼트 삽입

이 단계에서는 핫 스팟 유형에 따라 동적 분석을 위한 인스트루먼트를 삽입한다. 인스트루먼트 삽입은 AOP를 위빙하는 시점에서 이루어진다. 이 때 AOP가 제공하는 기능에 따라 핵심 모듈과의 결합도에 영향을 미치지 않으면서 취약점을 분석하기 위한 애스팩트를 자동으로 삽입한다. 동적 분석 대상은 입력 초기 값에 대한 분석 또는 입력 후 검사 대상 메소드의 파라메타 등이 될 수 있다. 따라서 분석 대상에 따라 삽입되는 인스트루먼트의 종류도 달라진다. 삽입 된 인스트루먼트는 모니터링 모듈을 호출한다.

3.1.3 런타임 모니터링

실행하는 동안 어플리케이션이 핫 스팟에 도달하면 런타임 모니터가 인스트루먼트에 의해 호출된다. 런타임 모니터는 검사 대상 메소드의 프로토콜에 맞는 검사를 한다. 입력되는 파라메타를 검사하거나 리턴 값을 검사하기도 한다. 입력 초기에 입력 값을 검사 했더라도 프로그램 수행 중에 변경될 수 있기 때문에 다시 검사할 필요도 있다.

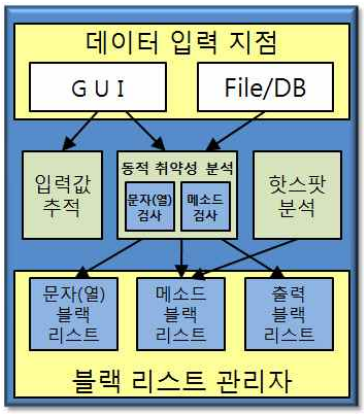

그림 3 취약점 분석 구조도

3.2 취약점 분석기

그림 2는 취약점 분석 구조도이다. 검증되지 않은 데이터들은 사용자에 의해 GUI를 통해 직접 입력되거나 시스템이 직접 File을 읽는다. 입력 지점에서 주입된 데이터를 입력 시점에서 문자(열) 블랙 리스트를 참조하여 검사한다. 문자(열) 블랙 리스트는 입력이 허용되지 않은 특수 문자 또는 문자열을 말한다. CERT의 10. IDS에서 정의한 취약한 메소드는 메소드 블랙 리스트에 보관되어 있다. 이러한 메소드가 사용되는 순간 애스팩트는 이를 포착하여 파라메터 또는 반환 정보를 보여준다. 출력 블랙 리스트는 사용자 인터페이스에 출력하면 안 되는 내용이나 파일이나 데이터베이스에 기록하면 안 되는 내용들을 저장하고 있다. 또한 핫스팍 분석기는 컴파일 타임에 블랙 리스트에 저장된 메소드를 정의한 코드의 위치를 미리 개발자에게 보여 줌으로써 취약성에 대비할 수 있는 정보를 제공한다.

4. 정적/동 적 분석 조합 애스팩트

본 절에서는 정적 분석과 동적 분석을 위한 애스팩트를 정의한다. 보안은 외부의 공격을 탐지하거나 회피하는 것이다. 보안을 확보하기 위해 외부로부터 입력되는 데이터를 확인하고 그 데이터를 처리하는 모듈에 악의적 영향을 미치는지 분석해야 한다. 예를 들어 이메일 서비스를 사용하기 위해서는 로그인 하기 위한 사용자 인증이 필요하다. 이때 공격이 가능한 부분은 사용자 인증을 위해 입력하는 ID가 될 수 있다. 사용자 인증에 있어 SQL 주입(injection)과 관련된 취약점은 이미 널리 알려져 있는 공격 기법이다[11]. 악의적 의도로 ID를 "bradseo' DELETE FROM user_T"로 입력한 경우 문자열을 검증하지 않고 그대로 코드 내부의 SQL문으로 받아와 실행하면 user_T 테이블을 삭제하는 결과를 초래할 수도 있다.

이러한 공격을 대비하기 위한 분석 방법은 크게 두 가지로 구분할 수 있다. 먼저 로그인 받는 코드가 외부 공격에 노출되어 있는지 컴파일 시점에서 정적 분석할 수 있다. 또한 동적으로 입력한 사용자 ID를 로그로 남기고 ID 값을 분석할 필요도 있다. 즉 정적 분석과 동적 분석 관점의 작업이 필요하다.

4.1 정적 분석을 위한 애스팩트

일반적으로 자바 프로그램의 스윙 패키지에 있는 JTextField를 이용하여 사용자에게 ID나 비밀번호와 같은 데이터를 입력 받는다. 이러한 경우 JTextField 타입의 객체가 getText()를 이용하여 어떤 값을 입력 받는다면 getText()의 결과 값은 반드시 검증을 받아야 한다. 즉 외부 접근이 있는 getText()의 사용이 취약할 수 있다는 것을 개발자에게 알려야 한다. 이러한 경고 내용을 컴파일 시점에 가능하도록 애스팩트로 정의하면 그림 3과 같다. 그림 3의 4 째 줄은 getText() 메소드가 호출되는 부분을 포착하는 교차점의 선언이다. 라인 6은 선언한 교차점에 대한 경고 메시지("attack")를 컴파일 시점에서 확인하는 메시지이다. 그 결과는 그림 3의 하단에 보는 것과 같이 경고 Warnings(1 item)와 attack 메시지 그리고 경고 받은 위치인 TextFieldTest.java 소스의 26 째 줄을 보여준다.

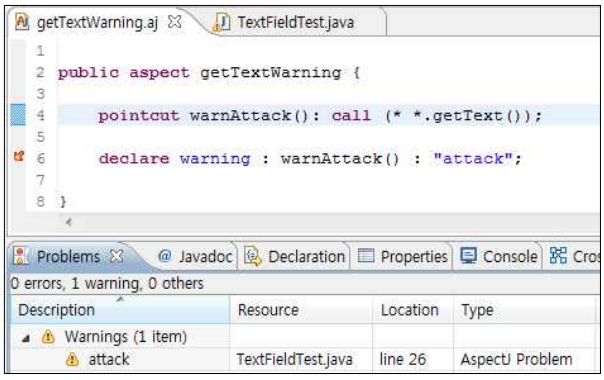

그림 4 정적 분석 애스팩트와 경고 메세지

4.2 동적 분석을 위한 애스팩트

동적 분석은 프로그램이 실행되는 시점에서 동작 과정을 살피는 것이다. 따라서 보안을 확보하기 위해 외부 데이터 주입과 관련된 코드 부분을 동적으로 관찰하고, 사용된 데이터를 로깅하여 동적 분석 할 수 있다. 그림 4는 로그인 과정에서 입력하는 데이터를 로깅하고 분석하는 애스팩트이다. 그림 4의 라인 5는 getText() 메소드에 대한 동적 분석을 위해 교차점으로 포착하고 있다. 일반적으로 자바 스윙(swing)의 JTextField 컴포넌트를 이용한 외부 입력은 getText() 메소드를 이용한다. 따라서 사용자가 입력한 문자열을 시스템에서 받아들이는 getText() 메소드에 대해 동적으로 검사한다. 그림 4의 라인 9와 10은 아이디 문자열의 길이가 8 보다 크거나 또는 아예 입력하지 않거나 작은따옴표(')가 있는 경우 외부의 공격 시도로 간주하고 이를 알려준다.

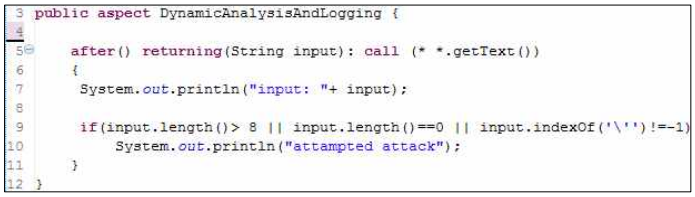

그림 5 동적 분석 애스팩트

분석하는 과정은 그림 3과 4의 애스팩트를 정의한 후 분석 대상 코드와 위빙 한다. 그리고 위빙 한 프로그램을 실행하여 사용자 입력에 대한 동적 분석의 실례를 그림 5부터 그림 7까지 보이고 있다. 그림 5는 정상적인 입력이다. 하지만 그림 6이나 그림 7과 같이 비정상적인 ID를 입력한 경우 로그 'attempted attack' 이라는 메시지를 출력한다. 따라서 사용자 입력에 따른 공격에 대한 동적 분석이 가능하다.

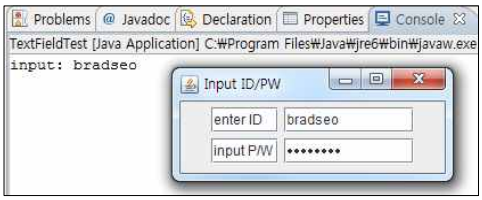

그림 6 정상적인 ID 입력

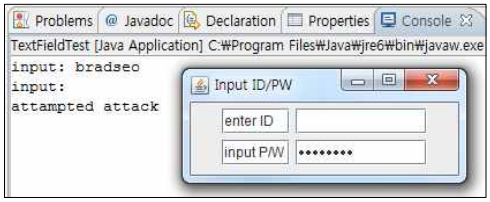

그림 7 ID 길이가 0인 경우

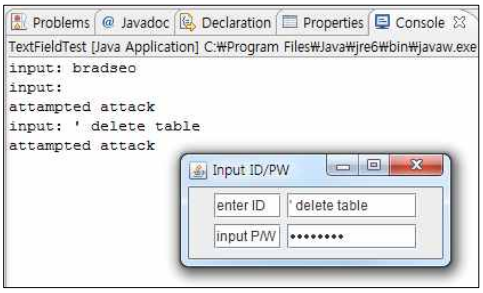

그림 8 ID에 ' 포함된 경우

5. 실험 평가

정적 분석을 위한 애스팩트를 이용하여 컴파일 시점에서 소스 코드에 대한 분석이 가능함을 3.1에서 보였다. 이러한 정적 분석 방법은 개발자가 구현한 프로그램 언어에 관점 지향 기법을 적용할 수 있는 애스팩트 언어만 이해하면 가능하다. 따라서 간단히 애스팩트를 정의하는 것만으로도 정적 분석이 가능하기 때문에 도구를 구입하거나 개발해야 하는 비용을 절약할 수 있다.

또한 동적 분석은 주로 분석이 필요한 위치에 인스트루먼트(instrument) 코드를 삽입한다. 이러한 방법은 분석해야하는 모든 위치에 인스트루먼트를 삽입하게 되어 분석 코드의 산재(scattering)가 발생한다. 즉 동적 분석에 대한 작업 코드가 여러 모듈에 퍼져 분산된다. 이러한 문제는 코드 재사용이나 유지 보수를 어렵게 한다. 하지만 애스팩트로 동적 분석에 대한 코드를 하나로 모듈화하고 개발된 코드와 독립적으로 관리할 수 있는 방법을 제공하고 있어 비모듈화에 대한 문제를 해결할 수 있다.

본 연구의 주요 기여는 정적 분석과 함께 동적 분석이 가능함에 따라 분리되어 있는 분석 작업 환경을 통합할 수 있다는 것이다. 이러한 통합 환경은 코드 자체의 취약점과 사용자의 외부 공격에 대한 분석을 모두 가능하게 한다. 이와 같은 정적 분석과 동적 분석을 통합하기 위해 애스팩트를 정의하고 실험하는 과정에서 확인한 사항을 표 2에 정리했다.

표 2 정적/동적 통합 분석

비교요소	통합 분석	정적 분석	동적 분석
분석대상	외부공격, 코드 취약점	코드 자체의 취약점	외부 공격
분석시점	컴파일 및 실행 시점	컴파일 시점	실행 시점
분석도구	직접 구현한 모듈화된 애스팩트	구입 또는 개발	분석코드 비모듈화

6. 결론

본 논문은 정적 분석과 동적 분석이 분리되어 실행되는 분석 환경을 하나로 통합하기 위한 방법을 제안했다. 정적 분석은 컴파일 시점에서 주로 소스 코드 자체의 취약점을 저비용으로 분석한다. 동적 분석은 산재된 인스트루먼트를 이용하여 프로그램 운용 시점에서 시스템 상태나 외부 공격을 분석하지만 정확한 분석 결과를 제공한다. 이러한 정적 분석과 동적 분석의 장점을 모두 취하면서 동시에 단점을 보완하는 방법을 관점지향 의 애스팩트로 보완했다. 그 결과 도구 구입이나 개발에 대한 많은 비용과 시간의 절약을 기대할 수 있다. 또한 실시간 외부 공격에 대한 분석 및 로그 코드를 애스팩트로 정의하여 코드의 모듈화를 실현했다. 이러한 통합적인 분석 방법은 정적 분석과 동적 분석에 대한 장점을 모두 획득하면서 동시에 단점을 보완할 수 있는 방법을 제공한다. 그 결과 인터넷과 같은 외부와의 상호작용이 많은 프로그램에 대한 보안성을 효과적으로 높일 수 있을 것으로 기대한다.

참고 문헌

[1] Gartner, "Now is the time for Security at Application Level," 2006, 12, 1
[2] D. Binkley, "Source Code Analysis: A Road Map," 2007 Future of Software Engineering(FOSE07), pp.104-119, 2007
[3] D. D. Cruz, P. Henriques, and J. S. Pinto, "Code Analysis: Past and Present," Proceedings of the Third International Workshop on Foundations and Techniques for Open Source Software Certification(OpenCert2009), 2009
[4] Kiczales, Gregor, G Lamping, A Mendhekar, C Maeda, C Lopes, J Loingtier, J Irwin, Aspect-Oriented Programming, Proc. of ECOOP, Vol. 1241, pages 220-242, 1997
[5] Fortify, www.fortify.com
[6] MITRE, www.mitre.org
[7] CERT, www.cert.org
[8] W.Huang, F Yu, C Hang, C Tsai, D.T Lee, S Kuo, Securing web application code by static analysis and runtime protection, Proc. of 13th ICWWW, pages 40-52, 2004
[9] D Wagner, D Dean, Intrusion detection via static, IEEE Symposium on Security and Privacy, pages 156-168, 2001
[10]Brian Chess, Secure Programming with Static Analysis, Addison-Wesley Professional, 2007
[11]Mei Junjin, An Approach for SQL Injection Vulnerability Detection, Proc. fo Informatin Technology conference 2009, p1411-1414, 2009