4

소프트웨어 보안취약성 자동진단도구 개발 사례

지티원 ▌권현준
한양대학교 ▌김현하 · 도경구*

1. 서 론

사용자 인터페이스가 모두 웹 중심으로 바뀌면서, 소프트웨어는 해킹으로 인한 보안사고에 더 많이 노출되어 있다. 고도의 암호화 기법을 사용하여 정보를 철저히 감추어도 소스코드가 부실하게 작성되어 있으면 이러한 보안 장치가 무용지물이 될 수 있다. 철저한 출입구 검사를 통해서 접근통제를 하는 방화벽도 소스코드의 취약점을 이용한 우회 공격에는 무력하다. 실제로 인터넷에서 발생하는 대부분의 해킹 사고는 소프트웨어의 소스코드에 존재하는 버그나 보안취약점을 악용하여 이루어진다. 따라서 이를 근본적으로 차단하기 위해서는 소스코드에 버그나 보안취약점이 없어야 하고, 이의 존재여부를 알아내는 것은 중요하다.

사람의 눈으로 코드리뷰를 하여 소스코드의 버그 또는 보안 취약점을 찾아낼 수 있지만, 개인의 숙련도에 의해서 품질이 좌우되고 경제적이지도 못하므로 자동으로 찾아주는 도구가 필요하다. 보안취약점을 들추어내주는 공격사례를 만들어 시험공격해보는 방법으로 해당 취약점의 존재여부를 알 수 있다. 그러나 이 방법을 사용하면 버그 또는 취약점 발생 지점을 정확하게 탐지하지 못하기 때문에 수정 노력 및 비용이 많이 든다. 프로그램을 실행해보지 않고 프로그램의 실행특성을 알아내는 방법을 정적 프로그램 분석(static program analysis)이라고 한다. 소스코드에 존재하는 버그 또는 보안취약점을 찾아내는 일도 정적 프로그램 분석으로 가능하다. 소스코드를 정적으로 분석하면 프로그램의 가능한 실행 경로를 모두 분석하기 때문에 놓치는 취약점이 없고, 정확한 지점을 지적해 줄 수 있어서 훨씬 더 경제적이고 이상적이다. 단점은 유한한 시간 안에 분석을 끝내기 위해서 요약하는 과정에서 정보를 잃어버릴 수 있어서 오탐의 가능성이 있다는 것이다. 오탐을 줄이기 위해서 분석의 정밀도를 높이면 분석시간이 길어지므로 적절한 경제공학적 균형 유지가 필요하다.

서울대학교 이광근 교수 연구팀이 개발한 Sparrow는 C 프로그램 소스에 담겨있는 버퍼넘침, 메모리누수, 이중해제와 같은 치명적인 메모리 오류를 자동으로 검출해주는 정적분석시스템으로, 파수닷컴(http://www.fasoo.com)에 의해서 상용화되어 판매되고 있다[1]. 한양대학교 도경구 교수와 이욱세 교수 연구팀이 지티원(http://www.gtone.co.kr)의 연구소와 공동으로 개발하여 상용화한 CodePrism은 보안취약성을 야기하는 소스코드의 구문패턴과 흐름패턴을 자체개발한 규칙명세언어로 기술하여 해당 패턴의 존재유무를 점검하는 정적 분석 도구이다[2]. 이러한 기술력을 바탕으로 2009년 행정안전부의 정보보호정책과가 주관한 전자정부지원사업의 일환으로 한국인터넷진흥원(KISA)이 수행한 "정보시스템 보안강화체계 구축" 사업에 지티원(주사업자), 파수닷컴, 한국정보보호학회의 소프트웨어보안연구회가 컨소시엄으로 참여하여 소스코드 취약점 자동진단도구를 개발하고 진단지원체계를 구축하였다. 이 자동진단도구는 최근 개발된 2건의 전자정부 소프트웨어를 진단하는데 성공적으로 사용되었으며, 앞으로 전자정부에서 발주되는 모든 소프트웨어에 확대 적용될 예정이다.

이 글에서는 이번에 개발된 소스코드 취약점 자동진단도구의 원리 및 구조에 대해서 간단히 소개한다.

2. 소스코드 취약점 진단 도구

소스코드 취약점 자동진단도구는 그림 1과 같이 클라이언트에서 수행되는 1차 분석과 분석엔진서버에서 수행되는 2차 분석으로 나누어진다. 1차 분석에서는 C와 Java 소스를 파싱한 후, 분석에 필요한 구문 및 의미 정보를 망라해서 가지고 있는 중간코드로 변환한다. 소스코드 유출의 우려를 불식시키기 위해 클라이언트 쪽에서 중간코드로 변환하고 암호화하여 분

* 종신회원

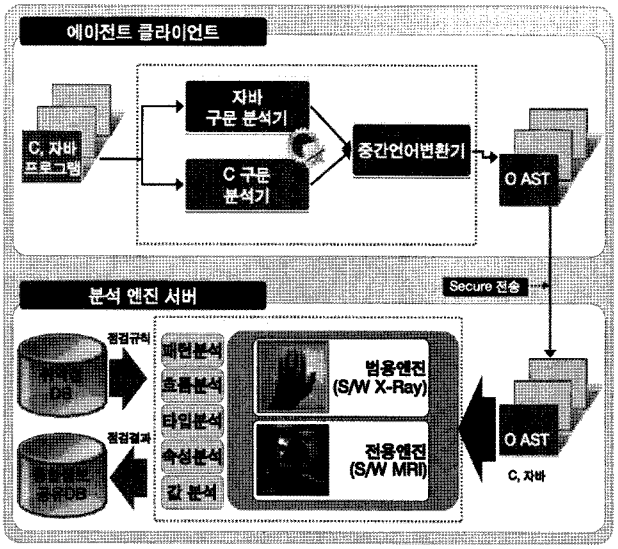

그림 1 소스코드 취약점 분석 엔진

석엔진서버로 전송한다. 중간코드는 유출이 된다 하더라도 형식 및 저장방식이 비공개이므로 중간코드에서 소스코드의 복원은 원천적으로 불가능하다. 2차 분석은 각 취약점의 특성 및 난이도에 따라서 두 가지 방법으로 나누어서 분석한다. 범용엔진은 단순한 구문 패턴이나 흐름 패턴으로 표현할 수 있는 다양한 취약점을 자체개발한 규칙명세언어로 기술하여 목록을 구축하고 이 목록에 수록된 패턴과 일치하는 취약한 부분을 일괄적으로 찾아낸다. 전용엔진은 단순한 패턴으로 표현할 수 없는 취약점에 대해서, 취약점 별로 고안된 분석기가 별도로 구현된 진단 엔진이다. 특정 취약점에 대해서 탐지의 정밀도를 높일 필요가 있는 경우에 전용엔진으로 진단하기도 한다.

개발된 소스코드 취약점 분석엔진은 현재 총 259개의 취약점을 진단할 수 있는데, 이 중에서 Java 취약점은 163개, C 취약점 96개이다. Java의 경우 119개의 취약점이 범용엔진으로, 44개 취약점이 전용엔진으로 진단되고, C의 경우 64개 취약점이 범용엔진으로, 32개 취약점이 전용엔진으로 진단된다.

2.1 범용 분석엔진

범용엔진은 일정한 구문 또는 흐름 패턴으로 취약점을 찾을 수 있는 분석엔진이다. 구문 또는 흐름 패턴은 자체개발한 규칙명세언어 RDL(Rule Description Language)로 기술할 수 있으며, 이 언어로 기술된 취약점 패턴을 가지고 범용엔진은 해당 취약점의 존재여부를 중간코드에서 진단한다. 각종 취약점 패턴은 취약점 DB로 유지되며 수시로 갱신할 수 있다. 취약점 패턴의 기본적인 구조는 취약조건과 제외조건의 2가지로 나누어진다. 취약조건은 UNSAFE 키워드로 표시되고, 제외조건은 EXCLUDE 키워드로 표시되며 취약조건을 만족하는 경우에도 제외조건을 만족하면 취약하지 않은 것으로 판정한다. RDL로 기술할 수 있는 패턴은 크게 구문패턴과 흐름패턴으로 나눌 수 있다.

구문패턴(syntactic pattern)은 특정 단어, 구문적 모양이나 위치를 표현할 수 있다. 예를 들어 버퍼넘침을 유발할 수 있는 위험한 함수 이름, 엔터프라이즈 서버 API에서 권고하지 않는 함수 이름, 요구조건을 갖추지 않고 선언된 전역변수이름이나 함수이름 등이 단어 패턴에 속한다. 추가로 특정 변수가 특정 타입이나 값을 가지고 있는지 여부, 특정 패턴이 특정 블록 안에 위치하고 있는지 여부 등도 표현이 가능하다. 구문패턴기술의 예를 몇 개 살펴보자.

strcpy함수는 C에서 버퍼넘침을 유발할 수 있는 위험한 함수이다. 규칙명세언어로는 아래와 같이 표현하며, 범용분석엔진은 strcpy 함수가 호출된 모든 지점들을 모아준다.

```
UNSAFE[strcpy(...)]
```

일반적으로 Java에서 두 문자열 값의 비교는 == 기호가 아닌 equals함수로 하도록 되어있다.

```
UNSAFE [($<java.lang.String>, ==,
                    $<java.lang.String>)]
```

위 패턴에서 $기호는 임의의 표현식을 의미하고 옆에 〈 〉기호로 해당 표현식이 java.lang.String 타입임을 의미한다. 범용분석엔진은 ==로 비교되는 java.lang.String 타입의 두 문자열 표현식을 찾아준다. public으로 한정된 finalize 메소드를 찾기 위해서 다음과 같이 표현하고,

```
UNSAFE [DEF(finialize)] qualified-by PUBLIC
```

Java 예외처리 문의 finally블록에 return이 있는 경우를 찾아내고 싶으면 다음과 같이 기술하면 된다.

```
UNAFE [return] in FINALLY
```

흐름패턴(flow pattern)은 프로그램의 흐름을 표현할 수 있다. 예를 들어, 연(open) 특정 파일을 프로그램 종료 전에 반드시 닫는지(close), 사용자 입력이 위험한 다른 함수의 인자로 전달되는지 등을 표현하기 위해서는 흐름을 기술할 수 있어야 한다. 흐름패턴은 구문패턴을 화살표로 연결하여 흐름관계를 표시한다. 제어흐름(control flow)은 프로그램의 수행 순서를 나타내고, 자료흐름(data flow)은 값의 전달되는 과정을 나타낸다. 자료흐름은 흐름이 직접적인지 간접적인지를 구분하여 직접적인 자료흐름(direct data-flow)과 간접적인 자료흐름(indirect data-flow)로 세분화하여 표현한다. 흐름의 방향에 따라 관계 화살표의 방향이 다르다. 화살표의 의미를 간단한 예로 살펴보면, 흐

름패턴 *A* -> *B*는 흐름 상 뒤에 *B*가 존재하는 *A*를 찾으라는 의미이고, *A* <- *B*는 흐름 상 앞에 *A*가 존재하는 *B*를 찾으라는 의미에 해당한다. 흐름패턴에서 나열된 구문패턴은 항상 프로그램 흐름 순서대로 좌측에서 우측으로 나열된다고 보면 된다. may-flow와 must-flow도 표현가능하다. may-flow는 *A* -> *B*와 같이 한 줄로 표시하며 *A*에서 *B*로의 흐름이 나타나는 경로가 하나라도 있다는 의미이고, must-flow는 *A* => *B*와 같이 두 줄로 표시하며 *A* 다음에 모든 경로에서 *B*가 존재해야 함을 의미한다.

흐름패턴의 종류별로 제어흐름은 cf, 직접자료흐름은 df, 간접자료흐름은 if로 화살표에 표시한다. 예를 들어 *A*와 *B*를 각각 특정지점을 나타내는 구문패턴이라 할 때, *A* <= cf *B*는 *B*에서 거꾸로 흐름을 따라 쫓아갔을 때 모든 경로에서 *A*가 존재한다는 뜻이다. 다른 종류의 흐름패턴을 예로 몇 개 살펴보자.

```
[createNativeQuery($1,...)] <-if
        [$1.setParameter(...)]
```

이 흐름패턴은 setParameter 패턴이 존재하며, 그 메소드를 호출하는 객체가 createNativeQuery 메소드의 첫 번째 인자이며, 두 지점 사이 자료흐름 경로가 최소한 하나 있으면 참이 되는 간접자료흐름패턴이다. 여기서 간접자료흐름이란 $1에 해당되는 변수가 하나 이상의 지점을 거쳐서 흘러갈 수 있다는 의미가 된다.

```
[getParameter(...)] ->df [setAttribute(..., $0)]
```

이 흐름패턴은 getParameter를 찾은 다음, 이 호출지점 이후 어느 한 경로라도 getParameter의 결과값($0는 좌측에 있는 단일패턴의 값을 명시한다)이 set-Attribute의 마지막 인자가 되면 참이 되는 직접자료흐름패턴이다. 이때 직접자료흐름이란 getParameter의 결과가 수정 없이(함수를 거치거나 다른 값을 더하거나 등) 바로 전달된다는 의미가 된다.

일반 패턴에서 일부 패턴을 제외하고 싶으면 EXCLUDE 구문을 사용하여 제외할 수 있다. 예를 들어 open 함수를 호출한 다음에 close 함수를 호출하지 않고 프로그램이 종료하는 경우를 찾고 싶다면 아래와 같이 표현할 수 있다.

```
UNSAFE [open(...)]
EXCLUDE [open(...)] =>cf [close(...)]
```

이 패턴에 대해서 범용분석엔진은 다음과 같이 작동한다. UNSAFE 패턴으로 소스코드에서 모든 open 함수의 호출지점을 모은 다음, EXCLUDE 패턴으로 open 함수 중에 모든 경로 상에 close함수가 호출되는 open 함수들을 모은다. UNSAFE 구문의 결과에서 EXCLUDE 구문의 결과를 제외시키면 어느 한 경로에서든지 close 함수를 호출하지 않은 open 함수의 호출지점만이 결과로 남게 된다.

범용엔진은 진단 대상이 되는 소스코드의 중간코드와 RDL로 기술된 검사규칙목록을 입력으로 받아서, 취약점을 탐지한다. 중간코드는 생성과정에서 모아둔 타입정보와 같은 프로그램의 기본적인 요소들을 가지고 있으며, 이러한 정보들은 분석에서 사용된다. 상수분석이나 널값여부 분석 등 간단한 수준의 값 분석은 성능을 위해서 미리 수행해 둔다. 단일패턴 분석기는 입력받은 규칙명세목록을 단일패턴단위로 쪼개서 각 단일패턴들의 위치와 부가정보들을 모은다. 흐름분석을 위해서 중간코드는 흐름그래프로 변환된다. 흐름패턴 분석기는 단일패턴들의 지점들과 흐름그래프로 흐름패턴으로 명시된 단일패턴들 사이의 관계를 분석하여 취약점 진단 결과를 내어준다.

2.2 전용 분석엔진

전용엔진은 패턴을 RDL로 기술할 수 없거나 정밀분석이 요구되는 경우 취약점별로 개별적으로 구축되는 분석엔진이다. 일반적으로 분석엔진의 정밀도를 높이기 위해서는 복잡한 분석을 감수해야 하고 따라서 분석시간이 많이 걸린다. 따라서 정밀도와 분석시간 사이에 적절한 조정이 필요하다. C의 경우 전용분석으로 처리된 32개의 취약점 중에서 18개 취약점은 값 분석을 자세히 수행하는 Sparrow 엔진을 기초로 제작되어서 정밀도가 이미 상당히 높다. C의 나머지 14개 취약점과 Java의 44개 취약점은 아직 정밀도 향상의 여지가 많이 남아있지만, 공학적이며 경제적인 측면에서 균형유지를 위한 적절한 의사결정이 필요하다.

2.3 진단 체계

이 진단도구를 사용하여 소스코드 취약점을 진단하는 체계는 그림 2와 같으며 전체적인 진단체계의 순서는 다음과 같다.

1. 소스코드 분석의뢰
   - 정부의 정보화담당자는 클라이언트 에이전트를 통해서 개발한 소스코드의 분석을 의뢰한다.
2. 1차 분석 및 가공 데이터 전송(SSL)
   - 정보화담당자의 클라이언트 에이전트는 분석의뢰 대상 소스코드를 중간코드로 변환한다.
   - 변환된 중간코드와 1차분석 도중에 가공된 데이터를 암호화하여 취약점분석엔진서버로 보낸다.
3. 2차분석
   - 취약점분석엔진에 1차분석의 결과가 전달되면 취약점DB로부터 분석에 요청된 최신의 점검규칙들

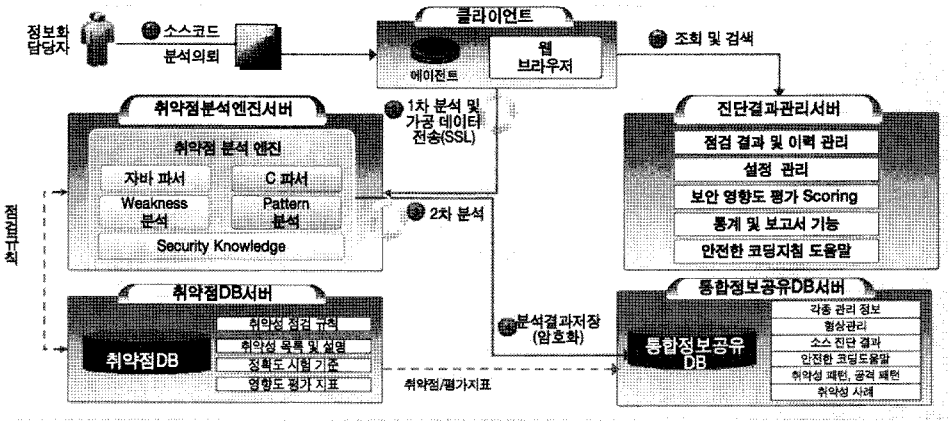

그림 2 소스코드 취약점 자동진단 체계

을 가져온다.

- 취약점 분석엔진에서 변환된 분석의뢰 대상 중간 코드와 점검규칙을 입력으로 받아서 취약점 분석을 수행한다.

4. 분석결과저장 (암호화)
   - 분석된 중간코드의 진단결과를 암호화하여 통합정보공유DB서버로 전송한다.
   - 통합정보공유DB서버는 전달된 진단결과를 저장 및 관리한다.
5. 조회 및 검색
   - 클라이언트는 진단결과관리서버를 통해 분석이 완료된 코드들의 진단 결과, 보안영향도 평가지표에 따른 평가성적표, 취약점이 없는 안전한 코딩 제안 등을 조회할 수 있다.

3. 결 론

2009년 행정안전부가 주관하고 한국인터넷진흥원이 수행한 "정보시스템 보안강화체계 구축" 사업은 정보화 프로세스에서 보안을 고려한 관리체계 구축을 통해 운영하기 전에 취약성 사전 제거로 보안 사고를 예방하고 이에 선도적으로 대응할 목적으로 시행한 사업이다. 이 사업의 결과로 개발된 소스코드 취약성 자동진단도구는 전자정부시스템에서 보안결함 발생을 최소화하는데 기여하게 될 것이며, 궁극적으로 국가 소프트웨어 보안수준 향상을 위한 안전한 소프트웨어 개발체계를 정립하는데 초석이 될 것이다.

앞으로 해야 할 과제로는 다양한 언어를 처리할 수 있도록 확장해야 하고, 추후로 드러나는 취약점에도 탄력적으로 대응하기 위한 체계를 구축해야 하며, 진단도구의 정밀도 향상을 위한 분석 기술 개발에 힘써야 할 것이다.

참고문헌

[1] Sparrow, http://www.spa-arrow.com

[2] Hyunha Kim, Tae-Hyoung Choi, Seung-Cheol Jung, Oukseh Lee, Kyung-Goo Doh, Soo-Yong Lee, "Rule-based Source-code Analysis for Detection of Security Vulnerability", WISA2009:The 10th International Workshop on Information Security Applications, Busan, South Korea, August 25~27, 2009

권 현 준

1986 서울대학교 자연과학대학 수학과 학사
1990~1998 펜타컴퓨터 Senior 소프트웨어 엔지니어
1998~2008 아이티플러스 CTO
2008~현재 지티원 정보통신연구소 개발1본부장
관심분야 : 소프트웨어 공학, 소프트웨어 구조분석, 애플리케이션 보안
E-mail : hjkweon@gtone.co.kr

김 현 하

2003 한양대학교 전자컴퓨터공학부 (학사)
2005 한양대학교 컴퓨터공학과 (석사)
2006~현재 한양대학교 컴퓨터공학과(박사과정 수료)
관심분야 : 프로그램분석, 소프트웨어보안
E-mail : hhkim@pllab.hanyang.ac.kr

도 경 구

1980 한양대학교 산업공학과(학사)
1987 Iowa State University, Computer Science(석사)
1992 Kansas State University, Computer Science (박사)
1993~1995 University of Aizu 교수
2005~2006 University of California Davis 방문교수
1995~현재 한양대학교 ERICA캠퍼스 교수
관심분야 : 프로그래밍언어, 프로그램분석, 소프트웨어보안, 소프트웨어공학
E-mail : doh@hanyang.ac.kr