한국통신학회논문지(J-KICS) '13-10 Vol.38C No.10
http://dx.doi.org/10.7840/kics.2013.38C.10.831

소프트웨어 개발 보안성 강화를 위한 주요 보안약점 진단규칙 연구

방 지 호◆, 하 란*

Research on Major Weakness Rules for Secure Software Development

Jiho Bang◆, Rhan Ha*

요 약

최근 소프트웨어 보안성 강화를 위해 소프트웨어 개발단계에서 보안취약점의 원인인 보안약점을 제거하기 위한 정적분석 기반의 도구를 많이 활용하고 있다. 따라서, 보안약점 진단도구는 다양한 보안약점을 진단할 수 있는 진단규칙을 보유하는 것이 필요하다. 2011년부터 2012년까지 국내 주요 정보화사업으로 개발된 소프트웨어에서 발견된 상위 5개의 보안약점은 연도별 상위 10개 보안약점의 76%에 해당된다. 소프트웨어 개발시 상위 5개의 보안약점만 적절히 조치하여도 소프트웨어 보안성이 많이 개선될 수 있다. 본 논문은 많이 활용되고 있는 공개용 진단도구인 PMD를 대상으로 주요 보안약점에 대한 진단규칙과 이에 대한 성능시험 결과를 제시한다.

Key Words : Weakness, Weakness Rule, Security, Static Analysis Tool, SW Development

ABSTRACT

Recently, to enhance the security of software, static analysis tools for removing weaknesses, the cause of vulnerability, have been used a lot in the software development stage. Therefore, the tools need to have the rules being able to diagnose various weaknesses. Top 5 weaknesses found in the software developed by major domestic information projects from 2011 to 2012 is 76% of top 10 weaknesses per year. Software security can be improved a lot if top 5 weaknesses just are removed properly in software development. In this paper, we propose the PMD's rules for diagnosing the major weaknesses and present the results of its performance test.

Ⅰ. 서 론

최근 소프트웨어(이하, SW) 보안성 강화를 위해 SW 개발단계에서 보안취약점(Vulnerability)의 원인인 보안약점(Weakness)을 제거하기 위한 정적분석 기반의 도구를 많이 활용하고 있다[1]. 이는 SW 개발단계에서 보안약점을 제거하는 것이 SW 개발완료 및 배포 이후에 발생할 수 있는 보안취약점의 패치 비용을 효과적으로 감소[2]시킬 수 있는 방법이기 때문이다.

국내의 경우, 정보화 사업에 따라 SW 개발시 보안약점이 존재하지 않도록 개발하는 것과 해당 SW의 보안약점 존재 유무 확인을 위해 감리법인이 진단도구 사용시 신뢰된 도구를 사용하는 것이 의무화[3] 되었다. 따라서, 신뢰성이 보장된 정적분석 기반의 도구

※ This work was supported by the National Research Foundation of Korea grant funded by the Korean government, Ministry of Education, Science and Technology (No.2013-023635).
◆ First Author : 홍익대학교 컴퓨터공학과 실시간시스템 연구실, jhbang@kisa.or.kr, 정회원
* 홍익대학교 컴퓨터공학과 실시간시스템 연구실, rhanha@cs.hongik.ac.kr
논문번호 : KICS2013-08-349, 접수일자 : 2013년 8월 26일, 최종논문접수일자 : 2013년 10월 14일

를 사용하는 것이 필요하며, 다양한 보안약점을 진단할 수 있는 진단규칙을 보유하는 것이 필요하다.

SW 보안약점 진단도구의 신뢰성을 보장하기 위해서는 진단도구가 갖추어야 하는 기능 요구사항과 이에 대한 평가 기준 및 방법 관련 연구가 필요하다. 또한, 진단도구가 보유해야 하는 최소 보안약점에 대한 진단규칙을 연구 및 개발하여 진단도구의 성능을 제고하는 것이 필요하다.

기존 연구들은 공개용 소프트웨어를 기반으로 상용 및 공개용 진단도구가 진단하는 보안약점을 비교분석[1]하거나, 버퍼오버플로우, SQL 삽입 등 특정 보안약점에 대한 진단규칙 제시 및 관련 성능 분석[4] 중심으로 진행되었다. 그리고, 보안약점을 분석하여 보안약점을 회피하는 시큐어코딩[2,5] 및 개발자 또는 보안약점 진단원 등이 보안약점을 확인하는 방법[6]에 대한 가이드를 개발하여 배포하고 있다.

다양한 보안약점에 대한 진단도구의 진단규칙 연구 및 개발은 진단도구의 보안약점 진단범위 및 진단 성능 개선에 있어 매우 중요한 부분이지만, 관련 연구는 매우 미비한 실정이다. 또한, 상용 진단도구의 경우 공개용 도구에 비해 사용자 인터페이스 및 성능이 상대적으로 우수하지만 구매 비용에 대한 부담이 있기 때문에 중소기업 및 영세기업의 경우 상용 진단도구를 선택하는 것이 쉽지 않다.

최근 행정안전부 및 한국인터넷진흥원(이하, KISA)는 국내외 상용 진단도구에 대한 시범검증을 수행하였으며, 이를 기반으로 IT보안인증사무국은 보안약점 진단도구에 대한 보안규격을 마련하였다. 이에 따라, 정보보호제품 보안성 평가·인증(CC 평가·인증)이 금년(2013년) 하반기에 진행되어 2014년 1월 전·후로 CC 인증된 제품이 출시되어 정보화사업의 정보시스템 감리시 활용되게 된다. 그러나, SW 개발사업자에 대한 CC 인증된 진단도구 사용 의무규정이 없기 때문에 기보유한 상용 진단도구를 사용하거나 공개용 진단도구를 사용할 수 있다.

국가 정보화사업시 정부에서 개발하여 보급하고 있는 전자정부 표준 프레임워크[7]를 많이 사용하고 있다. 해당 프레임워크는 JAVA 개발환경을 제공하고 있으며, JAVA 소스코드에 대한 진단도구로 PMD[8] 및 FindBugs[9]를 개발환경에 포함하여 배포하고 있다. 특히, PMD의 경우 PMD의 진단규칙 중 코딩 품질 관점의 별도의 진단규칙을 선별하여 전자정부용으로 배포하고 있다. 보안약점에 대한 진단규칙이 부족한 PMD를 대상으로 주요 보안약점에 대한 진단규칙을 개발하여 배포하면 다른 공개용 도구에 비해 파급효과가 높을 수 있다.

따라서, 본 논문은 공개용 진단도구 중 전자정부 프레임워크에서 사용을 권고하고 있는 PMD를 대상으로 주요 보안약점에 대한 진단규칙을 제시한다. 또한, 주요 보안약점을 대상으로 개발된 PMD의 진단규칙의 성능평가를 통해 PMD의 성능개선 효과를 설명한다. 전자정부프레임워크의 개발환경에 포함된 PMD는 공개용 소프트웨어로 국가 정보화사업 및 민간 SW 개발시 비용 없이 쉽게 활용할 수 있기 때문에 논문에서 제시한 주요 보안약점 진단규칙을 PMD에 적용하여 활용할 경우, SW 보안성을 효과적으로 개선할 수 있을 것으로 기대한다.

본 논문은 2장에서는 진단도구의 진단규칙 관련 연구 동향을 설명하고, 3장에서는 주요 보안약점 기반의 진단규칙을 제시한다. 그리고, 4장에서는 본 논문에서 제안하는 주요 SW 보안약점 진단규칙 적용 결과에 대해 설명하고, 5장에서 결론을 맺는다.

Ⅱ. 관련 동향

본 장에서는 SW 개발 단계에서 보안성을 보증하가 위한 기존 연구 동향과 SW 보안약점 진단도구에 대해 살펴보도록 한다.

2.1. SW 보안성 보증 관련 연구 동향

SW 보안약점 진단도구에 대한 SW 보안성 보증을 위한 SW 보안약점 진단도구 성능 관련 연구는 특정 보안약점(예, C언어의 경우 버퍼오버플로우, JAVA언어의 경우 SQL 삽입 등)을 대상으로 자체 구현한 예제코드를 기반으로 진단도구가 보유한 진단규칙을 분석하거나 진단도구 진단규칙을 구현하여 진단도구의 진단능력에 대한 성능을 분석[4]하였다. 그리고, 공개용 SW(예, wireshark 등)를 대상으로 상용 및 공개용 진단도구가 진단할 수 있는 보안약점을 분석[1]하여 진단도구별 특성을 식별하고 진단도구가 보유하고 있는 진단규칙이나 특정 보안약점을 중심으로 연구를 진행하였다.

그 외에도, 진단도구의 기능 요구사항을 도출하여 평가방법론을 제시한 연구[10]도 있으며, 보안약점 관련 가이드[2,5,6]를 개발하는 사례도 있었다.

가이드에는 안전한 SW 개발을 위한 '시큐어코딩 가이드[2,5]'와 보안약점 진단을 위한 'SW 보안약점 진단가이드[6]'가 있다. 시큐어코딩 가이드의 경우, 카네기멜론대학(CMU)의 SW공학연구소(SEI)에서 개발 및 배포하는 C/C++ 및 JAVA 언어 등에 대한 시큐

어코딩 가이드[5]가 대표적이다. 국내의 경우, 행정안전부와 KISA가 43개 보안약점을 중심으로 SW 개발자 대상의 'SW 개발보안 가이드[2]'를, SW 보안약점 진단원 등을 대상으로 'SW 보안약점 진단가이드[6]'를 개발하여 배포하고 있다.

시큐어코딩 가이드는 SW의 안전한 구현방법에 대한 내용으로 구성되는데, 국외 가이드의 경우 코딩규칙을 중심으로 기술되어 있으며 국내 가이드의 경우 보안약점을 중심으로 기술되어 있는 차이점이 있다. 국외 가이드의 코딩규칙을 기반으로 이에 대한 적용 여부를 분석할 수 있는 일부 진단규칙에 대한 연구도[11] 있었다. 진단가이드는 진단원 등이 소스코드를 직접 분석하여 SW 보안약점의 존재 여부를 확인하거나, 진단도구의 진단결과가 정탐(True Positive, TP) 또는 오탐(False Positive, FP)인지 여부를 확인하는 방법에 대해 기술하고 있다. 따라서, 진단가이드의 내용과 진단도구의 진단 알고리즘과는 차이가 있다.

2.2. 보안약점 진단도구

NIST(National Institute of Standards and Technology) SAMATE(Software Assurance Metrics and Tool Evaluation) 프로젝트[12]는 SW의 품질 및 보안성 개선을 위해 SW 개발단계별 사용할 수 있는 다양한 도구 목록을 제공하고 있으며, 소스코드를 대상으로 보안약점을 진단할 수 있는 정적분석 기반의 진단도구 진단 성능을 시험할 수 있는 예제코드도 제공하고 있다. SAMATE 프로젝트에 따르면, 소스코드를 대상으로 보안약점을 진단할 수 있는 상용 및 공개용 진단도구 중 JAVA 언어로 구현된 소스코드의 보안약점을 분석할 수 있는 공개용 진단도구는 FindBugs(FB), FindSecurityBugs(FSB), JLint(JLi), LAPSE+(LAP), PMD, Yasca(Yas) 등 6종이 있다. 또한, JAVA 기반의 보안약점 진단도구의 진단 성능을 측정할 수 있는 예제코드로 Juliet 코드를 제공하고 있는데 국내에서 의무화한 43개 보안약점 중 25개 보안약점에 대한 진단 성능을 측정할 수 있는 예제코드가 포함되어 있다. 기존 연구결과[13]에 따르면 43개 보안약점 중 Juliet 코드를 기반으로 분석 가능한 25개 보안약점에 대해 6종의 진단도구는 다음 그림 1과 같이 보안약점 유형별 일부 진단규칙만 가지고 있다.

2.3. PMD

PMD[8]는 JAVA 프로그래밍 언어로 구현된 SW를 실행하지 않고 해당 소스코드를 분석하여 소스코드에 존재하는 사용되지 않는 변수, try-catch 구문에서 비어있는 catch 블록 등 SW 결함을 탐지해주는 정적분석 기반의 공개용 진단도구이다. PMD는 JavaCC 컴파일러를 통해 소스코드를 분석하여 AST(Abstract Syntax Tree)를 구성한다. PMD는 AST 및 심볼테이블 등을 기반으로 패턴매칭, 구조 및 의미(Semantic) 분석, 제한적인 제어 및 데이터 흐름분석을 수행할 수 있다. PMD는 Java 및 XPath를 기반으로 진단규칙을 구현할 수 있으며, 270여개 정도의 진단규칙을 보유하고 있다.

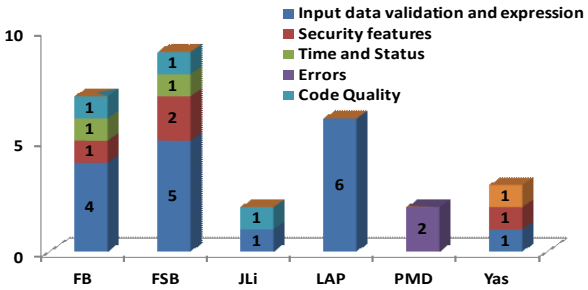

그림 1. 보안약점 유형별 진단규칙 수
Fig. 1. Rules number of weakness for each type

국내에서 전자정부 SW 개발시 필수적으로 제거해야 하는 보안약점 중 PMD가 보유하고 있는 관련 진단규칙은 EmptyCatchBlock(오류상황 대응 부재), AvoidPrintStackTrace(오류메시지를 통한 정보노출), ArrayIsStoredDirectly(Private 배열에 Public 데이터 할당) 등 3개 정도만 해당되며, AvoidPrintStackTrace 진단규칙의 경우, 'printStackTrace()' 메소드의 존재유무를 점검하기 때문에 '오류메시지를 통한 정보노출' 보안약점을 제한적으로 진단할 수 있다.

현재, 전자정부프레임워크에서 SW 구현시 소스코드 분석을 위해 Java 클래스 파일을 대상으로 보안약점을 분석하는 FindBugs를 포함하여 PMD 사용을 권고하고 있다. PMD의 경우, PMD가 보유하고 있는 진단규칙을 선별하여 전자정부용 진단규칙으로 별도 패키징하여 배포하고 있는데 위에서 언급한 3개의 진단규칙 중 EmptyCatchBlock(오류상황 대응 부재) 진단규칙만 포함되어 있다.

Ⅲ. 주요 보안약점 분석 및 규칙 연구

본 장에서는 최근 국내 전자정부 SW 보안약점 진단시 발견된 주요 보안약점을 분석하여 진단도구가 진단할 수 있는 방법과 공개용 진단도구인 PMD의 진단규칙 구현 알고리즘에 대해 설명한다.

3.1. 최근 발견된 주요 보안약점

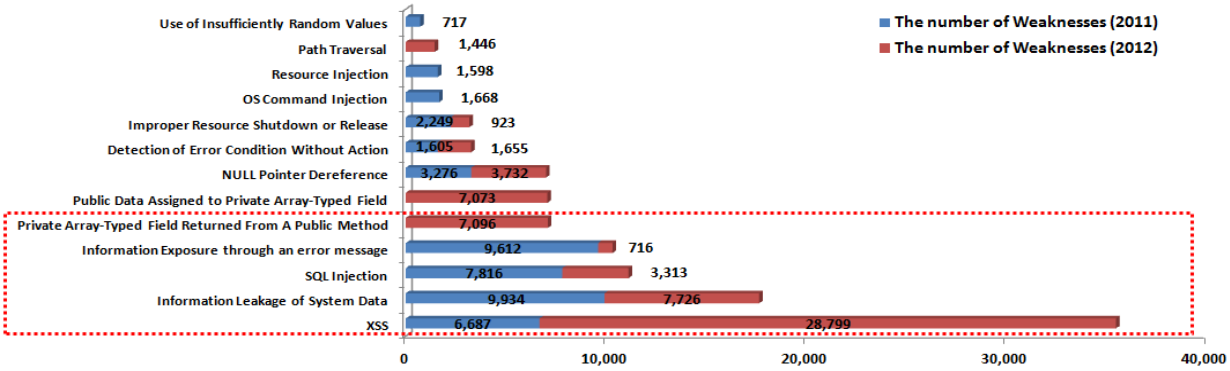

그림 2. 최근 2년간 발생한 주요 보안약점(2011~2012)
Fig. 2. Major Weaknesses occurred during last 2 years(2011~2012)

최근 2년간(2011년~2012년), 주요 정보화사업으로 개발된 SW에서 발견된 상위 10개의 보안약점[14]은 그림 2와 같다. 그 중 '크로스사이트스크립트(이하, XSS)', '시스템 데이터 정보노출', 'SQL 삽입', '오류 메시지를 통한 정보노출', 'Public 메소드부터 반환된 Private 배열'은 상위 5개 보안약점으로 전체의 76%에 해당된다. 상위 5개의 보안약점을 중심으로 SW 개발시 해당 보안약점만 조치를 취하여도 SW 보안성이 많이 개선될 수 있다.

국가 정보화사업으로 개발되는 SW의 경우, SW 개발시 43개 보안약점을 배제하여 개발하는 것이 필요한데 개발자의 경우 CC 인증된 도구 사용에 대한 의무가 없기 때문에 상용 또는 공개용 도구를 자유롭게 사용할 수 있다. 전자정부프레임워크 개발환경에서 포함되어 배포되는 공개용 진단도구 PMD의 경우, 별도의 행정기관용 진단규칙을 제공하고 있는데 해당 진단규칙에 본 논문에서 제시한 주요 보안약점을 진단할 수 있는 진단규칙이 포함된다면 보다 안전한 SW 개발에 기여할 수 있다.

따라서, 본 논문은 JAVA 프로그래밍 언어로 구현된 SW의 소스코드를 진단할 수 있는 정적분석 기반의 보안약점 진단도구 PMD를 대상으로 상위 5개의 주요 보안약점에 대한 진단규칙을 다음 절에서 제시한다.

3.2. 주요 보안약점 분석 및 진단규칙 개발

상위 5개의 보안약점을 관련 예제코드를 기반으로 분석하여 해당 보안약점을 진단할 수 있는 진단규칙에 대해 살펴본다. PMD 진단규칙은 JAVA 또는 XPath로 구현할 수 있는데, 데이터 흐름 등 복잡한 로직을 구현할 경우 JAVA로 구현해야 하는데 구현 코드 규모가 있기 때문에 JAVA로 구현한 진단규칙의 경우, 알고리즘을 중심으로 설명한다. XPath의 경우, 구조 및 패턴매칭 등 간단한 로직을 구현할 경우 사용되며 JAVA에 비해 규모가 작기 때문에 실제 구현한 진단규칙을 제시하여 설명하도록 한다.

3.2.1. XSS

'XSS' 보안약점은 동적으로 생성되는 웹페이지(예, 게시판)가 적절한 입력값 검증 없이 외부 데이터 입력을 허용하는 경우, 해당 웹페이지를 요청하는 사용자 브라우저를 통해 해당 웹페이지에 삽입된 악성코드가 실행되어 사용자의 중요정보 유출 등이 발생할 수 있는 보안약점이다.

Juliet 시험코드[12]는 'XSS' 보안약점을 다양한 유형으로 구현하여 제시하고 있는데, 대표적인 유형은 다음 표 1의 예제코드와 같으며, 진단도구가 해당 보안약점을 진단할 수 있는 진단방법(표2)은 다음과 같다.

표 1. 예제코드(XSS)
Table 1. The sample code(XSS)

```
public class XSS {
 public void flaw(HttpServletRequest request,
    HttpServletResponse response) throws Throwable  {
    // do something
    data = request.getParameter("name");
    // do something
    /* Weakness exists */
    response.getWriter().println("<br>data = " + data);
 }
}
```

표 2. 진단도구의 진단로직(XSS)
Table 2. Diagnosis Logic for Tool(XSS)

① Check the existence of strings related to web pages such as HTML tags, scripts etc.
② If exist, check the existence of variables in the string
③ If variable(s) is included, check the variable using data flow whether the variable is assigned the value of the external input without validation
ex) Methods related external input
HttpServeletRequest.getProperty()
HttpServeletRequest.getQuertyString()
HttpServeletRequest.getCookies() etc.
④ If the value of the external input is used, check whether the sting is output using the following servlet method
ex) HttpServeletResponse.getWriter() etc.
⑤ If the sting is used as a parameter(s) of methods such as getWriter().println() etc., the tool prints the message that the weakness exists

JAVA 소스코드에 HTML 태그('<', '>' 등) 등 스크립트가 포함된 문자열을 포함하고 있는 경우, 'XSS' 보안약점이 존재할 가능성이 있기 때문에 정규식 등을 이용한 매칭 함수를 사용하여 해당 문자열의 존재 유무를 확인한다. 그리고, HTML 문자열과 변수가 결합하는(예, "
 data =" + data) 문자열인지 확인하여 해당 변수(예, data)가 사용자 웹브라우저 등을 통해 입력될 수 있는 외부 값인지를 추적한다. 외부 값을 검증 없이 사용하여 사용자에게 전송하는 경우 웹페이지를 생성하여 사용자에게 전송하는지 여부를 점검하여 해당 보안약점의 존재여부를 확인한다.

따라서, PMD의 진단규칙으로 구현하기 위해서는 AST(Abstract Syntax Tree) 구조로 변환된 JAVA 소스코드를 visit() 메소드, 심볼테이블, 데이터 흐름 등을 이용하여 JAVA로 구현하는 것이 필요하며, 관련 알고리즘은 다음 표 3과 같다.

표 3. PMD 진단규칙 알고리즘(XSS)
Table 3. The algorithm of PMD's Rule(XSS)

```
class XSSRule extended AbstractJavaRule {
 public Objective visit(ASTMethodDeclaration) {
  if (call through other visit() method)
     check whether the variable in the sting is assigned
     the value of the external input without validation;
 }
 public Objective visit(ASTMethodDeclarator) {
  // for checking the value of the external input
  Store the method parameter(s);
 }
 public Objective visit(ASTAdditiveExpreesion) {
    Identify the string, Literal, included HTML tages etc.;
    Check the existence of the variable, Name;
   Store the variable(s) and call visit(ASTMethodDeclaration);
   if (the string is assigned the value of the external input) {
      Check whether the sting is output using servlet method;
      if (the sting is output)
       // for XSS weakness diagnosis
       Print diagnostic message;
} } }
```

3.2.2. 정보노출

정보노출 관련 보안약점은 '시스템 데이터 정보노출' 및 '오류메시지를 통한 정보 노출' 보안약점이 있다. '시스템 데이터 정보노출' 보안약점은 경로 정보 등 시스템 데이터 정보가 외부에 공개되어 해킹 등의 사이버공격에 도움이 될 수 있는 보안약점 이다. '오류메시지를 통한 정보노출' 보안약점은 SW 오류 발생으로 인해 생성되는 오류메시지에 실행환경, 사용자 등에 대한 중요 정보가 포함되어 외부에 공개되는 경우 해킹 등의 사이버공격에 도움이 될 수 있는 보안약점이다.

Juliet 시험코드는 '시스템 데이터 정보노출' 및 '오류메시지를 통한 정보 노출' 보안약점을 다양한 유형으로 구현하여 제시하고 있는데, 대표적인 유형은 다음 표 4의 예제코드와 같다.

표 4. 관련 예제코드
Table 4. The related sample codes

```
<Information Leakage of System Data>
public class ExposureOfSystemData {
 public void flaw(HttpServletRequest request,
   HttpServletResponse response) throws Throwable {
    // do something
    /* Weakness exists */
    response.getWriter().println("Not in path: "
            + System.getenv("PATH"));
    // do something
 }
    // do something
}
```

```
<Information exposure through an error message>
public class InformationExposure {
 public void flaw(HttpServletRequest request,
   HttpServletResponse response) throws Throwable {
   // do something
   try {
     // do something
   } catch (Exception e) {
     e.printStackTrace();    /* Weakness exists */
} } }
```

먼저, '시스템 데이터 정보노출' 보안약점의 경우 운영체제의 환경변수 정보를 다루는 메소드를 사용할 때 보안약점이 존재한다고 할 수 있기 때문에 진단도구로 진단하는 방법(표5)은 관련 메소드의 존재를 식별하여 해당 보안약점의 존재여부를 확인하는 것이다.

표 5. 진단도구의 진단로직(시스템 데이터 정보노출)
Table 5. Diagnosis Logic for Tool(Info. Leakage of Sys. Data)

① Check the existence of the following method related to the system information such as OS environment variable(s)(ex, PATH etc.) etc. ex) System.getenv() etc. ② If exist, the tool prints the message that the weakness exists

따라서, PMD의 진단규칙으로 구현하기 위해 AST 구조로 변환된 JAVA 소스코드 내에서 'System.getenv()' 메소드를 식별하도록 XPath로 구현하는 것이 필요하며, 관련 진단규칙은 다음 표 6과 같다.

표 6. PMD 진단규칙(시스템 데이터 정보노출)
Table 6. PMD's Rule(Info. Leakage of System Data)

```
//PrimaryExpression
[PrimaryPrefix/Name[contains(@Image,'getenv') or
starts-with(@Image, 'System.getenv')]]
```

'오류메시지를 통한 정보노출' 보안약점의 경우, 사전에 정해진 에러 메시지가 아닌 'printStackTrace()' 등과 같은 메소드를 통해 에러메시지를 출력할 경우 중요정보가 노출될 수 있는 보안약점이 존재한다고 할 수 있기 때문에 진단도구로 진단하는 방법(표7)은 관련 메소드의 존재를 식별하여 해당 보안약점의 존재여부를 확인하는 것이다.

표 7. 진단도구의 진단로직(오류메시지를 통한 정보노출)
Table 7. Diagnosis Logic for Tool(Info. exposure through an error message)

① Check the existence of the following methods ex) printStackTrace(), getMessage(), getLocalizedMessage(), toString() etc. ② If exist, the tool prints the message that the weakness exists

따라서, PMD의 진단규칙으로 구현하기 위해 AST 구조로 변환된 JAVA 소스코드 내에서 'printStackTrace()' 등과 같은 에러 메시지를 출력하는 다양한 메소드를 식별하도록 XPath로 구현하는 것이 필요하다. PMD의 기존 'AvoidPrintStackTrace' 규칙은 단순히 'printStackTrace()' 메소드만 식별하도록 구현되어있다. 그러나, 해당 보안약점에는 다양한 유형의 메소드가 존재하며 해당 메소드가 try-catch 구문에서 사용될 때만 보안약점의 의미와 일치하기 때문에 try-catch 구문에서 다양한 메소드를 고려하였으며, 특정 메시지가 출력되도록 메소드의 매개변수에 특정 문자열이 있는 경우 제외하도록 진단규칙을 개선하였다. 관련 진단규칙은 다음 표 8과 같다.

표 8. PMD 진단규칙(오류메시지를 통한 정보노출)
Table 8. PMD's Rule(Info. exposure through an error message)

```
//TryStatement/CatchStatement
[FormalParameter/Type/ReferenceType/ClassOrInterfaceT
ype[contains(@Image, 'Exception')]]
/Block/BlockStatement/Statement/StatementExpression/Pri
maryExpression
[PrimaryPrefix/Name[
contains(@Image, 'printStackTrace') or
contains(@Image,'getMessage') or
contains(@Image,'getLocalizedMessage') or
contains(@Image,'toString')]]
[PrimarySuffix[not(boolean(Arguments/ArgumentList/Expr
ession/PrimaryExpression/PrimaryPrefix/Literal))]]
```

3.2.3. SQL 삽입

'SQL 삽입' 보안약점은 외부 입력값을 적절한 검증 없이 데이터베이스(DB)에 대한 SQL로 사용하거나 SQL에 포함시키는 경우 DB의 중요정보가 노출되거나 변경될 수 있는 보안약점이다.

Juliet 시험코드는 'SQL 삽입' 보안약점을 다양한 유형으로 구현하여 제시하고 있는데, 대표적인 유형은 다음 표 9의 예제코드와 같으며, 진단도구의 진단방법(표10)은 다음과 같다.

표 9. 예제코드(SQL 삽입)
Table 9. The sample code(SQL Injection)

```
public class SQLInjection {
 public void flaw(HttpServletRequest request,
    HttpServletResponse response) throws Throwable {
     // do something
     String data = request.getParameter("ID");
     // do something
     Boolean result = sqlstatement.execute("SELECT
       * FROM USERS WHERE ID = '" + data);
 }
     // do something
}
```

표 10. 진단도구의 진단로직(SQL 삽입)
Table 10. Diagnosis Logic for Tool(SQL Injection)

① Check the existence of the following SQL ex) "insert into users (status) values ('updated') where name = ", "select * from users where id = " etc. ② If SQL exists, check the existence of variables in SQL ex) "select * from users where id ='"+data+"'" "insert into users (status) values ('updated') where name='"+data+"'" ③ If variable(s) is included, check the variable using data flow whether the variable is assigned the value of the external input without validation ex) HttpServeletRequest.getProperty() HttpServeletRequest.getQuertyString() HttpServeletRequest.getCookies() etc. ④ If the value of the external input is used, check whether the SQL is executed using the following method ex) Statement.execute(), Statement.executeBatch() Statement.executeQuery(), Statement.executeUpdate() etc. ⑤ If the SQL is used as a parameter(s) of methods such as execute() etc., the tool prints the message that the weakness exists

JAVA 소스코드에 SQL로 구성된 문자열을 포함하고 있는 경우, 'SQL 삽입' 보안약점이 존재할 가능성이 있기 때문에 정규식 등을 이용한 매칭 함수를 사용하여 해당 문자열의 존재유무를 확인한다. 그리고, SQL 문자열과 변수가 결합하는(예, "SQL문 =" + data) 문자열인지 확인하여 해당 변수(예, data)가 웹, 콘솔, 네트워크 등의 환경을 통해 입력될 수 있는 외부 값인지를 추적한다. 외부 값을 검증 없이 SQL로 결합하여 DBMS에 전송하는지 점검하여 해당 보안약점의 존재여부를 확인한다.

따라서, PMD의 진단규칙으로 구현하기 위해서는 AST 구조로 변환된 JAVA 소스코드를 visit() 메소드, 심볼테이블, 데이터 흐름 등을 이용하여 JAVA로 구현하는 것이 필요하며, 관련 알고리즘은 다음 표 11과 같다.

표 11. PMD 진단규칙 알고리즘(SQL 삽입)
Table 11. The algorithm of PMD's Rule(SQL Injection)

```
class SQLInjectionRule extended AbstractJavaRule {
 public Objective visit(ASTMethodDeclaration) {
  if (call through other visit() method)
     check whether the variable in the SQL sting is assigned
     the value of the external input without validation;
 }
 public Objective visit(ASTMethodDeclarator) {
  // for checking the value of the external input
  Store the method parameter(s);
 }
 public Objective visit(ASTAdditiveExpreesion) {
    Identify the SQL string, Literal;
    Check the existence of the variable, Name;
    Store the variable(s) and call visit(ASTMethodDeclaration);
    if (the string is assigned the value of the external input) {
       Check whether the SQL is execute through the method;
       if (the sting is output)
         // for SQL Injection weakness diagnosis
         Print diagnostic message;
} } }
```

3.2.4. Public 메소드부터 반환된 Private 배열

'Public 메소드부터 반환된 Private 배열' 보안약점은 Private 배열이 Public 메소드의 반환값으로 설정되어 Private 속성에 맞지 않게 임의로 접근·변경될 수 있는 보안약점으로 Juliet 시험코드에는 해당 시험코드가 존재하지 않는다. 그러나, 해당 보안약점은 다음 표 12와 같은 예제코드로 표현할 수 있다.

표 12. 예제코드(Public 메소드부터 반환된 Private 배열)
Table 12. The sample code(Private Array-Typed Field Returned From A Public Method)

```
public class ReturnPrivateArray {
 private byte[] flaw = {};
 public byte[] getFlaw( ) {
    // do something
    return flaw;  /* Weakness exists */
 }
 byte[] tmp_data = getFlaw();
}
```

'Public 메소드부터 반환된 Private 배열' 보안약점의 경우, Pulic 메소드의 반환값이 Private 속성의 배열이면 해당 보안약점이 존재한다고 할 수 있기 때문에 진단도구로 진단하는 방법(표13)은 Public 메소드의 반환값의 속성 점검을 통해 해당 보안약점의 존재여부를 확인하는 것이다.

표 13. 진단도구의 진단로직(Public 메소드부터 반환된 Private 배열)
Table 13. Diagnosis Logic for Tool(Private Array-Typed Field Returned From A Public Method)

① Check the existence of Public method which returns array ② If exist, check whether the 'array' variable is declared as Private in Class ③ If declared, the tool prints the message that the weakness exists

따라서, PMD의 진단규칙으로 구현하기 위해 AST 구조로 변환된 JAVA 소스코드 내에서 소스코드의

구조를 기반으로 변수, 메소드, 메소드 반환값 등의 속성을 비교하도록 XPath로 구현할 수 있으며, 관련 진단규칙은 다음 표 14와 같다.

표 14. PMD 진단규칙(Public 메소드부터 반환된 Private 배열)
Table 14. PMD's Rule(Private Array-Typed Field Returned From A Public Method)

```
//MethodDeclaration[@Public='true']
  [ResultType[@returnsArray='true']]
  /Block/BlockStatement/Statement[ReturnStatement/Expression
    /PrimaryExpression /PrimaryPrefix/Name
    [@Image=preceding::ClassOrInterfaceBodyDeclaration
    /FieldDeclaration[@Private='true']
    [Type/ReferenceType[@Array='true']]
    /VariableDeclarator/VariableDeclaratorId/@Image]]
```

Ⅳ. 시험 결과

제3장에서 설명한 주요 보안약점에 대한 진단규칙은 보안약점의 특징에 따라 PMD 진단규칙 구현 방법인 JAVA 및 XPath를 선별적으로 이용하여 구현하였으며, 다음 표 15와 같다.

표 15. PMD 진단규칙 구현방법
Table 15. Implementation of PMD's Rules

Weakness	Diagnostic Mechanism	Lang.
XSS(W1)	Data·Control flow	JAVA
Info. Leakage of System Data(W2)	Pattern matching	XPath
Info. exposure through an error message(W3)	Structure, Pattern matching	XPath
SQL Injection(W4)	Data·Control flow	JAVA
Private Array-Typed Field Returned From A Public Method(W5)	Structure	XPath

PMD에 구현한 진단규칙의 성능분석을 위해 Juliet 코드의 기본 유형(예, *_01.java)의 예제코드를 기반으로 시험하였으며, 'Public 메소드부터 반환된 Private 배열' 보안약점만 표 12의 예제코드를 활용하였다. 또한, 다른 공개용 진단도구와 비교를 위해 다음 표 16과 같은 JAVA 기반의 진단도구도 같이 시험을 하였다. FindSecurityBugs는 FindBugs에 Security 모듈을 추가적으로 적용한 것이기 때문에 FindBugs 대신 FindSecurityBugs로만 시험을 수행하였다.

표 16. 공개용 진단도구(JAVA)[13]
Table 16. Open Source Tool(JAVA)[13]

Tools	Version	Plugin
FindBugs	2.0.2	Eclipse
FindSecurityBugs	1.0.1	FindBugs module
JLint	3.0	CLI
LAPSE+	2.1.8	Eclipse
PMD	4.0.0	Eclipse
Yasca	2.21	CLI

PMD의 진단규칙에 대한 성능시험 결과, 개선된 PMD가 다른 도구에 비해 주요 보안약점을 보다 잘 진단(그림 3)하였으며, 세부 분석내용은 다음과 같다.

'XSS(W1)' 보안약점의 경우, 개선전 PMD, Jlint, Yasca 진단도구만 해당 보안약점을 식별하지 못하였다. FindSecurityBugs는 HttpServletRequest.getParameter() 메소드를 통한 외부 입력유형과 연계된 보안약점만 식별하였다. LASPSE+는 HTML 스크립트문의 일부를 정해진 안전한 값으로 입력되는 경우도 보안약점으로 식별하였다. 이는 LASPSE+는 외부 입력값에 대한 분석 없이 특정 API 등을 기반으로 단순 패턴매칭을 수행하여 보안약점을 식별하기 때문에 정탐 및 오탐이 동일하게 발생하는 것으로 판단된다. 개선된 PMD는 외부 입력값이 다른 메소드를 통해 필터링 되거나 토큰화 되는 경우를 제외한 예제코드에 대해 적절하게 보안약점을 식별하였다.

'시스템데이터 정보노출(W2)' 보안약점의 경우, 개

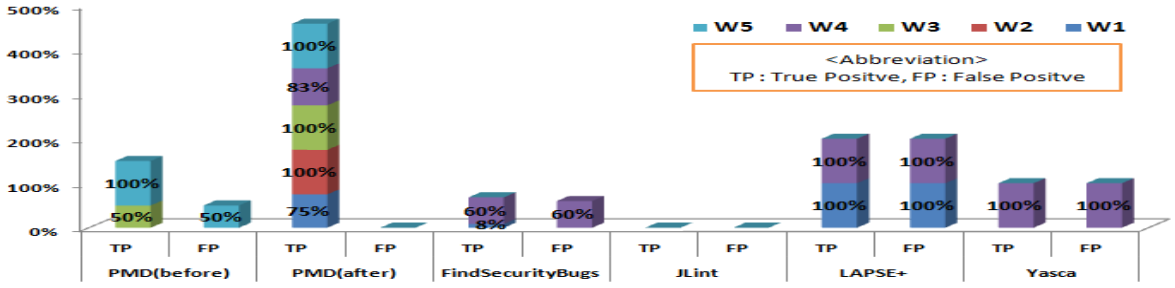

그림 3. 주요 보안약점에 대한 진단도구 진단성능
Figure 3. Diagnosis Capability of Security Static Analysis tools for major weaknesses

선된 PMD 이외의 진단도구는 해당 보안약점을 식별하지 못하였다.

‘오류메시지를 통한 정보노출(W3)’ 보안약점의 경우, PMD 이외의 진단도구는 해당 보안약점을 식별하지 못하였다. 그러나, 진단규칙 개선전의 PMD는 매개변수가 없는 printStackTrace() 메소드만 식별하기 때문에 ‘e.printStatckTrace(request.getWriter())’과 같이 매개변수에 변수타입이 오는 경우 보안약점으로 식별하지 못하였다.

‘SQL 삽입(W4)’ 보안약점의 경우, 개선전 PMD와 Jlint 진단도구만 해당 보안약점을 식별하지 못하였다. FindSecurityBugs는 외부 입력값에 대한 분석 없이 해당 보안약점을 발생시킬 수 있는 Statement.execute() 등의 메소드를 기반으로 보안약점을 식별하기 때문에 정탐 및 오탐이 동일하게 발생하였다. 또한, Statement.executeBatch() 등의 메소드에 대한 패턴이 부재하여 관련 예제코드는 분석하지 못하였다. LAPSE+ 및 Yasca는 FindSeucityBugs와 동일하게 외부 입력값에 대한 분석 없이 해당 보안약점을 유발하는 특정 API 등의 패턴을 기반으로 보약약점을 식별하기 때문에 정탐 및 오탐이 동일하였다. 개선된 PMD는 외부 입력값이 다른 메소드를 통해 필터링 되거나 토큰화 되는 경우를 제외한 예제코드에 대해 적절하게 보안약점을 식별하였다.

‘Public 메소드로부터 반환된 Private 배열(W5)’ 보안약점의 경우, PMD 진단도구만 해당 보안약점을 식별하였다. 그러나, 개선전 PMD의 경우, private 배열이 아닌 public 배열을 반환하는 예제코드도 보안약점으로 식별하는 오탐이 존재하였다.

V. 결 론

본 논문에서는 SW 개발단계에서 SW 보안성을 향상시킬 수 있는 주요 보안약점에 대한 자동화된 진단방법 및 PMD 기반의 진단규칙을 제시하였다. 그리고, 본 논문에서 제시한 주요 보안약점에 대한 진단규칙을 PMD에 구현하여 JAVA 기반의 공개용 진단도구와 함께 수행한 성능시험 결과를 제시하였다.

본 논문에서 제시하는 주요 보안약점별 PMD 기반의 진단규칙은 SW 개발시 SW 개발자 등에 의해 활용될 수 있으며, 진단도구 개발업체의 진단규칙 개선시 활용될 수 있다.

향후, 본 논문에서 제안한 보안약점 이외의 항목에 대한 진단규칙을 추가로 구현하고, 모바일 어플리케이션 관련 보안약점을 분석하여 관련 진단규칙을 구현하여 제안할 예정이다.

References

[1] R. K. McLean, “Comparing static security analysis tools using open source software,” in Proc. *IEEE 6th Int. Conf. SW Security Reliability Companion (SERE-C),* pp. 68-74, Gaithersburg, U.S.A., June 2012.

[2] MOPAS, *A guide to secure software development*, Publication No.11-1311000-000330-10, Retrieved June, 1, 2012, from http://www.mopas.go.kr.

[3] MOPAS, *Guidelines on building and operating Information Systems*, MOPAS Notification No.2012-25, June 2012.

[4] M. Johns and M. Jodeit, “Scanstud: a methodology for systematic, fine-grained evaluation of static analysis tools,” in *Proc. IEEE 4th ICSTW*, pp. 523~530, Berlin, Germany, Mar. 2011.

[5] CERT, *CERT Secure Coding Standards*, Retrieved July, , 2013, from https://www.securecoding.cert.org/.

[6] MOPAS, *A guide to diagnose software weakness*, Publication No.11-1311000-000395-14, Retrieved May, 12, 2012, from http://www.mopas.go.kr

[7] MOPAS and NIA, *eGovernment Standard Framework*, Retrieved June, 10, 2013, from http://www.egovframe.kr.

[8] SourceForge, *PMD*, Retrieved May, 1, 2013, from http://pmd.sourceforge.net.

[9] SourceForge, *FindBugs*, Retrieved Dec., 11, 2012, from http://findbugs.sourceforge.net.

[10] J. Bang and R. Ha, “Evaluation methodology for weakness diagnostic tool of e-GOV software,” *J. Korean Inst. Commun. Inform. Sci. (KICS)*, vol. 38C, no. 4, pp. 335-343, Apr. 2013.

[11] S. Kim, J. Joo, G. Lee, and G. Kwon, “Implementation of code vulnerabilities checker for secure software,” in *Proc. Korean Soc. Internet Inform. (KSII) Conf. 2010*, pp. 605-608, Jeju Island, Korea, June, 2010.

[12] NIST, *Software Assurance Metrics And Tool Evaluation*, Retrieved Feb., 20, 2013, from http://samate.nist.gov.

[13] J. Bang and R. Ha, "Comparing open source static security analysis tools based on software weakness," in *Proc. Korean Ist. Inform. Sci. Eng. Korea Computer Congress (KIISE KCC) 2013,* pp. 753-755, Jeju Island, Korea, June 2013.

[14] MOPAS and KISA, *2013 SW Weakness Dignosis Consultant Training Course Materials*, 2013

방 지 호 (Jiho Bang)

1996년 2월 홍익대학교 컴퓨터공학과 졸업
2001년 8월 홍익대학교 컴퓨터공학과 석사
2004년 3월~현재 홍익대학교 컴퓨터공학과 박사과정
2001년 7월~2009년 7월 한국정보보호진흥원 선임연구원
2009년 7월~현재 한국인터넷진흥원 책임연구원
<관심분야> 모바일SW보안, 모바일컴퓨팅, 실시간시스템 등

하 란 (Rhan Ha)

1987년 2월 서울대학교 컴퓨터공학과 졸업
1989년 2월 서울대학교 컴퓨터공학과 석사
1995년 4월 University of Illinois at Uubana-Champaign 전산학 박사
1989년 3월~1990년 7월 한국통신 전임연구원
1995년 9월~현재 홍익대학교 컴퓨터공학과 교수
<관심분야> 모바일컴퓨팅, 실시간시스템, SW보안 등