정보시스템 소스코드 보안성 강화를 위한 PMD Rule-set 의 확장과 분석: 생명보험 시스템의 사례 중심으로

남진오*, 최진영**
*, **고려대학교 컴퓨터 정보통신대학원 소프트웨어공학과
Email : jinonam@korea.ac.kr, choi@formal.korea.ac.kr

Analysis and extension of the PMD rule-set for the source code security strengthening of IT systems

Jin-O Nam*, Jin-Young Choi**
*, **Dept. of Software Engineering, Graduate School of Computer @ Information Technology, Korea University

요 약

최근 개인정보 유출 등으로 인해 정보시스템의 보안약점 및 소스코드 품질에 대한 관심이 높으며, 특히 개인자산과도 관련된 금융 정보 시스템의 경우에는 더욱 높다. 해당 시스템의 보안성 강화를 위해서는 개발단계에서부터 보안취약점과 코드의 품질을 높일 수 있는 정적분석 기반의 진단도구 활용이 중요하다. 많은 분야에서 진단도구의 활용이 이루어지고 있지만 금융 정보시스템의 경우 다른 SW 와 특성이 다르기 때문에 추가적인 진단규칙이 반영된 진단도구의 활용이 필요하다. 본 논문은 여러 진단도구 중 전자정부개발에 사용하고, 비교적 진단규칙 추가가 용이한 PMD 에 추가 진단규칙을 반영한 후 생명보험 정보시스템에 적용하고 이에 대한 PMD 검출 계수를 분석한다.

1. 서론

금융업계에서는 통상 금융 자체의 위험(Risk)에 대한 연구는 활발히 진행되고 있지만, 실제 그 금융을 운용하는 금융정보시스템의 위험에 대한 관리는 사후처리에 집중되고 있다[1].

SW 개발단계에서 보안약점을 제거하는 것이 SW 개발완료 및 배포 이후에 발생할 수 있는 보안취약점의 패치 비용을 효과적으로 감소시킬 수 있는 방법이지만[2], SW 는 그 특성 상 사전점검이 어렵기 때문에 process 품질확보나 테스트를 통해서 개발완료 전 결함 추출과 보안약점 강화에 주력하고 있다.

장애발생 시 국가적, 사회적으로 큰 파급효과와 막대한 손실을 발생시키는 금융정보시스템[3]의 보안약점 강화를 위해서는 신뢰성이 보장된 정적 분석 기반의 도구를 사용하는 것이 필요하며, 다양한 보안약점을 진단할 수 있는 진단규칙을 보유하는 것이 필요하다.

기존 C 나 COBOL 로 개발되어 있던 금융정보시스템, 그 중에서도 보험업계의 시스템은 차세대 프로젝트를 통해[4] 현행 거의 대부분이 JAVA 를 이용하여 개발되어 있다[5]. 따라서 JAVA 소스코드에 대한 여러 정적 분석도구 중 2013 년 하반기에 IT 보안인증사무국에서 정보보호제품 보안성 평가, 인증(CC 평가, 인증)된 진단도구를 금융시스템에 적용하는 것이 소스코드의 품질확보와 보안약점 강화 측면에서 타당해 보인다.

정부에서 개발, 보급하는 전자정부 표준 프레임 워크에서 제공하는 분석도구로는 PMD, FindBugs 가 있다. 이 중에서 PMD 는 코딩 품질 관점의 별도 진단규칙을 선별하여 전자정부용으로 배포하고 있으며[6], 주요 보안약점에 대한 진단규칙(Rule)을 추가로 개발하여 시스템에 적용할 경우 타 진단도구에 비해 효과가 높을 수 있다.

본 논문에서는 전자정부 프레임워크에서 사용을 권고하고 있는 PMD 를 JAVA 로 구현된 생명보험 정보시스템에 적용했을 때와 기존 진단규칙에 주요 보안약점 진단규칙을 추가하여 적용할 경우 SW 보안약점 강화에 어떠한 긍정적 효과를 줄 것인지 분석한다.

본 논문은 2 장에서는 관련동향에 대해 살펴보고, 3 장에서는 PMD 의 Rule-set 중 대표적인 것들과 생명보험 정보시스템에 PMD 를 적용 시 추가적으로 구현이 필요한 진단규칙을 제시한다. 4 장에서는 추가 구현된 Rule-set 을 생명보험 정보시스템에 적용한 결과에 대해 설명하고, 5 장에서 결론을 맺는다.

2. 관련동향

본 장에서는 금융정보시스템에서의 오류 원인을 살펴보며, JAVA 소스 코드 진단도구의 종류와 그 중에서도 PMD 에 초점을 맞춰 살펴보고자 한다.

구 분		건수	비율	비율(소계)
요구정의	요구기능 누락	2	2.7%	9.5%
	기능정의 부정확	5	6.8%	
설계	부정확한 IF 설계	2	2.7%	14.9%
	IF/Timing 오류	1	1.4%	
	초기 설정 값 오류	5	6.8%	
	변수타입 오류	3	4.1%	
구현	Logic 구현 오류	24	32.4%	64.9%
	변수 값 오류	1	1.4%	
	데이터처리 오류	11	14.9%	
	데이터 초기화 오류	2	2.7%	
	계산오류	5	6.8%	
	함수/변수 사용 오류	5	6.8%	
테스트	테스트범위 오류	1	1.4%	1.4%
이행	파일복사/이동 오류	3	4.1%	9.5%
	프로그램 복사/이동 오류	4	5.4%	
합계		74	100.0%	100.0%

<표 1> 금융정보시스템 SDLC 상 장애현황

2-1. 금융정보시스템 오류원인

금융관련 업무의 경우 업무에 대한 정의가 기존 사용되었던 정의를 재가공하여 사용하는 경우가 많아 일반적으로 명확히 정의되기 때문에 금융정보시스템에서는 대부분의 오류가 구현단계의 문제로 인해 발생하게 된다. 즉, 오류가 발생했을 경우, SW 의 요구사항이 명확하기 때문에 요구정의나 요구분석, 설계 단계 보다는 구현 및 테스트, 이행 단계가 해당 오류의 원인이 될 가능성이 매우 높음을 의미한다. <표 1>에서 보여지는 것과 같이 기존 연구에서도 SDLC(Software Development Life Cycle) 상 장애 분석 결과 구현 단계의 오류로 인해 장애 발생 가능성이 현저히 높음을 확인 할 수 있다[7]. 이는 다른 논문에서 발표한 일반적인 SW 오류 발생 원인이 "명세(Specification)의 오류"가 56%, "설계(Design)의 오류"가 27%, "구현(Code)의 오류"가 10%, "기타 오류"가 7%를 차지하는 것과는 매우 상이한 결과이다[8]. 따라서 다른 SW 와 달리 금융정보시스템의 SW 결함은 주로 구현 단계에서 많이 발생하기 때문에 일반적인 SW 결함 예방활동 적용보다는 금융 도메인의 특성을 고려하여 Code Inspection 강화 등의 조치가 필요함을 알 수 있다.

2-2. JAVA 소스 코드 진단도구

NIST(National Institute of Standards and Technology)에서는 SAMATE(Software assurance Metrics and Tool Evaluation) 프로젝트를 통해 JAVA 소스 코드의 보안약점 진단도구를 다음 6 개로 한정하고 있다[9]. PMD, FindBugs, LAPSE+, Yasca, JLint, FindSecurityBugs.

이 6 개의 도구들은 행정안전부에서 발간한 “소프트웨어 보안약점 진단가이드(2012)(이하 진단가이드)”에 따라 의무화된 SW 보안약점 기준 43 개[10] 중 유형별로 일부 진단규칙만을 가지고 있으며, PMD 의 경우 “에러처리”에 관한 2 가지의 진단규칙(오류메시지 통한 정보 노출, 오류상황 대응 부재), “캡슐화”에 관한 1 가지의 진단규칙(Private 배열에 Public 데이터 할당) 등 3 가지의 진단규칙을 기본적으로 가지고 있다.

PMD 는 JavaCC 컴파일러로 소스 코드를 분석하여 AST(Abstract Syntax Tree)를 구성하고, 그 AST 를 기반으로 소스의 구조와 의미 분석, 데이터 흐름 분석 등을 수행할 수 있다. PMD 는 270 여개의 진단규칙(Rule)을 가지고 있으며, 이 진단규칙들은 진단규칙세트(Rule-set)으로 되어있다. 또한 이 진단규칙은 XPath 기반으로 구현되어 있으며 추가 구현도 가능하다.

3. PMD Rule-set 연구

본 장에서는 PMD 에서 기본적으로 제공되는 Rule-set 중 대표적인 것들을 살펴보고, 생명보험 정보시스템에 PMD 적용 시 “진단가이드”에서 제시하는 SW 보안약점 기준을 기반으로 추가 구현이 필요한 Rule 에 대해 설명한다.

3-1. PMD Rule-set

PMD 의 Rule-set 중 대표적인 것은 <표 2>과 같이 크게 15 가지가 있으며, 이 중에서 몇 가지는 완벽하다고 할 수 없는 것도 존재한다[11].

3-2. 추가로 필요한 Rule 연구

PMD 의 기본 Rule-set 만으로는 “진단가이드” 기준 보안약점에 대해 3 가지의 규칙만을 만족시키고 있다. 따라서 생명보험 정보시스템에 PMD 적용 시에는 기본적인 진단규칙 외에도 “진단가이드” 에서 제시한 43 개의 보안약점을 만족시키면서 금융정보시스템의 특성에 맞는 진단규칙의 추가가 필요하다. 또한, 본 논문에서는 스프링 프레임워크 기반의 생명보험 정보시스템에 PMD 를 적용하므로, 해당 프레임워크에 적합성도 고려된 진단규칙이 필요하다.

Rule-set	설명
Basic	대부분의 개발자들이 동의하는 기본규칙 검사
Braces	for, if, while, else 문장이 괄호를 사용하는지 여부 검사
Clone Implementation	clone() 메소드에 대한 규칙 검사
Code size	과도하게 긴 메소드, 너무 많은 메소드를 가진 클래스, 리팩토링에 대한 유사한 후보들을 위한 검사
Controversial	변수에 null 할당, 메소드에서 온 다중의 리턴 포인트 sun 패키지에서 임포팅 검사
Coupling	클래스들간 과도한 커플링 표시 검사
Design	디자인 원리 검사
Finalizers	finalize() 메소드 관련 문제 검사
Import statements	임포트 문장 관련 문제 검사
Javabeans	JavaBeans 코딩 규약을 위배하는 JavaBeans 컴포넌트 검사
JUnit tests	테스트 케이스와 테스트 메소드 관련 특정 문제 검사
Naming	표준 자바 네이밍 규약을 위한 검사
Strict exceptions	예외 처리 검사
String and StringBuffer	스트링 관련 문제 검사
Unused code	사용되지 않는 private 필드와 로컬 변수, 접근할 수 없는 문장, 호출되지 않는 private 메소드 등 검사

<표 2> PMD Rule-set

따라서 본 논문에서는 생명보험 정보시스템에 PMD 적용 시 <표 3>과 같은 추가진단규칙을 선정하였다. 진단규칙 “ DoNotUseUserTreansactionOrThread ” 는 WAS 상에서 Thread 사용 시 WAS 보다 상위의 시스템 메모리를 사용하게 되어, 경우에 따라 WAS 자체를 종료시켜 버리고, 그렇게 Thread 로 인해 종료된 경우 원인파악이 어렵기 때문에 추가하였다. 또, “ SystemPrintln ” 진단규칙의 경우는 불필요한 System.out.println()문은 시스템에 부하를 발생시키기 때문에 기본 PMD 진단규칙에 System.out 으로 시작하는 부분을 모두 찾도록 보안하였다. 진단규칙 중 “UnreleasedResource”는 오류 조건 및 기타 예외 상황에서 리소스 해제를 지속적으로 하지 않을 경우 사용 가능한 리소스를 모두 소진할 수 있으므로 추가하였으며, “UseOwnSystemCode” 규칙은 본 논문에서 이용한 시스템의 연계시스템에서 해당 인터페이스 별로 사용할 수 있는 요청 시스템이 누구인지 등록된 요청 시스템만 통과하도록 구현되어 있기 때문에 추가하였다.

4. PMD Rule-set 적용 결과 및 분석

본 장에서는 3-2 절에서 제시한 추가적인 Rule-set 이 반영된 PMD 를 생명보험 정보시스템에 적용 후, 기존에 기본적으로 제공된 Rule-set 의 PMD 가 적용되었을 때와 비교 분석하여 본 논문 제시한 보안약점 강화 방면에서의 효과에 대해 설명한다.

Rule	설명
AvoidNarrowConversionBetweenPrimitiveTypes	기본 유형을 하위 정밀도로 캐스트 하면 정보가 손실될 수 있기 때문에 하위 캐스팅을 금지.
AvoidRethrowingExceptionForLI	단순히 예외를 재전송(rethrow)하는 catch 구문을 금지.
CreateClassInMethod	스프링 Bean 인 경우 참조클래스는 메소드 내에서 생성하도록 규정.
DoNotCallNewInitializerForBeanClass	Bean 을 new 연산자로 생성하는 케이스를 금지(해당 시스템 표준 상 Bean 은 Ctl, SubCtl, BTO).
DoNotUseServiceAnnotationInGeneralClass	General 클래스를 Bean 으로 정의 금지(화면에서 호출하는 시스템 공통유틸은 예외허용).
DoNotUseUserTreansactionOrThread	트랜젝션을 코드에서 handling 하거나 Thread 를 직접 handling 하는 것은 금지.
HardcodedPassword	하드코딩 된 암호 금지.
ReturnEmptyArrayRatherThanNull_LI	NullPointerException 발생 가능성이 있기 때문에, null 배열 리턴 금지(대신 빈 배열을 리턴).
ServiceAnnotationNameMustBeClassName	명명 표준 상 Bean 명과 해당 Class 명은 동일하도록 규정.
SystemPrintln	개발 시 디버그를 위한 System.out.println() 문 삭제 규정.
UnconditionalIfStatement_LI	항상 참인 if 또는 while 조건 금지.
UnreleasedResource	finally 블록에서 명시적으로 리소스 close 규정.
UseOwnSystemCode	해당 업무에 할당된 요청 시스템코드를 setting 하도록 규정.

<표 3> 추가 PMD Rule

4-1. 추가 Rule 의 구현

3-2 절에서 제시한 Rule 을 XPath 로 구현하였고, 소스의 일부분은 <그림 1>과 같다.

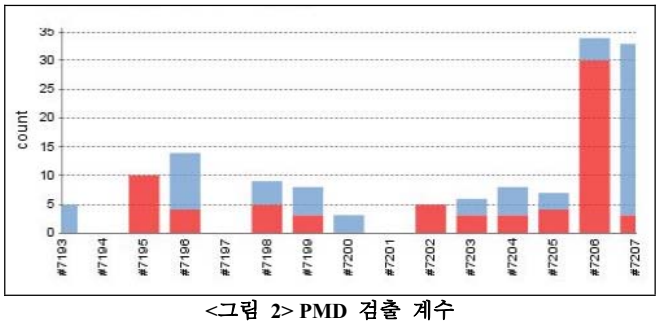
```
<rule class="net.sourceforge.pmd.rules.XPathRule" dfa="false"
externalInfoUrl="http://pmd.sourceforge.net/rules/strictexcep
tion.html#AvoidRethrowingException" message="A Catch
statement that catches an exception only to rethrow it should
be avoided." name="AvoidRethrowingExceptionForLI" since="3.8"
typeResolution="true">
    <description>
        Catch blocks that merely rethrow a caught exception
only add to code size and runtime complexity.
    </description>
    <priority>1</priority>
    <properties>
        <property name="xpath">
            <value><![CDATA[

//CatchStatement[FormalParameter
/VariableDeclaratorId/@Image = Block/BlockStatement/Statement
/ThrowStatement/Expression/PrimaryExpression[count(PrimarySuf
fix)=0]/PrimaryPrefix/Name/@Image
and count(Block/BlockStatement/Statement) =1
 and not(//PackageDeclaration/Name[contains(@Image,'.data')])
]

            ]]></value>
        </property>
    </properties>
    <example><![CDATA[
package li.ch.data.cont.impl; // exlude !!
public class Foo {
 void bar() {
  try {
  // do something
  } catch (SomeException se) {
    throw se;
  }
 }
}

    ]]></example>
</rule>
```

<그림 1> 구현된 추가 PMD Rule 의 일부분

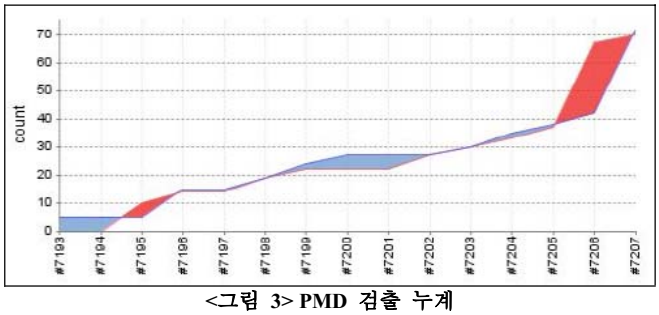
4-2. 추가 Rule 이 적용된 PMD 분석 결과

생명보험 정보시스템 처리계 코어 중 일부에 본 논문에서 제시한 추가 Rule 이 반영된 PMD 를 적용한 결과를 CI Tool 인 "HUDSON"을 이용하여 분석하였다. 빌드 #7205 까지는 기존 PMD 적용 결과이며, 빌드 #7206 부터 추가 Rule 이 반영되었다. <그림 2> PMD 검출계수와 <그림 3> PMD 검출누계에서 확인되듯이 PMD 검출이 현저히 증가하였다.

PMD 검출 증가로 인해 정보시스템 개발 및 유지보수 인력은 해당 부분에 대해 좀 더 상세하게 소스코드 분석이 가능할 것이고, 그 분석을 통해 보안 약점 및 코딩 품질 향상을 할 수 있다



<그림 2> PMD 검출 계수



<그림 3> PMD 검출 누계

5. 결론 및 향후 연구

본 논문에서는 생명보험 정보시스템과 "진단가이드"에 따라 의무화된 43 개 항목 중 일부를 고려하여 PMD Rule-set 에 추가적인 Rule 을 구현하고 처리계 코어 시스템 일부에 반영 후 그 결과를 분석하였다. 분석 결과, 구현된 Rule 적용 시 PMD 검출이 현저히 증가함을 확인하였으며, 이 증가로 인해 정보시스템의 소스코드 결함 및 보안약점 강화가 가능해지고, 결과적으로 정보시스템의 품질이 향상되게 된다.

본 논문에서는 HighError Level 에 해당하는 Rule 만을 추가하였다. 향후 연구는 보다 높은 수준의 HighError Level Rule 과 더불어 Warning, Information Level 의 Rule 을 PMD 에 적용하고 대상을 전체 생명보험 정보시스템으로 확장하여 연구를 진행함으로써 생명보험 정보시스템에 가장 적합한 수준의 Rule-set 을 제안할 예정이다.

참고문헌

[1] H. S. Park, "Recent Domestic Bank's Risk Management and Future Works," The Bank of Korea, Finance Systems Review, Vol.7, p.85~95, 2002

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

[3] "SW Engineering White Book," p.5~6, National IT Industry Promotion Agency, 2011

[4] 송주영, "자바(Java)의 등장. 내년 금융 IT 기술변화 주도 예고", 디지털데일리, 2007, from http://www.ddaily.co.kr

[5] 이상일, "보험, 증권 차세대 본격화, 은행권은 답보상태", 디지털데일리, 2013, from http://www.ddaily.co.kr

[6] "eGovernment Standard Framework," MOPAS and NIA, 2013, from http://www.egovframe.kr

[7] T. H. Kang and S. Y. Rhew, "A Proposal for Improvement and Current Situation of Risk Management on Financial Information System," 2012

[8] "Information System Risks and Risk Factors: Are they mostly about Information Systems?", Vol.14, Communications of the Association for Information Systems, 2004

[9] "Software assurance Metrics and Tool Evaluation," NIST, 2013, from http://samate.nist.gov

[10] "소프트웨어 보안약점 진단가이드", p.10, 218~221, 행정안정부, 2012

[11] "SourceForge.net, Rule Sets," SourceForge, 2013, from http://pmd.sourceforge.net/pmd-5.0.5/rules/index.html