제28권 제1호(통권102호) 2019년 6월 ISSN 1738-6322

소프트웨어공학소사이어티 논문지

Journal of Software Engineering Society

VOLUME 28, NUMBER 1, June 2019

한국정보과학회
KOREAN INSTITUTE OF INFORMATION SCIENTISTS AND ENGINEERS

SOFTWARE ENGINEERING SOCIETY

Automatic Usage Profiling을 통한 초기 앱 실행 속도 개선 방법†

(Improving application startup time by automatic profiling)

채 향 석 ‡ 백 종 문 §

(Hyangseok Chae) (Jongmoon Baik)

요 약 Google은 2009년 Bytecode로 구성된 Dex(Dalvik Executable)를 Dalvik Runtime의 Interpreter가 실행하는 형태의 Android를 공개하였다. 이후로 Interpreter 실행 속도 개선을 위해 JIT(Just-in-time) 컴파일 기술을 적용하였고 Lollipop(Android 5.0)부터는 Dalvik Runtime을 대체하여 ART Runtime을 제공하여 AOT(Ahead-of-time) 컴파일 지원을 통해 앱 설치 이후부터 Bytecode가 아닌 Native code로 동작하도록 함으로써 성능을 높일 수 있게 되었다. 하지만 앱 설치/업데이트 시점에 모든 대상을 컴파일하는 AOT 컴파일은 시간이 오래 걸리고 메모리/CPU 자원을 많이 사용함에 따라 느려지고 발열을 유발하여 사용자 불편함을 초래하였다. 시간이 지날수록 더 복잡하고 큰 코드를 지닌 앱들이 많이 등장함에 따라 AOT 컴파일로 인해 발생하는 문제들이 더 많이 발생하게 되었고, Nougat(Android 7.0)부터는 이를 개선하여 AOT 컴파일을 앱 설치/업데이트 시점에 모두 수행하지 않고 최적화 시점을 나중으로 미루고 실제 사용자의 사용 기록인 Profile을 사용하는 Profile-guided 컴파일 방법을 통해 문제를 회피하고 있다. 이 연구에서는 앱 실행 속도를 설치 직후부터 개선할 수 있도록 하기 위해 Profile에 따른 앱 실행 속도의 특성을 파악하여 앱 실행 속도를 개선할 수 있는 Profile을 앱 개발 시점에 자동 생성하는 방법과 자동 생성한 프로파일을 APK에 포함하고 앱 설치/업데이트 시점에 활용하여 최적화를 할 수 있는 방법을 제안한다. 제안하는 방법을 통해 앱 설치 시점에 Profile에 기반하여 선택적으로 컴파일할 수 있으므로 설치 시점에 발생하는 사용자 불편을 최소화할 수 있으며 앱 설치 이후 Native code 실행을 통해 앱 실행 속도를 최초 실행부터 개선할 수 있다.

키워드 : Profiling, AOT, JIT, Optimization

Abstract Google released an initial version of Android that runs Dex(Dalvik Executable) through the Dalvik Runtime. Since Dalvik Runtime is based on interpreter, JIT(Just-in-time) compilation has been applied to improve performance. After Lollipop(Android 5.0) Dalvik Runtime has replaced with ART Runtime which support AOT(Ahead-of-time) compilation of Dex into Native Code. The latest Android has a problem that the application execution speed is slow until the AOT compilation is completed according to the actual usage record after the installation of the app. To improve the problem we have investigate the characteristics of profile that can improve the execution speed of the application and generate the profile automatically. Finally we propose a method that can optimize the application at install time. With the proposed method we can optimize selectively at install time and can help improving the execution speed of the app from the initial execution.

Key words : Profiling, AOT, JIT, Optimization

1. 서 론

Google은 2009년 Cupcake(Android 1.5)를 시작으로 매해 새로운 기능이 포함된 새로운 버전의 Android를 공개하고 있다. 처음 공개될 시점의 Android는 Bytecode인 Dex(Dalvik EXecutable)를 Interpreter로 수행하는 형태로 동작하였다. 이후 Interpreter의 성능을 개선하기 위한 기술도 점진적으로 적용되었으며 대표적으로 Froyo(Android 2.2)부터 제공된 JIT(Just-in-time) 컴파일을 들 수 있다. Dalvik Runtime의 성능은 Interpreter에 기반한 실행이라는 태생적인 한계가 있었으므로 이를 극복하기 위해 Lollipop(Android 5.0)부터는 새로운 ART Runtime을 적용하였고 ART Runtime은 AOT(Ahead-of-time) 컴파일을 지원하므로 JIT컴파일 대비 우수한 성능을 빠른 시기에 확보할 수 있게 되었다.

Nougat(Android 7.0)부터는 ART Runtime을 통해 최적화가 수행되기 위해서 사용자의 실제 앱 실행 이력이

‡ 비 회 원 : LG전자 연구원
jen211@kaist.ac.kr
§ 종신회원 : KAIST 컴퓨터학부 부교수
jbaik@kaist.ac.kr
논문접수 : 2019년 2월 27일
심사완료 : 2019년 5월 8일

있어야 하며 또한 언제 최적화 될지 모르는 시점까지 사용자는 앱의 성능이 최대로 나오지 않는 Interpreter 상태로 사용해야 하는 문제를 갖고 있다.

본 연구에서는 Nougat(Android 7.0)부터 존재하는 문제를 개선하기 위하여 Profile을 자동 생성할 수 있는 방법과 Profile을 APK에 포함하여 앱 설치/업데이트 시점에 Profile을 활용한 최적화 방법을 제안한다. 제안하는 방법을 통해 앱 설치 직후부터 Interpreter 동작과 Profiling에 따른 성능 감소 없이 Native code 수행을 통해 앱 실행 속도를 개선할 수 있다.

본 연구의 구성은 다음과 같다. 2장에서는 Android 7.0부터 포함된 Profile-guided 컴파일 기술이 Android 최신버전에서 어떻게 동작하는지를 설명하며 이를 위해 Profile에 포함된 정보가 무엇 인지와 Profile을 활용한 최적화 방법, 최적화된 결과물을 활용하는 배경기술을 설명한다. 3장에서는 제안하는 방법을 구체화하여 Profile의 특성을 파악하고 앱 실행속도를 개선할 수 있는 Profile을 자동 생성하는 방법과 Profile을 APK에 포함하여 이를 앱 설치 시점에 활용할 수 있는 방법을 설명한다. 4장에서는 제안한 방법을 적용하여 앱 실행속도가 최초 실행부터 개선되는지를 측정한다. 5장에서 결론을 기술한다.

2. 배경기술

2.1 Profile 기록

Android 7.0부터는 그림1 과 같이 설치 시점에 APK에 포함된 Byte Code에 대해 최소한의 검증만 수행하고 실행하므로 앱 설치는 빠르고 설치 이후 사용 용량은 적게 사용한다. 이후 앱 실행시점에 Profiling을 수행하여 JIT으로 성능을 개선하고 Profile을 파일로 저장하여 최적화에 사용한다. PGO(Profile-Guided Optimization)는 앱과 독립적으로 Daemon에 의해 구동되며 Profile을 활용한 AOT 컴파일을 통해 다음 앱 실행 속도를 개선한다.

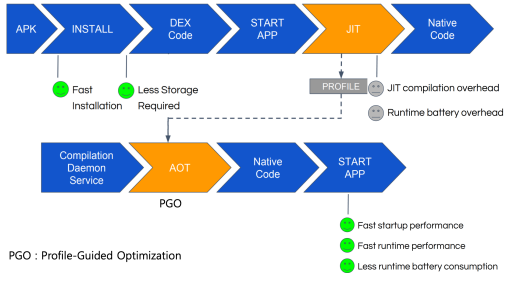


그림 3 : 최신 Android Profile 기록

2.1 Profile 기반한 최적화

Profile에는 Class와 Method 정보가 기록되며 Scheduling(충전중 & 24시간 주기 & 화면꺼짐후 71분 경과)에 따라 Background 최적화에 활용된다. 그림2 와 같이 앱 실행 이후 5초 이내에 Loading한 Class정보는 초기화된 Image형태로 Dump/Restore하여 앱 실행 시 Class Loading 시간을 줄이는데 활용되며 앱 실행 이후 5초 이내에 1번이라도 실행된 Method와 사용이 빈번한 Method는 Native Code로 컴파일 되어 앱 실행과 동작 중 성능을 높이는데 활용된다.

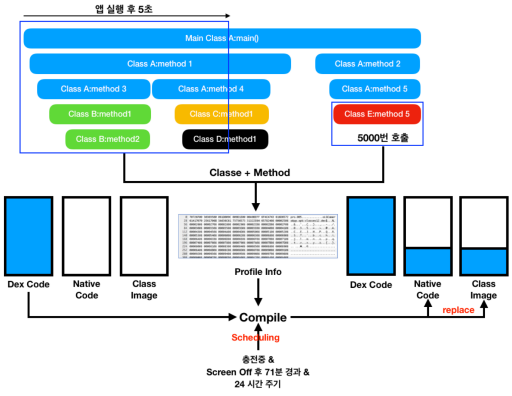


그림 4 : 최신 Android Profile 활용

3. Profile생성 및 앱 최적화

3.1 전체 구조

현재 최신 버전 Android에서는 앱 실행 속도가 개선되기 위해서는 앱 설치 이후 Optimization Loop(앱 사용 - Profile 기록 - 최적화 작업 반복)이 최소 1회 이상 동작하여야 한다. 그렇기 때문에 OOB(Out-of-box)또는 앱 설치 직후 실행속도는 최적화 전 상태로 동작하므로 느리다. 하지만 제안하는 방법은 그림3 의 아래와 같이 자동 생성한 Profile을 활용하여 앱 설치 시점부터 최적화된 상태의 실행코드를 생성하므로 앱 설치 직후부터 앱 실행 속도를 개선할 수 있다.

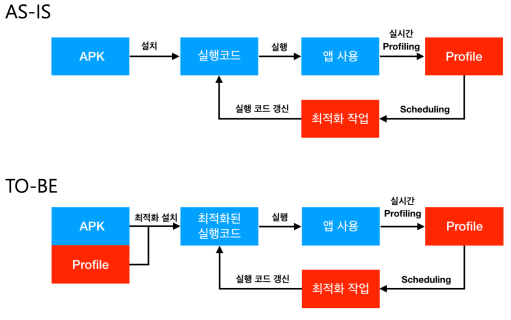


그림 5 : Profile 활용한 빠른 최적화

3.2 Profile의 특성 파악

Profile은 사용자별/앱별로 저장되며 동일 사용자/앱이라 하더라도 사용 패턴에 따라 다른 Profile이 기록되는

될 수 있다. 따라서 앱 실행속도를 개선할 수 있는 Profile을 미리 준비하고 활용하기 위해서 Profile의 특성을 파악이 필요하며 이를 위해 사용자/앱별로 최적화 횟수에 따른 앱실행속도의 차이가 있는지를 확인해 보았다.

3.2.1 사용자별 앱 실행 속도

사용자별 차이를 분석하기 위해 총 10명의 테스터에게 마켓 상위 10개 앱에 대해 최적화 전/후의 앱 실행 속도를 Nexus 5X(Android O, WIFI)에서 측정하여 비교한 결과 그림4와 같이 사용자 별 차이는 크지 않고 모든 사용자에게서 최적화 전 후로 앱 실행속도의 개선이 확인된다.

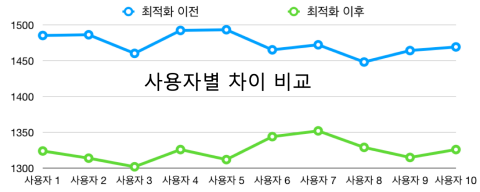


그림 6 : 사용자 별 최적화 전/후 비교

3.2.2 최적화 횟수 별 앱 실행 속도

최적화 횟수 별 차이를 분석하기 위해 총 2명의 테스터에게 마켓 상위 10개 앱에 대해 최적화 횟수 별 실행 속도를 Nexus 5X(Android O, WIFI)에서 비교해본 결과 그림5와 같이 1회 최적화 이후로 각각의 사용자 모두 앱 실행 속도가 개선되며 그 이후로는 큰 차이가 없음을 확인할 수 있다.

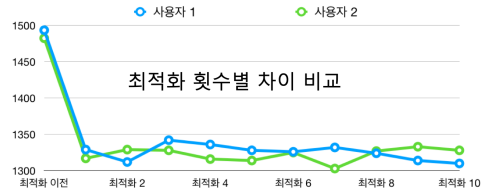


그림 7 : 최적화 누적에 따른 앱 실행속도 차이

3.2.3 Profile에 기록된 Class, Method의 개수

사용자/앱 별로 최적화 횟수 별로 Profile이 다름에도 불구하고 어떻게 최적화 이후 앱 실행속도가 개선되는지를 파악하기 위해 최적화에 포함되는 Class, Method를 살펴보았다. Pixel(Android 8.1)에서 YouTube앱(13.19.58)으로 다양한 최적화 수준에서 테스트 해본 결과 표1과 같이 앱 사용에 따라 각각 다른 수준의 Profile이 저장되지만 최적화 이후로는 큰 차이가 없음을 확인할 수 있다. 많은 시간 사용할수록 더 많은 Method 정보가 Profile에 기록되지만 앱 실행 속도는 더 개선되지 않는다. Class도 마찬가지이다. 그 이유로는 크게 2가지를 들 수 있다. 첫째 앱 실행 시 활용되는 Class와 Method는 앱 실행 시 첫 화면을 구성하는 UI에 연관되어 있으므로 이때 사용되는 Class와 실행되는 Method가 화면에 따라 정해져 있으며 실행 후 대기만 하는 최소 Profile로도 충분한 개선 효과를 거둘 수 있다. 둘째로는 Class와 Method정보를 더 누적하여 이를 최적화에 활용한다고 하더라도 이 결과 생성된 Class Image와 Native Code는 파일의 크기를 증가시켜 Locality가 감소함에 따라 I/O를 더 유발하고 앱 실행 속도 관점에서는 단점으로 작용할 수 있기 때문이다.

YouTube + Pixel 2 (13.19.58 버전)		최적화 이전 Interpreter 실행	최적화 이후 (실행후 대기)	최적화 후 (10분 사용)	최적화 후 (4시간 사용)	최적화 후 (10일 실사용)
Start up Methods	갯수	0 / 65489	4486	8458	9285	12093
	Reference 비교			98% (4418 개)	98% (4415 개)	93% (4193 개)
Hot Methods	갯수	0 / 65489	5648	8532	9287	13189
	Reference 비교			96% (5443 개)	96% (5477 개)	92% (5250 개)
Startup Classes	갯수	0 / 11187	2067	3934	4159	5181
	Reference 비교			100% (2067 개)	100% (2067 개)	99% (2048 개)
앱 진입 속도 (10회 평균)	ms	730	520	524	530	532

표 1 : Class, Method 개수 비교

3.3 Profile 자동 생성 방법

Profile을 획득하는 방법으로 이 연구에서는 좀 더 정확한 최소 Profile을 얻기 위해 Profiling Lunch를 직접 실행하는 방법을 적용하였다. 이를 위해서는 Launcher를 통해 실행 가능한 Activity를 직접 실행하여 하위 기능을 탐색할 수 있어야 한다. 이를 위해 Profile Generator라는 PC기반의 Tool을 작성하였으며 이 Tool은 그림6과 같이 ADB(Android Debug Bridge)를 통해 단말에 설치한 UIAutomator의 Agent와 Event를 주고 받으며 화면의 상태를 실시간으로 파악하여 Profile 기록을 위한 실행을 반복한다.

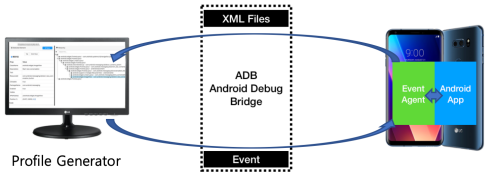


그림 6 : UIAutomator를 활용한 Profile Generator

Profile Generator의 동작은 그림7과 같이 표현할 수 있다. 대상 앱에 대해 화면 단위인 Activity를 실행하고 GUI 구성요소를 분석하여 1 Depth로 접근 가능한 메뉴를 탐색하고 탐색이 완료되면 Profile을 저장하고 동작을 완료한다.

Profile Generator의 기본 동작은 UI-Based Testing 기법과 유사한 부분이 많지만 실행의 목적에 있어서 Test Coverage를 높이는 것과 달리 앱 실행 속도를 높이기 위한 최소 Profile획득라는 점에서 큰

차이가 있다. 따라서 Profile Generator는 모든 GUI Tree를 탐색하지 않고 실행의 화면단위인 Activity를 실행하며 Activity내 모든 GUI구성요소를 재귀적으로 탐색하지 않고 1 Depth 실행으로 제한을 두는 형태로 동작한다. 또한 그림8의 Message앱과 같이 Activity내 1 Depth에 동일한 Component들이 반복 배치되는 경우 대표 Index만 실행하는 형태로 실행을 제한한다.

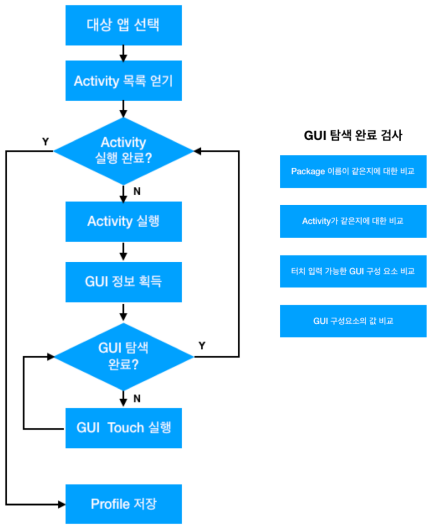


그림 7 : Profile Generator의 동작

그림 8 : Message앱 GUI Tree탐색의 최적화

동일한 GUI Component가 반복되고 탐색에 대한 최적화가 가능한 앱의 Activity는 List형태의 UI를 가지고 있는 특징이 있으며 대표적으로 Messge, Music, Youtube등이 있다. 표2의 테스트 결과를 보면 최적화 이후 대기하여 앱 실행 속도를 개선한 것과 List 첫 아이템만 탐색하는 것 List를 모두 탐색하는 것이 큰 차이가 존재하지 않고 동등한 개선을 보여주고 있고, 실제 Profile에 기록되어 있는 Method와 Class의 기록상으로도 List의 전체 탐색과 대표 Index만 탐색은 차이가 없음을 확인할 수 있다. 이와 같은 결과가 나오는 이유는 List에 포함되는 Item은 대부분 동일한 형태가 포함되어 있기 때문에 탐색을 하더라도 동일한 Class가 사용되고 동일한 Method가 실행될 가능성이 크기 때문이다.

SD845 +6G RAM Android Oreo MR1			최적화 이전 Interpreter 실행	최적화 이후 (실행후 대기)	List 첫 아이템만 탐색	List 모두 탐색
Message	Profile Class	갯수	0	389	450	450
	Profiled Method	갯수	0	1518	3113	3149
	앱 진입 속도 (10회 평균)	비율(%)	100	82	83	82
Music	Profile Class	갯수	0	494	508	543
	Profiled Method	갯수	0	2040	3171	3192
	앱 진입 속도 (10회 평균)	비율(%)	100	94	92	94
YouTube (13.25.56)	Profile Class	갯수	0	9544	9562	9653
	Profiled Method	갯수	0	20736	28473	29578
	앱 진입 속도 (10회 평균)	비율(%)	100	35	35	35

표 2 : GUI Tree탐색 최적화에 따른 테스트 결과

3.4 Profile 활용하여 최적화하는 방법

3.4.1 Build시점 최적화 방법

Build 시점 최적화는 그림9와 같이 기존에 존재하는 System Application Post-Build과정에 Profile Generator를 통해 획득한 Profile을 활용하도록 Build Script를 수정하면 가능하다. 기존의 Post-Build과정은 APK를 통해 검증된 형태의 DEX를 생성하는 것으로 끝나지만 앱 별로 다른 경로에 저장되어 있는 Profile을 활용하여 APK를 통한 최적화 수준을 Native Code와 Class Image를 생성하도록 할 수 있다. 최적화 결과 파일이 System Image에 포함되므로 사용자는 첫 실행부터 빠른 앱 실행을 경험할 수 있다.

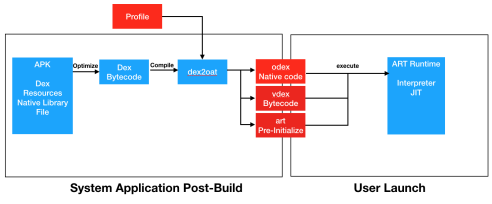


그림 9 : Build시점 최적화 방법

3.4.2 Install시점 최적화 방법

Install 시점 최적화는 그림10과 같이 2단계로 이뤄질 수 있다. APK외부에 존재하는 Profile을 직접 활용 가능한 Build시점 최적화와 달리 Install 시점 최적화를 위해서는 아래 그림10의 상단과 같이 APK내 Asset하위에 Profile을 포함하여 APK를 작성하는 과정이 필요하다. Install 시점에는 대상이 되는 APK이외에 다른 정보를 활용하지 않는 것이 기본이므로 다양한 방식의 Install을 지원하기 위해서는 APK내에 Profile을 포함하는 방식을 적용하였다. APK내 Profile이 존재하면 Install 시점에 최적화 작업을 미리 수행할 수 있고 앱 설치 후 첫 실행부터 빠른 앱 실행이 가능하다.

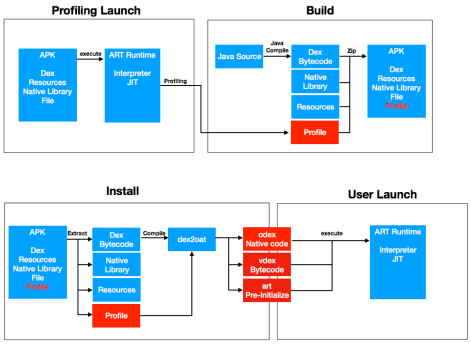


그림 10 : Install시점 최적화 방법

3.5 APK에 포함된 Profile을 활용하기 위한 구현

APK 내부의 Asset 하위에 Profile이 포함된 경우 이를 활용하여 앱 설치 시점에 최적화를 하기 위해서는 installd, PackageManager와 같은 Android Framework에 대한 수정이 필요하다. PackageManager는 Root권한의 installd를 활용하여 앱 설치를 진행하며, installd는 설치에 필요한 기능을 제공하고 dex2oat를 실행하여 Compile을 수행하므로 Compile전 APK 내부에 포함된 Profile을 추출하여 파일로 저장하는 과정인 그림11의 Prepare 단계가 필요하다. 또한 준비된 Profile을 Compile 과정에서 활용할 수 있도록 dex2oat의 입력으로 Profile을 제공하는 Android Framework 수정을 하였다.

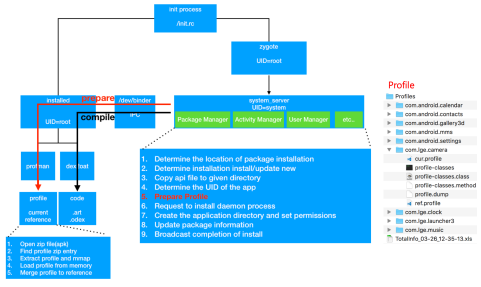


그림 11 : 준비단계를 통한 Profile 활용

4. 측정

4.1 앱 실행 속도 비교

제안한 방법인 Profile Generator를 활용하여 대상 앱들을 자동 실행과 자동 탐색한 결과로 얻은 Profile을 활용하여 Build/Install시점에 최적화를 수행한 테스트 결과가 표3과 같다. Contact, Music, Message, Camera, Gallery, Clock, Calendar, Settings 앱에 대해 총 100회 반복 실행한 평균이며 자동 생성한 Profile을 활용한 Build/Install시점의 최적화가 실제 사용자의 최적화 수준과 동등한 수준으로 최초 앱 실행부터 개선됨을 확인하였다.

MSM8996 +4G Android O	최적화 이전 (앱 실행 속도 비율 %)	Android 기본 최적화	Build 시점 최적화	Install시점 최적화
Contacts	100	94	95	95
Music	100	80	83	79
Message	100	96	96	95
Camera	100	95	95	95
Gallery	100	87	90	88
Clock	100	93	90	93
Calendar	100	94	96	94
Setting	100	91	91	91

그림 16 : 기본 최적화 및 제안 방법 적용 비교

5. 측정

최신 Android가 지니는 문제인 최적화 되기 이전까지 앱 실행 속도가 느린 문제를 개선하기 위해 본 연구에서는 앱 사용이력에 따라 어떤 Profile이 앱 실행속도를 높일 수 있는 것인지를 파악하였고 앱 실행속도를 높일 수 있는 Profile을 자동으로 생성하여 Build/Install시점에 활용함으로써 앱 실행속도를 최초 실행부터 개선할 수 있음을 확인하였다. 이를 통해 사용자는 OOB, 앱 설치/업데이트 이후 좀 더 빠른 앱 실행속도를 경험할 수 있을 것으로 기대한다.

향후에는 앱 실행 속도 관점 뿐만 아니라 앱 사용 중 다양한 사용 패턴에 따라 다를 수 있는 앱 동작 성능을 조기에 확보할 수 있는 방법에 대해 연구할 예정이다. 현재까지의 연구는 사용자 별로 큰 차이가 존재하지 않는 앱 실행시점의 최적화에 초점을 맞췄고 이 목적에 부합하는 Profile을 자동확보하여 활용하는 것에 목표를 두고 진행되었기 때문에 앱 동작 중 성능은 개선이 안되기 때문이다. 이를 위해 앱 동작 성능을 높일 수 있는 Profile의 특성을 파악하고 Profile Generator의 동작을 확장하는 방법에 대한 연구가 진행된다면 앱 실행속도 뿐만 아니라 앱 동작 성능을 조기에 확보할 수 있을 것이다.

참 고 문 헌

[1] 백영민, 홍광의, 배두환
: 효과적인 모델 기반 안드로이드 GUI 테스팅을 위한 GUI 상태 비교 기법 , 정보과학회논문지, 42(11), 2015, 1386-1396

채 향 석
2006년 아주대학교 정보컴퓨터공학과 학사
2006년 ~ 현재 LG전자
2019년 한국과학기술원 전산학부 석사
관심분야는 운영체제, 시스템 소프트웨어, 소프트웨어 신뢰성

백 종 문
1993년 조선대학교 컴퓨터과학 및 통계학과 학사
1996년 미국 University of Southern California Computer Science 석사
2000년 미국 University of Southern California Computer Science 박사
2001년 ~ 2005년 미국 모토로라 SSERL(Software and System Engineering Research Lab.) 수석연구원
2005년 ~ 2009년 2월 한국 정보통신대학교 공학부 부교수
2009년 3월 ~ 현재 한국과학기술원 전산학부 부교수
관심분야는 소프트웨어 신뢰성, 소프트웨어 메트릭스, 소프트웨어 비용 추정, 소프트웨어 동적 모델링, 소프트웨어 프로세스 개선, 소프트웨어 품질보증, 소프트웨어 식스 시그마

무기체계 소프트웨어 신뢰성 시험을 위한 효율적 시험 환경 구축 방안

(An Efficient Method of Test Environment Setup for Weapon System Software Reliability Test)

최 민 관 § 박 다 운 § 국 승 학 §
(Minkwan Choi) (Daun Bak) (Seunghak Kook)

요 약 최근 무기체계에서 소프트웨어가 차지하는 비중이 증가됨에 따라 소프트웨어의 품질이 매우 중요한 요소가 되고 있다. 무기체계 소프트웨어의 품질 향상을 위해 방위사업청은 무기체계 소프트웨어 개발 및 관리 매뉴얼에 소프트웨어 신뢰성을 제도화 하였고, 구체적인 방법 및 절차를 제시하고 있다. 매뉴얼에서 요구하는 소프트웨어 신뢰성 시험의 기준을 충족하기 위해서는 개발 전(全) 순기에 걸쳐 지속적인 시험을 통해 결함의 검출 및 수정이 필요하지만, 보안을 위한 망분리 환경, 시험 도구 확보를 위한 비용 문제로 인해 적정 수준의 시험 환경을 구축하는데 어려움이 따른다. 따라서 본 연구에서는 방위산업 분야에서 제한된 개발 환경과 한정된 자원을 활용해 효율적으로 소프트웨어 신뢰성 시험을 수행 할 수 있는 환경 구축 방안을 제시하고자 한다.

키워드 : 무기체계 소프트웨어, 소프트웨어 공학, 소프트웨어 테스팅, 소프트웨어 품질 관리

Abstract Recently, as the weight of software in the weapon system increases, the quality of the software becomes a very important factor. In order to improve the quality of the weapon system software, DAPA(Defense Acquisition Program Administration) has institutionalized software reliability in Weapon System Software Development and Management Manual. The manual presents specific methods and procedures to improve the weapon system software quality. In order to meet the required reliability test standards specified in the manual, it is necessary to continuously detect and correct defects throughout the entire development period. However, it is difficult to build proper reliability test environment due to the cost of software reliability tools, setting up secured and separated network environment, and etc. Therefore, in this study, we propose an efficient environment construction method for software reliability test of defense industry field in restricted development environment and limited resources.

Key words : Weapon System Software, Software Engineering, Software Testing, Software Quality Management

1. 서 론

여느 산업과 마찬가지로 방위산업 내에서도 소프트웨어의 중요성이 꾸준히 증가하고 있다. 무기체계가 수행하고자 하는 대부분의 기능들이 소프트웨어를 기반으로 구현되고 있으며, 연구개발 기간 중 상당 시간이 소프트웨어 개발에 할애되고 있다. 이처럼 소프트웨어가 차지하는 비중이 증가함에 따라, 연구개발의 근간이 되는 “무기체계 소프트웨어 개발 및 관리 매뉴얼” 역시 지속적으로 개선되고 있다. 해당 매뉴얼은 방위사업청에서 무기체계 소프트웨어 개발과 관리를 위해 2014년 최초 제정되었으며, 개발 절차와 단계별 활동들에 대해 정의하고 있다[1].

매뉴얼 개정의 핵심 내용 중의 하나는 소프트웨어 품질 강화를 위한 소프트웨어 신뢰성 시험 기준 강화이다. 소프트웨어 신뢰성 시험이란 소프트웨어가 동작 중에 유발할 수 있는 오동작 또는 결함을 식별하는 시험으로 정적(Static), 동적(Dynamic) 시험으로 구분되며 자동화 도구를 활용하여 수행한다.

원론적으로는 소프트웨어 신뢰성 시험을 구현 초기단계부터 지속적으로 수행하면서 결함을 식별하고 수정해야 한다. 하지만 한정된 예산 및 시간의 문제로 소프트웨어 통합 시점에 단발성으로 시험을 수행하고 검출된 결함을 수정하는 일이 빈번하다. 그 결과 사전에 식별되지 않은 많은 양의 결함이 사업 후반부에 검출되면서 개발자들이 이를 수정하는데 많은 어려움을 겪고 있다. 매뉴얼 개정 전의 시험 기준에서부터 발생했던 위와 같은 문제는 강화된 기준을 적용했을 때 더욱 심화 될 것으로 예상된다.

§ 비 회 원 : 국방과학연구소 연구원
{mkchoi, dwpark90}@add.re.kr
§ 비 회 원 : 국방과학연구소 선임연구원
kuk@add.re.kr
논문접수 : 2019년 3월 8일
심사완료 : 2019년 5월 8일

종전의 방식보다는 구현단계부터 시험을 지속적으로 수행하여 결함을 제거하는 것이 필요하다. 도구를 추가적으로 도입하여 개정된 매뉴얼에 대응할 수 있지만, 현실적으로 비용 문제가 있기 때문에 한정된 도구들을 적시에 활용할 수 있는 방안 모색이 필요하다. 따라서 본 연구에서는 한정된 시간과 도구로 보다 효율적으로 시험을 수행할 수 있는 소프트웨어 신뢰성 시험 환경 구축에 관한 연구를 수행하고자 한다.

본 논문의 구성은 다음과 같다. 2장에서는 소프트웨어 신뢰성 시험 현황 및 제한사항에 대해 서술한다. 3장에서는 국방 및 민수 분야의 소프트웨어 시험 환경에 관한 연구를 정리한다. 4장에서는 소프트웨어 신뢰성 시험을 위한 효율적 시험 환경 구축 방안을 제시한다. 마지막으로 5장에서는 결론을 기술한다.

2. 소프트웨어 신뢰성 시험 현황 및 제한사항

무기체계 소프트웨어 개발 및 관리 매뉴얼에서는 정적시험을 소프트웨어를 실행하지 않은 상태에서 잠재적인 결함을 검출하는 시험으로 코딩규칙, 취약점 점검, 소스코드 메트릭(Metric) 점검을 포함한다고 정의하고 있다. 동적시험은 소프트웨어를 실제 하드웨어(Target)에 탑재한 상태에서 시험절차서에 기술된 순서에 따라 소프트웨어 코드 실행률(Coverage)을 점검하는 것을 의미하며, 소프트웨어 품질 향상을 위해 소프트웨어 신뢰성 시험의 기준을 강화하는 방향으로 표 1과 같이 매뉴얼이 개정되었다.

첫 번째, 소프트웨어 신뢰성 시험의 대상이 되는 개발 언어가 확대되었다. 기존에는 C 또는 C++로 구현된 코드만을 시험의 대상으로 정의하였으나, 2016년 매뉴얼 개정을 통해 C#, Java도 포함이 되었다.

두 번째, 정적시험에서 코딩규칙의 기준이 수정 및 강화되었다. 2016년 개정된 매뉴얼에서는 방위사업청에서 선정한 코딩 규칙(65개)을 준수하던 수준을 넘어서 MISRA-C(143개 규칙), MISRA-C++(228개 규칙) 등의 국제 표준을 적용하는 것을 요구하고 있다[2]. 사업 특성에 맞게 규칙의 적용여부를 테일러링(Tailoring) 할 수 있으나, 과거에 비해 준수해야 하는 항목이 2배 이상 증가하였다. 더욱이 취약점 점검의 경우에는 CWE[1] 항목 중 체계에서 필요한 항목을 식별하여 적용할 수 있었던 과거와는 달리 CWE 항목을 모두 적용하는 것을 원칙으로 하고 있다[3]. 그리고 소프트웨어의 복잡도 감소, 유지보수 용이성 증대를 목표로 6개의 품질 측정 지표를 선정하여 소스코드를 제한 값 이하로 개발하는 것을 요구하고 있다. 예를 들어 “Number of Function Parameters”는 함수에 입력되는 파라미터의 개수를 의미하여 매뉴얼에서는 개별 함수의 파라미터 수를 8개 이하로 개발할 것을 요구하고 있다. 만일 제한 값 적용이 제한되는 경우에는 그 사유를 제시하고 사업관리 회의를 통해 결정할 수 있다.

마지막으로 동적시험의 경우에는 테스트 케이스(Test Case)를 활용해 구조적 실행률을 달성하는지 초점을 맞춘 과거에 비해 실제 요구사항 기반으로 소프트웨어를 실행하고 이에 대한 코드 실행률을 확인하는 방식으로 기준이 변경되었다.

무기체계 소프트웨어 개발 및 관리 매뉴얼		2014.02 제정	2016.07 개정
적용언어		C, C++	C, C++, C#, Java
정적 시험	코딩규칙	무기체계 SW 코딩 규칙 준수 (65개)	국제표준 (MISRA-C/C++ 등) 준수
	취약점 점검	CWE(Common Weakness Enumeration) 658/659 중 선별 적용	CWE-658/659/660 모두 준수
	소스코드 메트릭	없음	매뉴얼 상에 정의된 메트릭 종류 및 제한값 준수 <Number of Function Parameters 외 5종>
동적 시험	대상	위험 분석 후 일정 수준 이상 위험 소프트웨어 항목 설정	결함의 발생빈도, 영향성, 제어가능성 평가 국제표준(DO-178등) 적용 시 해당표준 준수
	목표값	위험 수준별 목표값 설정	결함의 빈도, 영향성, 제어가능성 평가표에 의한 코드 실행률 달성
	방법	구조기반시험 (Statement, Branch, MC/DC)	실제 하드웨어를 탑재한 상태에서 요구사항 기반으로 코드 실행률 점검

표 1 무기체계 소프트웨어 신뢰성 시험 범위의 변화

무기체계 소프트웨어의 신뢰성 확보라는 방향성을 가지고 시험 기준들이 개정되었다. 그 취지는 충분히 공감되나 강화된 기준은 다음과 같은 현실적인 제한사항을 지니고 있다. 우선 일반적으로 한정된 예산으로 인해 연구개발 참여기업들은 소프트웨어 신뢰성 시험을 위한 도구 라이선스를 소량 확보하여 다수의 개발자들이 함께 사용하고 있다. 만약 도구 라이선스가 모두 사용 중일 경우 개발자는 시험을 수행하지 못하고 도구 사용 가능여부를 수시로 확인해야 한다. 이와 같은 문제로 인해 개발자들은 개발 전(全) 순기에 걸쳐 소프트웨어 신뢰성 시험을 수행하지 않고, 소프트웨어 통합 시점에 시험을 단발성으로 수행하고 이때 발견된 오류를 수정하는 방식을 취하고 있다. 그러다 보니 개발 마무리 시점에 많은 양의 오류가 검출되어 이를 수정하는데 큰 어려움을 겪고 있다. 따라서 적은 수량의 도구로 개발자들이 보다 손쉽게 소프트웨어 신뢰성 확보 활동을 수행할 수 있는 시험 환

1) Common Weakness Enumeration : 미국 국토안보부 산하의 MITRE에서 제공하는 취약점 항목(CWE-658: C, CWE-659: C++, CWE-660: JAVA)

경 구축이 필요하다.

3. 관련 연구

무기체계 소프트웨어 연구개발에서 소프트웨어 신뢰성 시험과 관련된 여러 문제들을 해소하기 위해 국방 분야에 참여하고 있는 기업들은 많은 노력들을 해왔다. 특히 소프트웨어 신뢰성 시험을 자동화하여 투입되는 시간과 인력을 최소화하고자 하였다. 예를 들어 한 연구에서는 동적시험에 활용되는 Vector사의 VectorCast를 해당 도구에서 제공하는 인터페이스를 활용하여 빌드(Build)와 시험을 자동으로 수행할 수 있는 환경을 구축하였다[4]. 변경된 소스코드에 대한 빌드 및 동적시험 수행과 라이선스가 가용하지 않을 경우 자동으로 재시도를 하는 기능을 그림 1과 같은 배치 처리 방식으로 구현함으로써 개발자에게 편의를 제공하는 시스템이다. 구축 후에는 개발한 환경이 개발자의 작업시간을 얼마나 감소시켰는지 조사하였다. 그 결과 자동화 시스템을 활용하여 시험하는 경우 소요시간이 기존 104시간에서 72시간으로 약 30% 감소했다는 것을 확인하였다. 특히 시스템에서 자동으로 도구 라이선스를 확보하고 동적 시험을 수행함으로써 그 시간 동안에 개발자들이 다른 업무를 수행할 수 있었던 것이 이와 같은 큰 시간 단축을 불러 왔다.

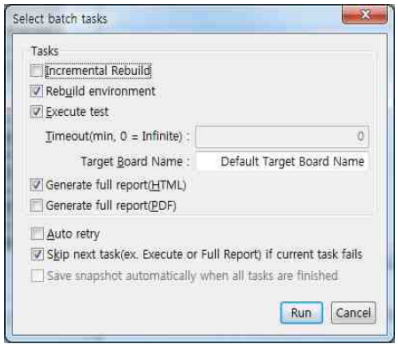


그림 19. VectorCast 자동화 시스템

동적시험 자동화시스템과 관련된 연구 뿐만 아니라, 시험의 소요시간에 영향을 주는 요인들을 추출하고 이를 향상할 수 있는 방안에 대한 연구도 진행이 되었다[5]. 해당 연구에서는 시험 소요시간에 영향을 미치는 요인으로 개발자의 도구 숙련도, 보조 도구 사용 여부, 라이선스 수, 시험 횟수 등을 선정하였다. 그리고 영향을 미치는 인자들의 향상을 위해 그림 2과 같은 절차를 통해 정적시험 자동화 시스템을 설계 및 평가하였다. 또한 자동화 시스템 구축을 통해 성능 향상을 기대하기 힘든 항목에 대해서는 개별적으로 개선 요소를 식별하였다. 예를 들어 개발 PC 성능이 시험의 시간에 영향을 미칠 수 있다고 판단하여 개발 PC 교체 주기 등을 포함한 하드웨어 성능 개선 방안을 제시하였다.

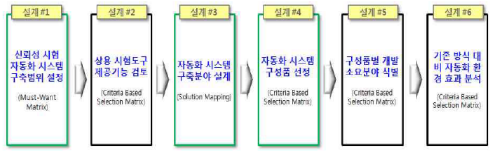


그림 20. 자동화시스템 구축 절차

국방 분야에서 소프트웨어 신뢰성 시험과 관련된 비효율성을 제거하기 위해 시험 환경 구축에 대한 연구가 활발한 만큼 민수 분야에서도 소프트웨어 시험 환경에 대한 연구가 활발히 수행되고 있다. 그 중 지속적 통합 기반의 소프트웨어 시험 환경 구축에 대한 연구가 수행되고 있다 [6-7]. 특히 정보통신진흥원에서는 CI 기반의 품질관리 시스템을 구축하여 국가 연구개발 사업에 참여하고 있는 기업들의 품질 관리를 지원하고 있다[8].

지속적인 통합(CI, Continuous Integration)은 소프트웨어 공학 관점에서 지속적인 품질 제어(Quality Control)를 적용하는 프로세스를 의미한다[9]. 프로젝트에 참여하는 구성원들이 개발한 산출물을 주기적으로 통합한다. 통합이 수행될 때마다 시험을 포함한 자동화된 빌드 절차에 의해 통합된 내용은 자동 검증되며, 이로 인해 발생하는 문제를 조기에 식별할 수 있다.

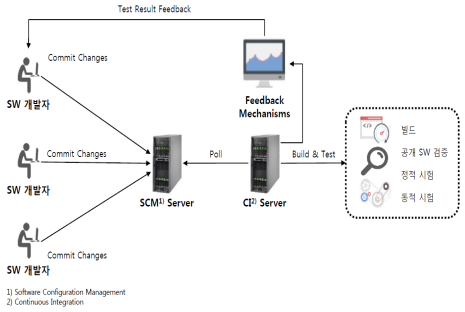


그림 21 CI 기반의 소프트웨어 시험 환경의 예시

일반적인 CI 기반의 시험 환경은 그림 3과 같다. 개발자는 개발 PC에서 소스코드를 형상관리 저장소(SCM, Software Configuration Management)에 최신화 한다. 이후 CI는 소스코드의 변경유무 또는 특정 시점에 소스코드를 자동으로 빌드하고 서버에서 제공하는 정적, 동적 시험 및 공개 소프트웨어 검증 등의 시험을 수행한다. 이후 그 결과를 개발자에게 전송하는 구조를 가진다.

CI 기반의 시험 환경을 구축한다면 소스코드 변경 시에 빌드와 요구되는 시험을 자동화함으로써 투입되는 리소스를 감소하면서도 품질 향상을 기대할 수 있다. 그리고 단일 소스코드 저장소를 통해 모든 이해관계자(Stakeholder)가 소스코드 및 시험 결과에 대한 모니터링과 추적성 관리가 가능하다는 장점이 있다. 그림 4는 CI

를 이용하여 소프트웨어 신뢰성 시험 환경 구축 전과 후를 비교한 것이다. 2장에서 언급했듯이 환경 구축 전에는 개발부서에서 SW 신뢰성시험 도구를 보유하지 못하거나 소량의 도구만을 보유하고 있기 때문에 도구를 선점한 개발자가 시험을 모두 수행할 때까지 다른 개발자는 시험을 적시에 수행하지 못하는 문제가 있었다. 환경 구축 후에는 각각의 개발부서에서 보유한 도구들을 서버에 통합 구축하여 소프트웨어 신뢰성 시험이 필요한 개발자에게 우선순위에 따라 서비스를 제공함으로써 기존의 문제를 극복할 수 있다. 따라서 본 연구에서는 관련 연구들을 기반으로 국방 분야의 특성에 맞는 CI 기반의 소프트웨어 신뢰성 시험 환경 구축이 필요하다.

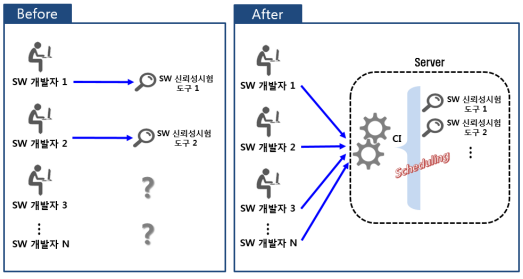


그림 22 SW 신뢰성시험 환경 구축 전/후 비교

4. 소프트웨어 신뢰성 시험 환경 구축 방안

시험환경 구축에 앞서 무기체계 소프트웨어 개발 현장에 존재하는 제약사항에 대한 이해가 필요하다. 무기체계 내장형(Embedded) 소프트웨어라 함은 각종 무기·전력지원체계에 내장되어 해당 장비의 임무에 전용으로 제공되는 소프트웨어를 의미한다. 표 4의 무기체계 소프트웨어 개발환경 예시와 같이 개발 품목이 매우 다양하다. 그리고 일반적으로 사업 규모가 크기 때문에 다양한 이해관계자가 사업에 참여하고 있다. 기존에 진행된 연구들에서는 하나의 회사 기준으로 소프트웨어 신뢰성 시험 환경 구축에 대한 연구를 진행하여, 개발 도구나 운영체제 등이 비교적 표준화되어 있는 상태였다. 하지만 본 연구에서는 전체 무기체계 개발 프로세스를 지원하는 것을 목표로 하고 있기 때문에 다양한 탑재환경, 운영체제, 개발도구를 비롯한 개발환경에 대한 고려가 필요하다.

다음으로 무기체계 소프트웨어 개발은 인터넷망과 분리된 내부망(Intranet) 환경에서 수행되고 있다. 대부분의 소프트웨어가 내부망 환경에서 개발과 시험이 진행되지만, 일부 보안에 민감한 소프트웨어의 경우 단독으로 구성된 폐쇄망 환경에서 개발과 시험이 이루어지기도 한다. 따라서 본 연구에서는 내부망과 폐쇄망을 모두 지원할 수 있는 소프트웨어 신뢰성 시험 환경을 구축하고자 한다.

구 분	예 시
운영체제	Windows, Linux, VxWorks, pSOS, FreeRTOS 등
IDE/ 컴파일러	WindRiver Workbench, Code Composer Studio, Visual Studio, Atmel Studio, GCC 등
기반 SW	BSP(Board Support Package), 칩 제조사 제공 SW
개발 언어	C, C++, C#, Java, Python 등
탑재 환경	 일반컴퓨터(PC, 워크스테이션)
	 보드형 컴퓨터(SBC)
	 마이크로 컨트롤러(MCU)
	 디지털신호처리 프로세서(DSP)

표 7 무기체계 소프트웨어 개발환경 예시

4.1. SW 신뢰성시험 내부망 환경

내부망에서 개발되는 소프트웨어의 신뢰성 시험을 위한 환경은 그림 5와 같다. 서버 내에 가상머신(Virtual Machine)을 활용하여 각각의 개발환경을 구축한다. 가상머신이란 컴퓨팅 환경을 소프트웨어로 구현하여 그 위에서 운영체제가 작동하도록 하는 기술이다. 이는 하나의 서버에서 여러 개의 운영체제가 작동되는 것을 가능케 한다. 서버에는 각각의 개발환경을 구현한 가상머신, CI, 형상관리 저장소, SW 신뢰성시험 도구, SW 신뢰성시험 관리시스템 등이 탑재가 된다. 형상관리 저장소는 개발한 소스코드를 저장 및 관리할 수 있는 저장소이다. 개발자는 최종 소스코드 뿐만 아니라 개발 과정 동안 수정된 이력 관리를 위해 지속적인 형상관리를 해야 한다. SW 신뢰성시험 도구는 정적 및 동적 시험을 지원하는 도구를 의미한다. 마지막으로 SW 신뢰성시험 관리시스템은 개발환경에 접근할 수 있는 인터페이스와 개발자에게 소스코드 빌드와 SW 신뢰성시험 결과를 전송하는 역할을 한다.

내부망 환경의 소프트웨어 신뢰성 시험 절차는 다음과 같다. 우선 개발자(사용자)는 SW 신뢰성시험 관리시스템을 통해 개발환경에 접근한다. 그리고 개발·시험환경 및

소스코드가 있는 가상머신에서 개발·수정한 소스코드를 형상관리 저장소에 최신화 한다. 서버의 CI는 주기적으로 형상관리 저장소의 변화를 감지하고, 변화가 있는 경우 사용자가 설정한 SW 신뢰성시험 도구를 이용하여 시험을 수행한다. 마지막으로 시험 종료 후 시험에 대한 결과를 사용자에게 전송 및 SW 신뢰성시험 관리시스템에 저장 및 전시한다.

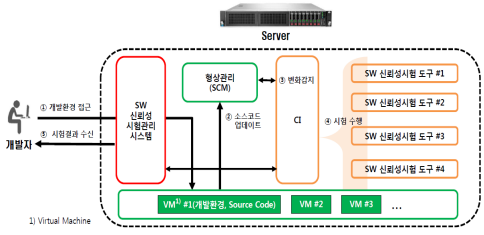


그림 23 SW 신뢰성시험 내부망 환경

이와 같이 CI를 활용해 개발자의 시험 도구 라이선스 확보를 위한 노력과 시험 수행을 자동화함으로써, 환경 구축 이전보다 개발자가 보다 수월하게 소프트웨어 신뢰성 시험이 가능할 것으로 기대된다. 하지만 내부망 환경에서는 사용자 집중에 따라 서버 성능 문제가 발생할 수 있으며, 확보된 도구 라이선스 수에 영향을 받을 수 있다는 점은 고려해야 한다.

4.2. SW 신뢰성시험 폐쇄망 환경

폐쇄망 환경에서 개발되는 소프트웨어의 신뢰성 시험 환경은 그림 6와 같다. 별도의 망 연결 없이 개발 PC와 도구가 설치된 서버를 직접 물리적으로 연결 한다. 서버에는 CI와 SW 신뢰성시험 도구가 설치되어 있으며, CI는 SW 신뢰성시험 도구의 설정 정보와 도구 스케줄링과 관련한 역할을 수행한다. 우선순위 알고리즘에 따라 자동으로 SW 신뢰성시험을 수행하고 그 결과를 CI를 통해 전시한다.

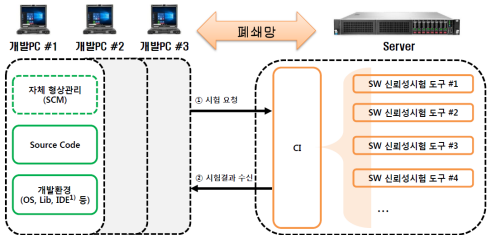


그림 24 SW 신뢰성시험 폐쇄망 환경

위와 같은 시험 환경은 보안에 민감한 소프트웨어를 개발할 때 효과적일 수 있다. 또한, 별도의 형상관리 저장소 및 개발환경 구축 없이 CI를 SW 신뢰성시험 도구의 스케줄링을 활용하기 때문에 개발환경 구축에 투입되는 시간과 비용을 절약할 수 있다는 장점이 있다. 하지만, 시험 장소가 서버가 설치된 위치에 한정된다는 단점이 존재하며, SW 신뢰성시험 관리시스템 및 형상관리 저장소 부재로 개발·시험환경, 소스코드 및 시험결과를 개발자가 자체적으로 수행하기 때문에 관리가 어려울 수 있다.

4.3. SW 신뢰성시험 통합 환경

그림 7의 SW 신뢰성시험 통합 환경은 4.1.과 4.2.에서 정리한 내부망, 폐쇄망 환경을 모두 지원하는 환경이다. 내부망 환경으로 시험 환경을 제공하는 것은 많은 장점이 있지만 보안 등의 문제로 폐쇄망 환경에 대한 수요도 적지 않다. 따라서 그림 7의 SW 신뢰성시험 통합 환경과 같이 내부망과 폐쇄망의 환경을 모두 지원하는 시험 환경 구축이 필요하다.

개발자는 자신의 필요에 맞게 시험 환경을 선택할 수 있다. 그리고 CI에는 내부망과 폐쇄망의 사용자가 SW 신뢰성시험 도구를 사용하기 위해 필요한 스케줄링 정책 정보(시험 일시, 시험 우선순위, 시험 도구간의 우선순위 등)가 저장되어 있어, 그 조건에 따라 도구 사용에 대한 우선순위를 부여하여 소프트웨어 신뢰성 시험을 자동으로 수행한다.

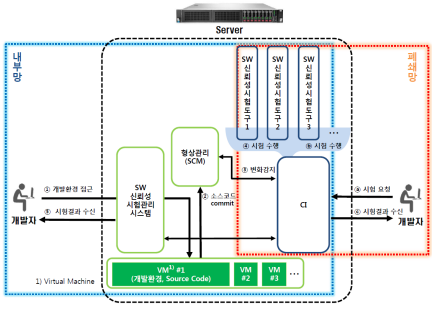


그림 25 SW 신뢰성시험 통합 환경

이와 같은 소프트웨어 신뢰성 시험 환경 구축을 통해 시험 도구 확보에 대한 문제를 해결 할 수 있을 것이라 기대한다. 환경 구축 전에는 개발부서에서 SW 신뢰성시험 도구를 확보하지 못했거나, 소량의 도구 라이선스를 보유하고 있기 때문에 도구를 선점한 개발자가 시험을 모두 수행할 때까지 다른 개발자는 시험을 적시에 수행하지 못하는 문제가 있었다. 하지만 환경 구축 후에는 각각의 개발부서에서 보유한 도구들을 서버에 통합 구축하여 시험이 필요한 개발자에게 우선순위에 따라 서비스를 제공함으로써 기존의 문제를 극복할 수 있다. 또한, CI에서 소스코드의 변경 유무를 지속적으로 확인하여, 빌드 및 시험을 수행하여 결과를 개발자에게 송신함으로써 이해관계자간 소스코드 및 시험 결과에 대한 모니터링과

추적성 관리가 가능할 것으로 기대된다.

하나의 서버에서 폐쇄망과 내부망 사용자들에게 동시에 서비스를 제공하고 내부망 서비스를 구축함에 따라, 발생할 수 있는 보안 이슈를 대응하기 위해 보안 취약점과 관련된 추가적인 시험과 사용자 별 접근 권한에 대한 엄격한 관리가 필요할 것으로 판단된다.

5. 결 론

본 논문에서는 무기체계 개발에서 수행하는 소프트웨어 신뢰성 시험을 효율적으로 수행 할 수 있는 시험 환경 구축 방안을 제안하였다. 망분리 환경에서 소프트웨어 신뢰성 시험 도구의 라이선스 제한을 최소화 하여 시험을 수행할 수 있는 환경을 구축함으로써 개발자는 개발한 소프트웨어에 대한 지속적인 시험이 가능하다. 따라서 개발 이외의 부분에서 발생하는 업무량을 감소시킬 수 있기 때문에 무기체계 소프트웨어의 품질과 생산성 향상을 기대할 수 있다.

참 고 문 헌

[1] 방위사업청, "방위사업청 매뉴얼 제2018-7호 무기체계 소프트웨어 개발 및 관리 매뉴얼", 2018.

[2] MISRA-C/C++ [Online]. Available: https://www.misra.org.uk

[3] CWE 659/659/660 [Online]. Available: https://cwe.mitre.org

[4] 차상철, 김정열, "무기체계 SW 동적시험 회귀시험 자동화 프로그램 개발", 한국항공우주공학회지, pp. 892-897, 2017.10.

[5] 김형권, "SW 신뢰성시험 자동화를 통한 시험기간 단축 및 인적오류 감소 방안", 한국컴퓨터정보학회논문지, pp. 45-51, 2015.10.

[6] 전종훈, "지속적인 통합 서버를 이용한 소프트웨어 품질 관리", 한국컴퓨터종합학술대회, pp.619-621, 2016.6.

[7] 마진 외 "계산과학공학 플랫폼 소프트웨어 품질 향상을 위한 테스트 시스템 구축 방안", 한국통신학회, pp.1513-1514, 2018.6.

[8] 정보통신산업진흥원, "SW개발 품질관리 매뉴얼", 2013.

[9] CI [Online]. Available: https://ko.wikipedia.org/w/index.php?title=지속적_통합&oldid=20682038

최 민 관

2011년 용인대학교 컴퓨터정보학부 졸업(학사)
2017년 한양대학교 컴퓨터소프트웨어학과 졸업(석사)
2017년~현재 국방과학연구소 연구원.
관심분야는 소프트웨어공학, 소프트웨어 테스팅

박 다 운

2013년 성균관대학교 시스템경영공학과 (학사)
2015년 KAIST 경영공학과 (석사)
2015년~현재 국방과학연구소 연구원.
관심분야는 소프트웨어공학, 소프트웨어 테스팅

국 승 학

2004년 충남대학교 컴퓨터과학과(학사)
2006년 충남대학교 컴퓨터공학과(석사)
2012년 충남대학교 컴퓨터공학과(박사)
2012년~2016년 국방과학연구소 연구원
2016년~현재 국방과학연구소 선임연구원.
관심분야는 소프트웨어공학, 소프트웨어 테스팅

행정안전부 소프트웨어 보안 취약점 진단기준과 Java 웹 어플리케이션 대상 오픈소스 보안 결함 검출기 검출대상의 총체적 비교†

(Systematic and Comprehensive Comparisons of the MOIS Security Vulnerability Inspection Criteria and Open-Source Security Bug Detectors for Java Web Applications)

이 재 훈 ‡ 최 한 솔 § 홍 신 ¶
(Jaehun Lee) (Hansol Choe) (Shin Hong)

요 약 경쟁적이며 급진적으로 오늘날 소프트웨어 개발 산업 현장에 시큐어 코딩 방법론을 효과적으로 적용하기 위해서는 보안 취약점 결함을 자동으로 검출하는 결함 검출기의 효과적이고 효율적인 적용이 필수적이다. 본 논문은 Java 웹 어플리케이션을 대상으로 하여 우리 행정안전부가 정의한 42개의 보안 취약점 진단 기준과 총 323개의 오픈소스 보안 취약점 결함 검출기의 검출 대상 결함 패턴을 비교하여, 동일한 결함 패턴을 대상으로 하는 것이 무엇인지를 명시화한 결과를 소개한다. 조사 결과를 바탕으로, 본 논문에서는 현재 행정안전부 보안 취약점 진단 기준 방법론의 한계점, 오픈소스 보안 결함 검출 프레임워크 간의 결함 검출 범위의 비교, 그리고 시큐어 코딩 가이드라인에 기반 한 개발 보안 방법론의 발전 과제를 논의한다.

키워드 : 보안취약점, 시큐어 코딩, 정적 분석, 오픈소스 소프트웨어

Abstract To enhance effective and efficient applications of automated security vulnerability checkers in highly competitive and fast-evolving IT industry, this paper studies a comprehensive set of security bug checkers in open-source static analysis frameworks and how they can be utilized for source code inspections according to the security vulnerability inspection guidelines by MOIS. This paper clarifies the relationship between all 42 inspection criteria in the MOIS guideline and total 323 security bug checkers in 4 popular open-source static analysis frameworks for Java web applications. Based on the result, this paper also discuss the current challenges and issues in the MOIS guideline, the comparison among the four security bug checker frameworks, and also the ideas to improve the security inspection methodologies using the MOIS guideline and open-source static security bug checkers.

Keywords : Security vulnerability, Secure coding, Static analyzer, Open-source software

1. 서 론

오늘날 정부, 금융 기관 등 주요 사회 기반의 정보 시스템에서 모바일/웹 어플리케이션 형태의 사용자 인터페이스를 제공함에 따라 모바일/웹 어플리케이션 개발 중 발생하는 사소한 소프트웨어 결함이 시스템 전체의 치명적인 보안 문제로 이어지는 사례가 늘고 있다[1]. 이와 같은 모바일/웹 어플리케이션 개발에서의 보안 취약점 발생에 대한 효과적인 대응하기 위해 소프트웨어 개발 및 검토 과정에서 프로그래밍 요소의 안전한 사용 규칙을 규제/강제하는 시큐어 코딩 기반 개발보안 방법론이 실제 소프트웨어 산업 현장에서 적용되고 있는 추세이다. 우리 정부의 행정안전부는 2013년 소프트웨어 보안 취약점 진단 기준[2]을 발표하고 이에 기반 한 보안 전문가를 양성함으로써, 전자정부 소프트웨어의 코드를 보안 전문가가 수검토하여 잠제적인 보안취약점을 진단하는 방법론을 정립해 오고 있다.

시큐어 코딩 기반 개발보안 방법론을 개발 주기가 빠른 일반적인 소프트웨어 산업 현장에 효과적으로 적용하기 위해서는 수기적 검토를 대신, 자동화된 정

† 본 연구는 과학기술정보통신부의 소프트웨어중심대학 지원사업 (2017-0-00130)과 한국연구재단을 통한 신진연구자지원과제 (NRF-2017R1C1B1008159)의 지원을 받음.
‡ 학생회원 : 포항공과대학교 컴퓨터공학과 (본 논문을 작성할 당시 한동대학교 전산전자공학부 소속)
GSoJC234@gmail.com
§ 학생회원 : 한동대학교 전산전자공학부
hansolchoe@handong.edu
¶ 종신회원 : 한동대학교 전산전자공학부 (교신저자)
hongshin@handong.edu
논문접수 : 2019년 3월 9일
심사완료 : 2019년 5월 8일

적분석(static analysis) 기반 보안취약점 결함 검출기(bug checker) 사용이 필수적이다. 하지만 보안취약점 검출을 자동적이고 체계적으로 제공하는 상용 도구의 경우, 사용에 많은 비용이 소요되어 일반적인 모바일/웹 어플리케이션 개발상황에서 활용이 비교적 제한적인 실정이다.

상용도구가 가지는 접근성을 해결하는 방법으로 오픈소스 보안취약점 자동검출 도구를 활용한 방법론을 생각해 볼 수 있다. FindBugs[3], PMD[4]와 같은 오픈소스 보안취약점 자동검출 도구를 우리 정부가 정립한 보안 취약점 진단 가이드라인을 중심으로 한 시큐어 코딩 방법론에 효과적으로 사용하기 위해서는 다음 두 가지 기술적 난점이 해결되어야 한다:

(1) 현재 우리 정부의 소프트웨어 보안 취약점 진단이 보안 취약점으로 진단하는 대상과 오픈소스 결함 검출기 간의 검출 대상 연관 관계가 명시적으로 파악되어있지 않음

(2) 검출 대상이 유사한 다수의 오픈소스 결함검출기의 검출 대상 사이의 연관 관계가 명시적으로 파악되어 있지 않음

이러한 문제로 인해, 현재 현업에서는 특정한 종류의 보안 취약점 진단을 위해 어떠한 오픈소스 결함 검출기를 사용해야 하는지 판단하기 어렵다. 또한 복수 개의 오픈소스 결함 검출기를 사용할 경우, 검출대상이 달라 검출 범위를 효과를 증대하는지 혹은 검출대상이 중복되어 실제적인 효과가 없는 지를 판별하기 어려운 실정이다.

본 논문은, 오늘날 웹 어플리케이션 개발에 널리 사용되는 Java 프로그램에 대하여 우리 정부가 정의한 42개 보안 취약점 진단 기준이 보안 취약점으로 정의하는 결함과 오픈소스로 개발된 총 323개의 결함 검출기가 결함으로 검출하는 대상을 간의 대응 관계를 명시적으로 조사한 연구 결과를 소개한다. 또한, 조사연구를 바탕으로 현재 개발된 보안 취약점 진단 기준 기반 방법론의 약점을 검토하고, 이를 보완하기 위한 방법을 논의한다. 본 연구는 궁극적으로 소프트웨어 산업 현장에서 자동화 도구를 활용한 개발보안 증진을 기대한다.

본 논문의 구성은 다음과 같다. 2장에서 관련 연구를 개관한 후, 3장에서 본 연구의 조사 대상인 우리 정부의 보안 취약점 진단 기준과 오픈소스 결함 검출기를 소개한 후, 상호 간의 대응 관계를 밝힌 결과를 소개한다. 4장에서는, 3장에서 소개한 조사연구 결과를 기반으로 개발보안 방법론에 오픈소스 도구의 활용 방안과 개선을 논의한 후, 5장에서는 본 연구의 결론을 소개함으로써 논문을 마무리한다.

2. 관련 연구

2.1. 보안 취약점 진단 기준 정립과 효과적인 적용 방법에 대한 연구

국내 IT산업 환경에서 전자정부 시스템 등 높은 신뢰성이 요구되는 소프트웨어 개발과 검수에서 활용할 수 있는 보안 취약점 진단 기준을 수립하기 위해, 시큐어 코딩 가이드라인을 기반으로 한 효과적인 모바일/웹 어플리케이션 개발보안 방법론에 대한 연구가 진행되었다. 2012년에는 기존에 제시되어 온 보안 취약점 목록을 바탕으로 국내환경에 적합한 소프트웨어 보안 취약점 최소 기준을 제시하는 연구가 발표되었다[5]. 또한 2015년에는 우리 정부의 소프트웨어 개발 보안을 위한 보안 취약점 표준 목록을 제안한 연구[6]가 발표되었다.

또한, 시큐어 코딩에 기반한 보안 취약점 진단 방법을 우리 IT 산업 현장에 효과적으로 적용하여 소프트웨어 보안 문제에 대응하기 위한 정책적/제도적 방안이 연구 되어 왔다. 보안 취약점의 심각성을 객관적으로 평가할 수 있는 정량 평가기준에 대한 연구[11], 국내 IT산업 환경에 적합한 진단도구 요구사항과 진단도구의 신뢰성을 보증할 수 있는 평가 방법론에 관한 연구[12], 미국의 개발보안 방법론 사례 연구[13], 시큐어 코딩중심으로 신뢰할 수 있는 안정적인 정보보호와 정보보안활동을 강화하는 방법 제안[14]과 보안 취약점 점검도구 확산과정에서 발생하는 문제를 정부에서 추진한 표준프레임워크 사업과 비교 점검하여 개선안을 제시[15]하는 등의 연구가 발표되었다.

2.2. 자동 결함검출기의 활용에 대한 연구

정확하고 효율적인 보안 취약점 진단을 위한 자동화 도구로 오픈소스 보안 결함 검출기를 활용하는 방안이 연구되어 왔다. 보안 취약점을 검출 대상으로 하는 오픈소스 결함 검출기의 동향에 대한 조사[7]로 한국인터넷진흥원은 오픈소스 도구인 FindBugs, FindSecurityBugs, PMD를 활용하여 행정자치부 47개 보안 취약점 진단 기준에 활용하는 방안을 소개하고, 진단방법과 89개 결함검출기 간의 연간 관계를 조사한 결과를 소개한 자료를 2016년에 발표했다[8]. 방지호 등(2013)의 연구결과에서는 보안 취약점 진단 기준과 연관된 결함검출기의 검출 성능을 평가하기 위해 테스트 입력으로 사용할 검증 대상 코드 집합을 개발한 결과를 발표하였다[9][10].

본 논문은, 앞서 소개한 관련 연구에 이어서 다음의 부분에서 새로운 결과를 보고 한다:

- 본 논문에서는 보안 취약점과 관련된 총 323개의 오픈소스 결함검출기의 검출 대상을 총체적이며, 구체적으로 조사/비교한 결과를 수행하였다(3.1절, 3.2절)
- 본 논문에서는 보안 취약점 진단방법과 결함검출

기 간의 검출 대상의 대응 관계는 물론, 결함검출기 간의 연관 관계도 조사하였다. 이 연구 결과를 소프트웨어 산업현장에서 오픈소스 결함검출기를 활용한 개발보안 방법론 적용에 활용할 수 있도록, 조사 결과 데이터를 체계적으로 조직화 하여 공개하였다(3.3절)

- 조사연구 결과를 바탕으로, 현재의 방법론과 도구가 가지는 한계점을 파악한 결과를 소개하며, 이를 바탕으로 구체적인 개선 방향을 제시하였다(4장)

3. 보안 취약점 진단 기준과 오픈소스 결함검출기의 연관 관계 조사

3.1. 조사의 목표와 대상

본 연구는 행정안전부가 발표한 보안 취약점 진단 기준[2]이 제시하는 총 47개 Java 프로그램 보안 취약점 진단 방법과 2018년 6월 현재 4종의 오픈소스 프로젝트를 통해 공개된 총 1310개의 보안 취약점 대상 결함검출기에 대하여, 다음 3가지 연관 관계를 명료화 하는 것을 목표로 하였다:

1. 특정 보안 취약점 진단 방법(이후 진단 방법)이 검출하고자 하는 보안 취약점 결함을 특정한 오픈소스 결함 검출기를 통해 검출할 수 있는가?
2. 특정 보안 취약점 결함검출기가 검출하는 대상은 어떠한 보안 취약점 진단 방법의 진단 대상에 속하는가?
3. 두 개의 보안 취약점 결함 검출기가 검출하고자 하는 대상이 같은가?

본 연구는 2013년 행정안전부(당시 안전행정부)가 발표한 보안 취약점 진단 기준[2]의 7개 카테고리의 총 47개의 보안 취약점 진단 방법 중 (1) Java 프로그램과 연관성이 없는 1개의 진단 방법과 (2) 보안 취약점에 한정되기 보다는 일반적인 결함 유형(예: Null Pointer Dereference)에 대해 정의한 4개의 진단 방법을 제외한 42개 진단 방법을 조사 대상으로 선정하였다.

또한, 본 연구는 Java 프로그램을 대상으로 한 오픈소스 프로젝트 형태의 보안 취약점 결함검출기를 총체적으로 조사하기 위하여, OWASP(오픈소스 웹 애플리케이션 보안 프로젝트)에 소개 된 총 41개의 결함검출기 프레임워크의 오픈소스 프로젝트 중 (1) Java언어를 지원하고, (2) 오픈소스 프로젝트로 개발되고 무료로 공개되었으며, (3) 결함검출기의 검출 대상에 대한 설명이 구체적으로 갖추어져 져있는 FindBugs[2]를 포함하는 FindSecurityBugs[15] (이후 FB+FSB), PMD [4], LAPSE+ [16], SonarQube [17]를 일차적 조사대상 프레임워크로 선정한 후, 선정된 4개의 프레임워크 내에 있는 총 1310개의 결함검출기의 결함검출 기능을 검토하여 보안 취약점과 연관성이 높은 총 323개의 결함검출기를 최종적 조사 대상으로 선정하였다.

표1(Table 10). Studied Bug Checkers

Platform (Version, Release Date)	#. Target Checkers (#. Total Checkers)
FindBugs (3.0.1) + FindSecurityBugs (1.7.1, 2017.11.20)	219 (549)
PMD (6.0.0, 2018.2.25)	34 (297)
LAPSE+ (2.8.1, 2012.2.11)	12 (12)
SonarQube (7.0, 2018.2.15)	58 (452)

표1(Table 29). Studied Bug Checkers

표 1(Table 1)은 본 연구에서 조사대상으로 선정한 4종의 오픈소스 도구가 제공하는 오픈소스 보안 취약점 결함 검출기의 개수이다. 참고를 위해, 각 프로젝트에 포함된 결함 검출기의 전체의 개수는 괄호 안에 표기하였다.

현존하는 오픈소스 보안 결함 검출기를 총체적으로 조사 범위에 포함시키기 위해서, 본 연구에서는 조사 대상으로 선정한 4개 프로젝트가 제공하는 결함 검출기 관련 문서, 결함 검출기 카테고리(분류 기준)을 면밀하게 검토하였다. 각 문서에는 특정 결함 검출기의 검출 대상, 작동 방식에 대한 문서를 제공하고 있으며, 각 도구별로 1개에서 10개의 카테고리로 결함검출기를 분류한 정보를 제공한다.

본 연구에서는 (1) 문서 에 보안과 연관성이 명시적으로 나타나 있는 결함검출기와 (2) 선정된 결함검출기와 같은 카테고리에 속한 결함검출기를 모두 선별하여, 포괄적인 분석이 될 수 있게 하였다.

3.2. 조사 방법

본 연구에서는 개발보안 관련 연구 및 소프트웨어 개발에 경험이 있는 1명의 대학원생과 1명의 학부생, 그리고 1명의 교수가 보안 취약점 진단 방법과 결함검출기의 설명, 관련 결함사례(예: 관련된 CVE, CWE 사례)를 정성적으로 분석하여, 검출 대상 결함이 같은 경우를 찾았다. 우선 모든 보안 취약점 점검 기준과 모든 결함검출기의 설명을 검토한 후, 각 보안 취약점 점검 기준을 중심으로 이와 연관된 결함검출기 집단을 정의하였다. 그 후, 같은 집단에 속한 두 검출기의 명세, 연관 결함 사례, 소스코드를 검토하여, 두 결함 검출기가 같은 대상을 결함으로 검출하는 경우를 찾았다. 이 때, 보안 취약점 진단 방법과 결함검

출기는 기본적으로 다대다 연관관계를 가질 수 있으며, 결함검출기 간에도 다대다 연관관계를 가질 수 있음을 전재로 조사를 수행하였다.

조사 과정에서, 대부분의 경우, 3명의 판별이 동일하여 연관성 판별이 명확하였다. 다만, 행정안전부의 보안 취약점 진단 기준에서 제시한 진단 방법에서 보안 취약점에 대한 설명은 일반적이나, 사례로 제시한 예제가 특정 상황에 국한되어 검출대상의 연관 여부에 합의가 되지 않는 9개의 경우가 있었다. 이 경우, 보안 취약점 진단 기준의 일부 설명에서 편협하게 한정된 부분을 확장하는 방향으로 해석하여, 최종적인 연관성 판별을 결정하였다(4.2절 참조).

조사 과정을, 보안 취약점 진단 기준 중 하나인 '경로 조작 및 자원 삽입'의 사례를 예로 들어 자세히 설명하겠다. '경로 조작 및 자원 삽입'은 사용자가 웹 어플리케이션의 입력에 악의적인 문자열을 삽입할 경우, 웹 서버가 의도하지 않은 파일을 접근하게 되는 결함을 뜻한다. 행정안전부의 보안 취약점의 점검 기준에서는 CWE-22, CWE-99를 관련 결함 사례로 제시하며, 해당 결함을 아래와 같이 간단히 기술하고 있다[8]:

"검증되지 않은 외부입력값을 통해 파일 및 서버 등 시스템 자원에 대한 접근 혹은 식별을 허용할 경우, 입력 값 조작을 통해 시스템이 보호하는 자원에 임의로 접근할 수 있는 보안 약점".

본 조사에서는 CWE-22, CWE-99를 언급하거나 혹은 보안 취약점 점검 기준에서 언급된 '경로(path)', '자원 삽입(injection)'의 키워드가 등장하는 모든 결함 검출기를 일차적으로 나열하였으며, 각 결함 검출기의 동작을 검토하여 FindBugs에서 10개, LAPSE+에서 1개로 총 11개의 관련 결함 검출기 집단을 정의하였다.

이들 11개 검출기는 정적 오염 분석(taint analysis)을 바탕으로 보안 결함을 검출하며, 각 검출기 별로 어떠한 함수 호출을 오염 발생 지점(source)과 오염 도달 지점(sink)로 지정하느냐에 따라 검출 대상이 달라짐을 확인할 수 있었다. 각 결함 검출기의 명세, 소스코드를 분석하여 오염 발생 지점, 오염 도달 지점을 분석한 결과, FindBugs의 'PATH_TRAVERSAL_IN'(표2의 19 행)와 LAPSE+의 'Path Traversal'(표 2의 29행) 검출기는 JDK의 File, RandomAccessFile, FileReader 객체의 메소드를 동일한 대상을 오염 도달 지점으로 정의하므로, 두 결함 검출기의 검출 대상이 같음을 확인할 수 있었다(표2의 19행과 29행의 2열에 나타난 결함 검출기 번호가 같음). 반면, 'PATH_TRAVERSAL_OUT'은 오염 도달 지점으로 FileWriter 객체의 메소드 등 다른 결함 검출기와 구별되는 결함을 검출하므로, 관련 결함 검출기 집단 중 중복되는 검출기가 없는 것으로 판별하였다(표2의 20행 결함 검출기 번호가 고유함).

3.3 조사 결과

본 연구에서는 앞서 선장한 42개의 보안 취약점 진단방법과 323개의 오픈소스 결함검출기 중 동일한 종류의 결함을 검출하는 경우를 파악했다.

표2(Table 2)는 총 42개의 보안 취약점 진단 기준 중 동일한 보안 결함을 검출하는 오픈소스 결함 검출기가 1건 이상 있는 32개 경우에 대하여, 각 진단 기준이 어떠한 결함 검출기와 동일한 범주의 결함을 검출하는 지를 대응한 결과다. 표2는 총 238행을 3단 병행으로 표현한 것으로, 병합된 열로 표현된 행은 보안 취약 진단 기준의 이름을 표시하고 있으며, 그 아래 3열은 각각 결함 검출기 프레임워크의 이름(F는 FindBugs, L은 LARSE+, P는 PMD, S는 SonarQube를 지칭함), 결함 패턴 고유 번호, 결함 검출기 명칭을 지칭하고 있다. 결함 패턴 고유 번호는 검출기의 검출 대상 결함 패턴이 상이한 경우, 고유한 번호를 부여하였다(1~211). 서로 다른 결함 검출기이지만 동일한 고유 번호를 부여 받은 경우는, 해당 결함 검출기가 동일한 결함 패턴을 검출하는 것으로 판단되는 경우로, 고유 번호 위에 별표(*)로 표시하였다. 3열에 표시한 결함검출기 이름이 길 경우 말줄임표로 간단히 표시하였다. 표2의 데이터는, 관련 URL을 포함하여, 공개 소프트웨어 저장소에 공개하여 일반에 활용할 수 있도록 하였다.

다음 10개 보안 취약점 진단 기준은 총 323개의 오픈소스 결함 검출기 중에 대응되는 검출기가 1건도 발견되지 않았고, 따라서 표2에 표시되지 않았다(자세한 논의는 4.1절 참고):적절한 인증 없는 중요기능 허용, 부적절한 인가, 중요한 자원에 잘못된 권한 설정, 중요 정보 평문 저장, 취약한 비밀번호 사용, 주석문에 포함된 시스템 주요정보, 솔트 없이 일방향 해시함수 사용, 반복된 인증시도 제한기능 부재, DNS lookup에 의존한 보안결정.

총 323개의 조사대상 결함 검출기 중 1개 이상의 보안 취약점에 대응되는 경우는 총 285건이었으며, 이 중 동일한 결함 검출 대상을 가지는 결함 검출기를 고려할 경우, 이들은 총 211종의 고유한 결함 패턴을 검출하였다. 총 323개의 조사대상 결함 검출기 중 38개는 42개의 보안 취약점 진단 기준 중 어디에도 대응되지 않아 표2에 표시되지 않았다(자세한 논의는 4.2절과 4.3절 참고).

4. 관찰 및 논의

본 장에서는 3장에서 소개한 결과를 관찰하여 행정안전부가 발표한 보안 취약점 진단 기준의 제한점, 효과적인 활용 방안, 그리고 발전을 위한 개선 방향을 논의한다.

표3(Table 3)은 3장의 결과에 대한 관찰과 논의를 위해, 표2(Table 2)의 내용을 행정안전부 결함 검출 기준을 중심으로 요약한 데이터다. 표3은 상하로 크게 두 부분으로 나누어지는데 1~44행까지는 보안 취약점 진단 방법과 결함검출기 사이의 연관관계 있는 경우에 대한 결과이며, 45~50행은 연관된 행정안전부 보안 취약점 진단 기준이 없는 오픈소스 결함 검출기에 대한 결과이다. 표3의 1~44행 부분에는 7열이 있는데, 첫 번째 열과 두 번째 열은 보안 취약점 점검방법의 커테고리와 구체적인 방법 이름을 보여준다. 세 번째 열에서 여섯 번째 열은 각각 FB+FSB(F), PMD(P), LAPSE+(L), SonarQube(S)에 해당 검토방법과 연관된 결함검출기의 개수 나타낸다. 이 때 괄호 안의 숫자는 해당 보안 취약점 점검 방법에 직접해당 되지는 않으나, 관련성이 높은 결함검출기의 개수이다(4.2절 참고). 마지막 열은, 앞선 4개의 열에서 보고하는 결함검출기 중 중복된 것을 제외한, 결함검출 의도가 다른 결함검출기의 숫자이다. 표3의 45~50번째 행은 연관된 보안 취약점 진단방법이 없는 결함검출기에 대한 정보를 보고한다. 이 영역은 총 6개의 행으로 나누어졌으며, 첫 번째 행은 본 연구에서 명명한 결함 패턴 이름이며, 나머지 행은 앞서 설명한 바와 같이, 4개의 결함검출기 프레임워크에서 해당 결함패턴에 대해 존재하는 결함 검출기의 숫자를 나타낸다.

4.1 자동 진단을 지원하는 결함 검출기가 없는 보안 취약점 진단 방법

표3에서 마지막 열이 0인 9개 취약점 진단 방법은 현존 하는 오픈소스 결함 검출기 중 대응되는 경구가 없는 경우로, 오픈소스 도구를 통한 자동적인 진단을 지원하는 방법이 없는 것으로 관찰 되었다. 이에 해당하는 보안 취약점 진단방법은 다음과 같다:

- 적절한 인증 없는 중요기능 허용 (표3의 17행)
- 부적절한 인가 (표3의 18행)
- 중요한 자원에 잘못된 권한 설정 (표3의 19행)
- 중요 정보 평문 저장 (표3의 21행)
- 취약한 비밀번호 사용 (표3의 26행)
- 주석문에 포함된 시스템 주요정보(표3의 29행)
- 솔트 없이 일방향 해시함수 사용(표3의 30행)
- 반복된 인증시도 제한기능 부재 (표3의 32행)
- DNS lookup에 의존한 보안결정(표3의 43행)

해당 보안 취약점 진단 방법에 대한 결함 검출기가 존재하지 않는 이유를 파악하기 위해, 해당 보안 취약점 진단 기준을 정성적으로 분석하였다.

분석 결과, 'DNS lookup에 의한 보안결정'의 경우, 해당 결함 검출 기준을 바탕으로 패턴 기반 결함 검출기 개발이 가능하나, 아직 이루어지지 않은 상황으로 파악되었다. 객관적이고 효율적인 보안 취약점 진단을 위해, 'DNS lookup에 의한 보안결정'을 FindSecurityBugs와 같은 오픈소스 정적 분석기 프레임워크 내에 결함 검출기로 개발하여 활용성이 높은 도구로 확보하는 것이 유용할 것으로 보인다.

나머지 8개 보안 취약점 진단 기준의 경우, 다음 2 가지 원인으로 인해 연관된 결함 검출기의 개발이 어려웠던 것으로 파악되었다:

① 동적 정보가 필수적인 보안 취약점 진단 기준. 아래의 3 가지 진단과정 중 코드 정보 이외에 어플리케이션 별 기능을 고려한 동작(동적) 정보가 필수적으로 요청되어, 결함검출기 개발이 원천적으로 어려운 것으로 판별 된다:

- 적절한 인증 없는 중요기능 허용
- 부적절한 인가
- 반복된 인증시도 제한기능 부재

예를 들어, '적절한 인증 없는 중요기능 허용'의 경우, 프로그램의 코드만을 가지고, 사용자 인증이 된 상태인지, 안 된 상태인지 일반적으로 알 수 없으므로, 코드 정보 외에 프로그램 실행을 고려한 분석이 필요하다. 이와 같은 진단 방법의 경우로, 코드 정보만을 활용하는 정적분석 기반 결함검출기의 설계가 원천적으로 어려우므로, 시큐어 코딩 가이드라인만으로는 정확한 검침이 어려운 상황이다. 해당 보안 취약점을 정확하게 진단하기 위해서는, 코드 정보에 기반한 진단 기준 이외에 프로그램 동적 정보나 보안 테스팅 정보를 활용하는 개발보안 방법론의 적용이 요청되는 주제로 보인다(4.6절).

② 취약점 여부 판별 기준이 모호한 진단 기준. 아래의 5개 진단 기준은 보안 취약점 판별 조건이 모호하게 작성되었거나, 특정 세부사항에 한정되어 편협하게 개발되어, 객관적인 검출 기준을 마련하기 어렵고, 따라서 관련된 결함 검출기를 특정하기 어려운 경우로 판별 되었다:

- 중요한 자원에 대한 잘못된 권한 설정
- 중요정보 평문 저장
- 취약한 비밀번호 저장
- 주석문 안에 포함된 시스템 주요정보
- 솔트 없이 일방향 해쉬 함수 사용

표2(Table 11). Mapping from MOIS Inspection Criteria to Corresponding Security Bug Checkers
(F: FindBugs+FindSecurityBugs, P: PMD, L: LARSE+,S: SonarQube)

		SQL 삽입
F	1	CUSTOM_INJECTION
F	2*	SQL_INJECTION
F	3	SQL_INJECTION_TURBINE
F	4	SQL_INJECTION_HIBERNATE
F	5	SQL_INJECTION_JDO
F	6	SQL_INJECTION_JPA
F	7	SQL_INJECTION_SPRING_JDBC
F	8	SQL_INJECTION_JDBC
F	9	SCALA_SQL_INJECTION_SLICK
F	10	SCALA_SQL_INJECTION_ANORM
F	11	SQL_INJECTION_ANDROID
F	12	AWS_QUERY_INJECTION
F	13	SQL_NONCONSTANT_STRING…
F	14*	SQL_PREPARED_STATEMENT…
L	2*	SQL Injection
S	14*	SQL binding mechanims should …
		경로조작 및 자원삽입
F	15*	PATH_TRAVERSAL_IN
F	16	PATH_TRAVERSAL_OUT
F	17	SCALA_PATH_TRAVERSAL_IN
F	18	STRUTS_FILE_DISCLOSURE
F	19	SPRING_FILE_DISCLOSURE
F	20	REQUESTDISPATCHER_FILE…
F	21	EXTERNAL_CONFIG_CONTROL
F	22	BEAN_PROPERTY_INJECTION
F	23	PT_ABSOLUTE_PATH_TRAVERSAL
F	24	PT_RELATIVE_PATH_TRAVERSAL
L	15*	Path Traversal
		크로스사이트 스크립트
F	25*	XSS_REQUEST_WRAPPER
F	26	JSP_JSTL_OUT
F	27	XSS_JSP_PRINT
F	28	XSS_SERVLET
F	29	ANDROID_GEOLOCATION
F	30	A.W.V_JAVASCRIPT
F	31	A.W.V_JAVASCRIPT_INTERFACE
F	32	HTTPONLY_COOKIE
F	33	SCALA_XSS_TWIRL
F	34	SCALA_XSS_MVC_API
F	35	XRP_TO_JSP_WRITER
F	36	X.R.P_TO_SEND_ERROR
F	37	X.R.P_TO_SERVLET_WRITER
L	25*	Cross-Site Scripting(XSS)
		운영체제 명령어 삽입
F	38*	COMMAND_INJECTION
F	39	SCALA_COMMAND_INJECTION
L	38*	Command Injection
S	38*	Values passed to OS command …
		위험한 형식 파일 업로드
F	40	WEAK_FILENAMEUTILS
F	41	FILE_UPLOAD_FILENAME
6. 신뢰되지 않는 URL 주소로 자동접속 연결		
F	42*	UNVALIDATED_REDIRECT
F	43	PLAY_UNVALIDATED_REDIRECT
F	44	SPRING_UNVALIDATED_REDIRECT
L	42*	URL Tampering
		XQuery 삽입
F	45	XMLStreamReader
F	46	XXE_SAXPARSER
F	47	XXE_XMLREADER
F	48*	XXE_DOCUMENT
F	49	XXE_DTD_TRANSFORM_FACTORY
F	50	XXE_XSLT_TRANSFORM_FACTORY
F	51	XML_DECODER
F	52	JSP_XSLT
F	53	MALICIOUS_XSLT
L	48*	XML Injection
		XPath 삽입
F	54*	XPATH_INJECTION
L	54*	Xpath Injection
		LDAP 삽입
F	55*	LDAP_INJECTION
F	56	LDAP_ANONYMOUS
F	57	LDAP_ENTRY_POISONING
L	55*	LDAP Injection
S	55*	Values passed to LDAP queries …
		크로스사이트 요청 위조
F	58	S.C_PROTECTION_DISABLED
F	59	S.C_UNRESTRICTED_REQUEST…
F	60	SCALA_PLAY_SSRF
F	61	URLCONNECTION_SSRF_FD
P	62	NoUnsanitizedJSPExpression
		HTTP 응답분할
F	63*	HTTP_RESPONSE_SPLITTING
L	64*	HRS_REQUEST_PARAMETER…
	64*	Header Manipulation
	63*	HTTP Response Splitting
		정수형 오버프로우 [Extended]
F	65	BAD_HEXA_CONVERSION
P	66	BadComparison
S	67	Double.longBitsToDouble
		보안기능 결정에 사용되는 부적절한 입력값
F	68*	SERVLET_PARAMETER
F	69	SERVLET_CONTENT_TYPE
F	70	SERVLET_SERVER_NAME
F	71*	SERVLET_SESSION_ID
F	72	SERVLET_QUERY_STRING
F	73	SERVLET_HEADER
F	74*	SERVLET_HEADER_REFERER
F	75	SERVLET_HEADER_USER_AGENT
F	76	HTTP_PARAMETER_POLLUTION
F	77	TRUST_BOUNDARY_VIOLATION
L	78	Cookie Poisoning
L	68*	Parameter Tampering
S	71*	HttpServletRequest.getRequested…
S	74*	HTTP referers should not be…
S	79	Untrusted data should not be …
		포맷 스트링 삽입
F	80	FORMAT_STRING_MANIPULATION
		취약한 암호화 알고리즘 사용
F	81	WEAK_MESSAGE_DIGEST_MD5
F	82*	WEAK_MESSAGE_DIGEST_SHA1
F	83	SSL_CONTEXT
F	84*	CUSTOM_MESSAGE_DIGEST
F	85	HAZELCAST_SYMMETRIC…
F	86*	NULL_CIPHER
F	87*	DES_USAGE
F	88	TDES_USAGE
F	89	RSA_NO_PADDING
F	90	ECB_MODE
F	91	PADDING_ORACLE
F	92	ESAPI_ENCRYPTOR
F	93	CIPHER_INTEGRITY
S	87*	NeitherDES nor DESede…
S	94	Cryptographic RSA algorithms …
S	86*	"javax.crypto.NullCipher" …
S	84*	Only standard cryptographic…
S	95	Pseudorandom number generators…
S	82*	SHA-1 and Message-Digest hash…
		중요정보 평문 전송
F	96	DEFAULT_HTTP_CLIENT
F	97	UNENCRYPTED_SOCKET
F	98	UNENCRYPTED_SERVER_SOCKET
F	99*	INSECURE_COOKIE
F	100	INSECURE_SMTP_SSL
F	101	URL_REWRITING
S	99*	Cookies should be secure
		하드코드된 비밀번호
F	102	HARD_CODE_PASSWORD
F	103*	DMI_CONSTANT_DB_PASSWORD
F	104	DMI_EMPTY_DB_PASSWORD
S	103*	Credentials should not be…
		충분하지 않은 키 길이 사용
F	105	BLOWFISH_KEY_SIZE
F	106	RSA_KEY_SIZE
		적절하지 않은 난수값 사용
F	107	PREDICTABLE_RANDOM
F	108	PREDICTABLE_RANDOM_SCALA
S	109	"SecureRandom" seeds should …
		하드코드된 암호화 키
F	110	HARD_CODE_KEY
S	111	IP addresses should not be …
		사용자에게 저장되는 쿠키를 통한 정보노출
F	112	COOKIE_USAGE
F	113	COOKIE_PERSISTENT
F	114	HRS_REQUEST_PARAMETER_TO…
		부적절한 검사 없는 코드 다운로드
F	115	USP_INCLUDE
S	116	Classes should not be loaded ….
		경쟁조건(TOCTOU)
F	117	AT_OPERATION_SEQUENCE_ON_…
F	118	DC_DOUBLECHECK
F	119	DC_PARTIALLY_CONSTRUCTED
F	120	D.S.O_BOOLEAN
F	121	D.S.O_UNSHARED_BOXED_…
F	122	D.S.O_SHARED_CONSTANT
F	123	D.S.O_UNSHARED_BOXED_…
F	124	DM_MONITOR_WAIT_ON_…
F	125	DM_USELESS_THREAD
F	126	ESync_EMPTY_SYNC
F	127	IS2_INCONSISTENT_SYNC
F	128	IS_FIELD_NOT_GUARDED
F	129	JLM_JSR166_LOCK_MONITORENTER
F	130	JLM_JSR166_UTILCONCURRENT…
F	131	JML_JSR166_CALLING_WAIT…
F	132	LI_LAZY_INIT_STATIC
F	133	LI_LAZY_INIT_UPDATE_STATIC
F	134	ML_SYNC_ON_FIELD_TO_GUARD…
F	135	ML_SYNC_ON_UPDATED_FIELD
F	136	MWN_MISMATCHED_NOTIFY
F	137	MWN_MISMATCHED_WAIT
F	138	NN_NAKED_NOTIFY
F	139	NO_NOTIFY_NOT_NOTIFYALL
F	140	RS_READOBJECT_SYNC
F	141	RV_RETURN_VALUE_OF…
F	142	RU_INVOKE_RUN
F	143	SC_START_IN_CTOR
F	144	SP_SPIN_ON_FIELD
F	145	STCAL_INVOKE_ON_STATIC_…
F	146	STCAL_INVOKE_ON_STATIC_DATE_…
F	147	STCAL_STATIC_CALENDAR_INST…
F	148	STCAL_STATIC_SIMPLE_DATE…
F	149	SWL_SLEEP_WITH_LOCK_HELD
F	150*	ITLW_TWO_LOCK_WAIT
F	151	UG_SYNC_SET_UNSYNC_GET
F	152	UL_UNRELEASED_LOCK
F	153	UL_UNRELEASED_LOCK_EXCEPTI…
F	154	UW_UNCOND_WAIT
F	155	VO_VOLATILE_INCREMENT
F	156	VO_VOLATILE_REFERENCE…
F	157	WL_USING_GETCLASS_RATHER…
F	158	WS_WRITEOBJECT_SYNC
F	159	WA_AWAIT_NOT_IN_LOOP
F	160	WA_NOT_IN_LOOP
P	161	AvoidSynchronizedAtMethodLevel
P	162	AvoidUsingVolatile
P	163	DoubleCheckedLocking
P	164	NonThreadSafeSingleton
P	165	UnsynchronizedStaticDateFormatter
P	166	UseConcurrentHashMap
S	150*	"wait" should not be called…
S	167	Value-based classes should not be…
S	168	"getClass" should not be used for…
S	169	Getters and setters should be…
S	170	Non-thread-safe fields should not…
S	171	Blocks should be synchronized on…
S	172	".equals()" should not be used to…
S	173	Synchronization should not be…
		종료되지 않은 반복문 또는 재귀함수
F	174	IL_CONTAINER_ADDED_TO_ITSELF
F	175*	LI_INFINITE_LOOP
F	176	LI_INFINITE_RECURSIVE_LOOP
P	177	EmptyWhileStmt
S	175*	Loops should not be infinite
S	178	Double-checked locking should …
S	179	Locks should be released
		오류메시지를 통한 정보노출
S	180	Throwable.printStackTrace(…) shoul…
		오류 상황 대응 부재
P	181	AvoidInstanceofChecksInCatch…
P	182	AvoidLiteralsInIfCondition
P	183	CloneThrowsCloneNotSupported…
P	184	DoNotExtendJavaLangThrowable
P	185	EmptyCatchBlock
P	186	ReturnFromFinallyBlock
S	187	Exceptions should not be thrown…
S	188	"SingleConnectionFactory" …
S	189	"Iterator.next()" methods should…
S	190	Return values should not be ignored…
S	191	Exception should not be created…
		부적절한 예외 처리
F	192	DE_MIGHT_DROP
F	193*	DE_MIGHT_IGNORE
P	193*	AvoidCatchingNPE
P	193*	AvoidLosingExceptionInformation
P	194	UseCorrectExceptionLogging
P	195	DoNotThrowExceptionInFinally
S	193*	"InterruptedException" should not…
		잘못된 세션에 의한 데이터 정보노출
F	196*	MSF_MUTABLE_SERVLET_FIELD
P	197	StaticEJBFieldShouldBeFinal
S	198	Members of Spring components…
S	196*	Servlets should not have mutable …
		제거되지 않고 남은 디버그 코드
S	199	Web applications should not have…
		Public 메소드로부터 반환된 Private 배열
F	200*	EI_EXPOSE_REP
F	201	MS_EXPOSE_REP
S	200*	Mutable members should not be…
		Private 배열에 Public 데이터 할당
F	202	EI_EXPOSE_REP2
		취약한 API
F	203	DM_EXIT
F	204	DM_RUN_FINALIZERS_ON_EXIT
P	205	AvoidThreadGroup
P	206	DoNotUseThreads
P	207*	DontCallThreadRun
P	208	ProperCloneImplementation
P	209	UseNotifyAllInsteadOfNotify
P	210	UseProperClassLoader
S	211	"File.createTempFile" should not be…
S	207*	Thread.run() should not be called…

예를 들어, '중요정보 평문 저장'의 경우 프로그램에서 저장하는 수많은 정보들 중, 중요한 정보가 무엇인지를 일반적으로 판별하는 기준을 세우기 어렵다. '솔트 없이 일방향 해쉬 함수 사용'의 경우, 해쉬 함수를 사용한 상황에 따라 보안 취약점을 발생시킬 수도 있고, 그렇지 않을 수도 있으므로 진단 기준이 모호하다. 이와 같은 진단 방법의 경우, 과학적이고 체계적인 진단 기준을 정의하기 어렵고, 따라서 결함검출기와 같은 자동화 기법의 도입이 어렵다. 해당 보안 취약점 진단 기준에 대응되는 자동화된 결함 검출기를 개발하기 위해서는, 실제 결함 사례를 중심으로 하여 진단기준을 구체화하는 개선이 필수적으로 보인다. 예를 들어, '중요정보 평문 저장'의 경우 중요정보를 URL, 특정 패턴의 문자열로 구체화하는 접근이 가능할 것으로 보인다.

4.2 진단 기준이 편협하여 개선이 필요한 보안 취약점 진단 방법

표3의 괄호 안에 표시한 9개의 결함 검출기는, 해당 보안 취약점 진단 기준이 특정 하는 진단 대상 결함에 대한 설명에 따를 경우 대응되지 않으나, 진단 기준의 편협한 조건을 개선할 경우 대응 관계를 찾을 수 있는 경우이다. 해당 9개 결함 검출기가 연관된 6개 보안 취약점 진단 기준의 개선 방향은 다음과 같이 논의할 수 있다:

- '크로스사이트 요청 위조'의 경우, 크로스사이트 요청 위조와 본질적으로 유사한 기존 진단 기준에서는 Server Side Request Forgery (SSRF) 결함검출기를 포함하도록 확장
- '정수형 오버플로우'의 경우, 타입 변환(예: int를 double로)에서 발생하는 버그 등 기본형 타입의 예외적 연산에 대한 확장이 필요함
- '보안기능 결정에 사용되는 부적절한 입력 값' 의 경우, 부적절한 값 조건은 물론 입력 값의 속성에 위험요소가 없는 지 검사하는 결함검출기를 포괄하도록 확장
- '취약한 암호화 알고리즘의 사용'은 암호화 알고리즘을 사용의 부적절한 사용으로 인한 결함패턴을 아우르도록 확장
- '중요정보 평문전송'의 경우, 중요정보의 다양한 종류(예: URL, IP 주소, 해쉬 키)를 포괄할 수 있도록 확장
- '하드코드된 암호화 키'의 경우, 암호화 키 외에도 하드코드가 되었을 경우 위험성이 있는 정보(예: IP 주소)를 고려하도록 확장

표3(Table 3). Associating MOIS Security Vulnerability Weakness Inspection Methods and Security Vulnerability Bug Checker (F: #. checkers in FindBugs and FindSecurity Bugs, P: #. checkers in PMD, L: #. checkers in Larse-Plus, S: #. checkers in SonarQube, Tot the sum of F, P, L and S)

MOIS Security Vulnerability Weakness Inspection Method		Bug Checkers				
Categ	**Security Vulnerability Issue**	**F**	**P**	**L**	**S**	**Tot**
입력 데이터 검증 및 표현	SQL 삽입	14	0	1	1	16
	경로 조작 및 자원 삽입	10	0	1	1	12
	크로스사이트 스크립트	13	0	1	0	14
	운영체제 명령어 삽입	2	0	1	1	4
	위험한 형식 파일 업로드	2	0	0	0	2
	신뢰되지 않는 URL 자동접속연결	3	0	1	0	4
	XQuery 삽입	9	0	1	0	10
	Xpath 삽입	1	0	1	0	2
	LDAP 삽입	3	0	1	1	5
	크로스사이트 요청 위조	4 (2)	1	0	0	5
	HTTP 응답분할	2	0	2	0	4
	정수형 오버플로우	1 (1)	1 (1)	0	1 (1)	3
	보안기능 결정에 부적절한 입력값 사용	10 (1)	0	2	3	15
	포맷 스트링 삽입	1	0	0	0	1
보안 기능	적절한 인증 없는 중요기능 허용	0	0	0	0	0
	부적절한 인가	0	0	0	0	0
	중요한 자원에 잘못된 권한 설정	0	0	0	0	0
	취약한 암호화 알고리즘 사용	13 (1)	0	0	6	19
	중요정보 평문 저장	0	0	0	0	0
	중요정보 평문 전송	6 (1)	0	0	1	7
	하드코드된 비밀번호	3	0	0	1	4
	충분하지 않은 키 길이 사용	2	0	0	0	2
	적절하지 않은 난수값 사용	2	0	0	1	3
	취약한 비밀번호 사용	0	0	0	0	0
	하드코드된 암호화 키	1	0	0	1 (1)	2
	저장된 쿠키를 통한 정보노출	3	0	0	0	3
	주석문에 포함된 시스템 주요정보	0	0	0	0	0
	솔트 없이 일방향 해시함수 사용	0	0	0	0	0
	무결성 검사 없는 코드 다운로드	1	0	0	1	2
	반복된 인증시도 제한 기능 부재	0	0	0	0	0
시간, 상태	경쟁조건: 검사 시점과 사용 시점	44	6	0	8	58
	종료되지 않는 반복문/재귀함수	3	1	0	3	7
에러 처리	오류 메시지를 통한 정보노출	0	0	0	1	1
	오류 상황 대응 부재	0	6	0	5	11
	부적절한 예외 처리	2	4	0	1	7
캡슐화	잘못된 세션에 의한 데이터 정보노출	1	1	0	2	4
	제거되지 않고 남은 디버그 코드	0	0	0	1	1
	시스템 데이터 정보노출	1	0	0	0	1
	Public 메서드의 Private 배열 반환	2	0	0	1	3
	Private 배열에 Public 데이터 할당	1	0	0	0	1
API 오용	DNS lookup에 의존한 보안결정	0	0	0	0	0
	취약한 API 사용	2	6	0	2	10
Uncovered Security Vulnerability Issues		**F**	**P**	**L**	**S**	**Tot**
Misuse of Cryptology Algorithms		4	0	0	0	4
Incorrect Use. Framework Features		13	0	0	6	19
Overconsumption. System Resource		1	0	0	2	3
Code Injection		10	0	0	0	10
Log Injection		2	0	0	0	2

앞서 논의한 6개 보안 취약점 진단 기준의 경우, 현행 보안 진단 기준이 특정 하는 보안 취약점 결함 이외에도, 해당 취약점과 연관성이 높은 보안 취약점 결함 패턴이 추가로 존재하는 경우다. 따라서 해당 보안 취약점 진단 기준을 보다 포괄적인 보안 취약점 검출이 가능하도록 확장하는 개선이 요청된다.

4.3 현재 보안 취약점 진단 기준에는 없으나, 결함검출기가 탐지하는 보안 취약점 결함 패턴

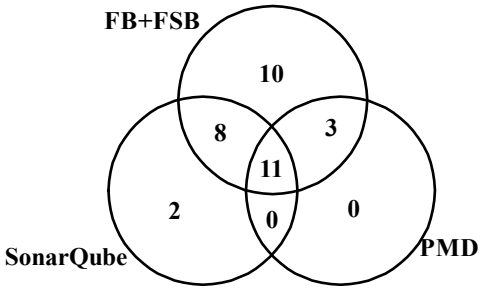
표3의 46~50행에서는 현재 행정자치부 보안 취약점 진단 기준과는 무관하나 오픈소스 결함 검출기가 개발된 총 38개의 결함 검출기를 5가지 유형으로 분류한 결과를 소개하고 있다. 이들 38개 검출기의 검출 대상을 조사한 결과, 관련된 보안 취약점이 2013년 이후에 정리되었으며, 이들 대부분이 2017년 이후에 보고된 보안 취약점에 착안하여 개발된 것을 확인할 수 있었다. 이에 해당하는 5가지 보안 결함 유형에 대한 설명은 다음과 같다:

- 'Misuse of Cryptology Algorithm'은 암호화 알고리즘의 잘못된 사용으로 인해 효과적인 암호화가 실패하는 보안 취약점 결함
- 'Incorrect Use of Framework Feature'는 Spring, Android과 같은 프레임워크의 고유한 개발 규칙 위반에 따른 보안 취약점(예: 특정 메서드의 접근 제어자를 보안에 취약하게 설정)
- 'Overconsumption of System Resource'는 시스템 자원을 무리하게 사용하여 다른 사용자들이 시스템 자원을 사용할 수 없게 되는 취약점
- 'Code Injection'은 악의적인 코드가 프로그램 내부에서 실행되는 결함으로, 예를 들어, 무분별한 deserialization으로 인해 공격자의 코드가 프로그램 내부에서 실행되도록 허용하는 취약점
- 'Log Injection'은 프로그램 실행결과 발생하는 로그를 조작하여 침입 분석과 같은 보안활동을 방해 하는 보안 취약점에 관한 패턴임

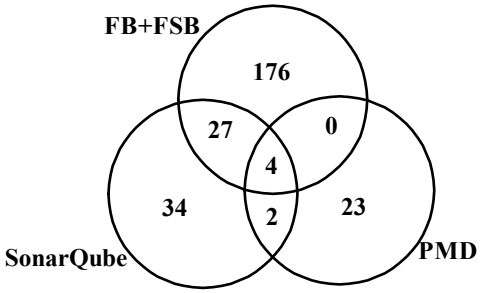
앞서 설명한 5종의 보안 결함 유형은 실제 산업현장에서 새롭게 파악된 보안 취약점으로, 이들이 보안 취약점 진단 과정에서 검출 될 수 있도록 취약점 진단 기준의 확장이 요청된다.

4.4 보안 취약점 결함검출기 프레임워크 간 비교

그림1(Figure 1)은 표2의 데이터를 바탕으로, 본 논문의 논의 대상 중 3개의 오픈소스 결함검출기 프레임워크의 소속 결함검출기의 검출 대상 범위를 보안 취약점 진단방법의 개수(그림1(a))과 결함 검출기의 개수(그림1(b))를 기준으로 비교한 결과이다. 밴다이어그램의 각 영역은, FB+FSB, PMD, SonarQube의 각 조합이 1개 이상의 결함검출기로 연관된 진단방법의 수(그림1(a) 혹은 해당 결함검출기의 수(그림1(b))를 나타낸다. 그림에 나타내지 않은 LAPSE+의 경우, 제공하는 12 결함검출기가 FB+FSB에 모두 완전히 포함되어, 그림1에는 표현하지 않았다.



(a) Number of MOIS Review Criteria



(b) Number of Static Bug Checkers

그림 1(Figure 1). Comparison of Target Coverage Among Three Open-source Static Bug Checker Frameworks

그림1(a)에서 확인할 수 있는 바와 같이, 조사 대상 오픈소스 결함 검출 프레임워크 중에서는FB+FSB가 가장 많은 수의 진단 방법과 연관된 결함검출기를 제공하고 있으며, SonarQube와 연관된 2개의 진단기준을 제외하고 대부분의 다른 프레임워크의 진단방법 범위를 포함하고 있음을 알 수 있다.

그림1(b)를 볼 때, 상이한 두 프레임워크가 같은 보안 취약점 진단방법과 연관된 결함검출기를 가지고 있다고 하더라도, 결함검출기의 기능이 일치하는 경우는 전체의 12.8%(=34/265)로, 나머지 87.2% 결함검출기는 고유한 결함 패턴을 검출 대상으로 하고 있음을 알 수 있다.

따라서, 보안 취약점 점검에 시간/자원이 한정된 상황에서는 보안 취약점 진단방법 커버리지를 고려하여 FB+FSB와 SonarQube 중 일부(FB+FSB가 검사하지 못하는 점검방법)의 우선적 적용을 생각해 볼 수 있다. 이에 더하여, 진단방법 내에도 결함검출기

별로 검출을 목표로 하는 결함 유형이 다를 수 있으므로, 본 연구에서 파악한 결함검출기 간의 상관관계를 고려하여 총 265개의 고유한 결함검출기 전체를 활용한 보안 취약점 진단을 고려해볼 수 있다.

자원의 제약(예: 시간, 인력)이 적은 경우에는, 두 결함검출기가 같은 결함 유형을 검출하는 연관성이 높은 결함검출기로 판단된다고 하더라도, 각 구현에 있어서 구체적인 사례에서 탐지 결과가 다를 수 있으므로(예: 오탐률), 여러 개의 결함검출기 프레임워크를 모두 수행한 후, 본 연구 결과가 파악한 연관관계를 바탕으로 상호 간 결과의 정합성을 비교할 경우, 보다 정확하고 총체적인 보안 취약점 진단 결과를 얻을 수 있을 것으로 기대된다.

4.5 개발 보안 방법론 개선을 위한 제안

3장의 조사 결과와 4장의 앞선 논의를 종합해 볼 때, 현재 개발보안 방법론의 한계를 개선하고 위한 방법으로 다음과 같은 발전 과제를 발견할 수 있다:

① **동적 정보를 활용하는 보안 테스트 가이드라인의 개발: 4.1절에서 지적하고 있는 바와 같이, 다수의 보안 취약점은 일반적인 코드 패턴보다, 어플리케이션의 동작에 기반 한 검토가 필수적이다. 이를 효과적으로 지원하기 위해서는 시큐어 코딩 기반의 방법론에 더하여, 검증대상 프로그램에 대한 테스팅을 수행하며, 동적 정보를 활영하여 보안 취약점을 체계적으로 보안 테스트 가이드라인의 개발이 필요한 것으로 보인다.**

② 신규로 발견되는 보안 취약점이 진단 기준에 지속적으로 반영될 수 있도록 하는 시스템 개발: 4.2절과 4.3절에서 지적하고 있는 바와 같이, 소프트웨어 개발 기술의 빠른 변화에 따라 새로운 보안 취약점이 지속적으로 발생하고, 이에 대한 대응책 역시 빠른 속도로 개발되고 있다. 하지만 현재와 같은 형식의 보안 취약점 진단 기준은 2013년 이후 상황에 능동적인 반영에 한계가 있다. 국내 소프트웨어 개발 환경에서 새로운 보안 취약점 점검기준의 지속적인 발전을 위해서는, 보안 취약점 보고를 수집, 연계하는 온라인 색인 시스템(예: CVE, CWE)이 필요하며, 이를 통해 국내 IT산업의 개발자가 개발 보안에서의 새로운 이슈를 용이하게 접근할 수 있도록 지원할 필요가 있는 것으로 보인다.

③ 보안 취약점 진단기준의 활용에 필요한 오픈소스 형태의 결함검출기 및 활용 방법 개발: 산업체 환경에서 보안 취약점의 효과적인 진단을 위해서는 오픈소스 기반의 결함검출기의 활용이 절대적으로 필요하다. 이러한 필요에 효과적으로 응답하기 위해서 기존에 개발된 진단기준 적용에 필요한 결함검출기를 FindSecurityBugs와 같은 오픈소스 프레임워크 상에서 추가 개발하여, 공개하고 해당 프레임워크를 활용한 자동 진단 방법론의 보완이 필요할 것으로 보인다. 또한, 4.4절의 논의와 같이, 현업에서도 복수 개의 오픈소스 결함 검출 프레임워크를 복합적으로 활용함으로써 효과적이면서도 효율적으로 시큐어 코딩 방법론을 적용하는 방법론 구축을 지원할 필요가 있다.

5. 결론

본 논문에서는 Java 프로그램에 대하여 동작하는 총 323종의 오픈소스 보안 취약점 결함검출기와 행정안전부가 2013년 발표한 보안 취약점 진단 방법 간의 대응 관계를 총체적이고 구체적으로 식별한 연구 결과를 소개한다. 조사연구 결과를 통해서 특정 결함검출기가 탐지하는 보안성 결함이 어떠한 보안 취약점 진단 방법과 대응되는지를 파악할 수 있으며, 또한 같은 보안 취약점 결함을 탐지하는 서로 다른 결함검출기 간의 관계를 구체적으로 파악하였다. 또한, 조사연구에서의 관찰을 바탕으로 오픈소스 결함검출기를 행정안전부 보안 취약점 진단 기준 방법론에 적용하는데 있어서의 문제점을 구체적으로 소개하고 개발보안 방법론 개선 방안을 제안하였다.

참 고 문 헌

[1] US-CERT, OpenSSL 'Heartbleed' Vulnerability (CVE-2014-0160), https://www.us-cert.gov/ncas/alerts/TA14-098A

[2] Ministry of the Interior and Safety, Guide of Validating Software Security Weakness for e-Government Software Validators, 2013

[3] FindBugs, https://findbugs.sourceforge.net

[4] PMD, https://pmd.github.io

[5] Jiho Bang, Rhan Ha, Jung Whan Park, Pil Young Kang, Minimum Standard of Weakness in Development of Reliable e-GOV Software, Proceedings of Symposium of the Korean Institute of Communications and Information Sciences, 2012

[6] Joonseon Ahn, Eunyoung Lee, Byeong-Mo Chang, A Study on Security Weakeness for Secure Software Development (SW 개발보안을 위한 보안약점 표준목록 연구), Journal of Korea Institute of Information Security and Cryptology

[7] Jiho Bang, Trend in Open-source Security Vulnerability Detection Tools (공개용 소스코드 보안약점 분석도구 개발 동향), Internet and Security Focus, Korea Internet & Security Agency, May

2014

[8] Ministry of the Interior and Safety, Manual on Validating Security Issues Using Open Source Tools for Software Developers and Validators (전자정부 SW 개발자, 진단원을 위한 공개SW를 활용한 소프트웨어 개발보안 진단가이드), 2016

[9] Jiho Bang, Rhan Ha, Validation Test Codes Development of Static Analysis Tool for Secure Software, Journal of the Korean Institute of Communication Sciences, 38 (5), 2013

[10] Jiho Bang, Rhan Ha, Comparing Open Source Static Security Analysis Tools based on Software Weakness, Proceedings of Korea Computing Congress, June 2013

[11] Joonseon Ahn, Ji-ho Bang, Eunyoung Lee, Quantitative Scoring Criteria on the Importance of Software Weaknesses, Journal of the Korea Institute of Information Security & Cryptology, 22 (6), Dec. 2012

[12] Jiho Bang, Rhan Ha, Evaluation Methology of Diagnostic Tool for Security Weakness of e-GOV Software, The Journal of the Korean Institute of Communication Sciences, 38(4), Apr. 2013

[13] Yanghwan Park, Minkyung Kim, Policy of Secure Coding for Secure e-Government Software Development (전자정부 소프트웨어의 보안성 강화를 위한 개발보안 제도 연구), Review of KIISC, 26(1), Feb. 2016

[14] Kilho Lee, Information Security Enhancement Focusing On Secure Coding, Proceedings of the KIISE Winter Conference, Dec. 2016

[15] Sukjin Kang, Jinyoung Choi, A Study on the Spread of Inspection Tools for the Secure Coding Culture, Proceedings of the KIISE Winter Conference, Dec. 2016

[17]FindSecurityBugs,https://find-sec-bugs.github.io

[18] LAPSE+, https://www.owasp.org/index.php/OWASP_LAPSE_PROJECT

[19] SonarQube, https://sonarqube.org

이 재 훈
2019년 한동대학교 전산전자공학부(학사)
2019년~현재 POSTECH 컴퓨터공학과 석사과정

최 한 솔
2018년 한동대학교 전산전자공학부(학사)
2018년 ~ 현재 한동대학교 전산전자공학부 석사과정

홍 신
2007년 KAIST 전산학부(학사)
2010년 KAIST 전산학부(석사)
2015년 KAIST 전산학부(박사)
2016년 ~ 현재 한동대학교 전산전자공학부 조교수

동의어 치환을 이용한 심층 신경망 모델의 테스트 데이터 생성†

(Generating Test Data for Deep Neural Network Model using Synonym Replacement)

이 민 수 ‡ 이 찬 근 ¶
(Min-soo Lee) (Chan-gun Lee)

요 약 최근 이미지 처리 응용을 위한 심층 신경망 모델의 효과적 테스팅을 위해 해당 모델이 올바르게 예측하지 못하는 코너 케이스에 해당하는 행동을 보이는 데이터를 자동 생성하는 연구가 활발히 진행되고 있다. 본 논문은 문장 분류 심층 신경망 모델에 기반하고 있는 버그 담당자 자동 배정 시스템의 테스트를 위해 입력 데이터인 버그 리포트의 내용에서 임의의 단어를 선택해 동의어로 변형하는 테스트 데이터 생성 기법을 제안한다. 그리고 제안하는 테스트 데이터 생성 기법을 사용한 경우와 기존의 차이 유발 테스트 데이터 생성 기법을 사용했을 경우를 다양한 뉴런 기반 커버리지를 중심으로 비교 평가한다.

키워드 : 심층 신경망, 테스팅, 코너 케이스, 테스트 데이터 생성

Abstract Recently, in order to effectively test deep neural network model for image processing application, researches have actively conducted to automatically generate data in corner-case that is not correctly predicted by the model. This paper proposes test data generation method that selects arbitrary words from input of system and transforms them into synonyms in order to test the bug reporter automatic assignment system based on sentence classification deep neural network model. In addition, we compare and evaluate the case of using proposed test data generation and the case of using existing difference-inducing test data generations based on various neuron coverages.

Key words : Deep Neural Network, testing, corner-case, test data generation

1. 서 론

우리의 주변 일상생활의 응용은 물론 안전 크리티컬 시스템에서도 심층 신경망 모델을 이용하여 개발하는 사례가 근래에 많아졌다. 일반적으로 심층 신경망 모델은 복잡도가 높고 확률을 기반으로 하는 비결정적(non-deterministic) 시스템이다. 이러한 특성에 의해 심층 신경망 모델 기반의 시스템에 임의의 입력을 가했을 경우 시스템이 응답해야 하는 안전한 반응의 범위를 보장하기 힘들다. 따라서 테스팅(Testing)에 기반한 접근 방법 [4]과 정형 기법에 기반한 접근 방법 [5] 모두 심층 신경망을 다루기 위한 많은 연구가 진행되고 있다.

최근 소프트웨어 테스트 분야에서 화이트박스 테스팅(White-box testing)을 위해 심층 신경망이 예측하지 못하는 코너 케이스(corner case) 영역에 해당하는 데이터를 찾는 연구가 활발히 진행되었다. 기존의 연구는 대부분 이미지 기반 학습 모델의 테스트 데이터인 이미지를 변형시키는 접근을 시도하였다.

본 논문은 이미지 데이터를 기반으로 하고 있는 기존 연구와는 달리 텍스트 데이터를 기반으로 하고 있는 심층 신경망 모델에 대해 동의어 치환을 통해 해당 모델을 코너 케이스에 위치하도록 테스트 데이터를 변형시키는 방법을 제안한다.

본 논문의 구성은 다음과 같다. 2절에서는 관련 연구를 소개하고, 3절에서는 배경지식을 설명한다. 4절에서는 본

† 본 연구는 한국연구재단 기초연구사업 (과제번호: NRF-2017R1E1A1A01075803)과 한국원자력연구원 원자력 ICT 안전성 검증체계 구축 및 운영과제 (과제번호: 524320-18)의 지원을 받았음.

‡ 학생회원 : 중앙대학교 소프트웨어학부
als950901@naver.com
¶ 종신회원 : 중앙대학교 소프트웨어학부 교수
cglee@cau.ac.kr

논문접수 : 2019년 1월 4일
(Received 4 January 2019)
심사완료 : 2019년 1월 12일
(Revised 12 January 2019)

논문에서 제안하는 텍스트 데이터 변형 방법에 대해 소개하며, 5절에서는 제안한 변형 방법에 대한 평가 방법과 실험 결과를 다룬다. 마지막으로 6절에서 본 연구의 결론을 소개한다.

2. 관련 연구

심층 신경망 모델의 잘못된 예측을 유도하는, 코너 케이스 영역에 해당하는 행동을 보이는 테스트 데이터를 찾기 위한 선행 연구는 [1, 2]가 있다. [1]은 심층 신경망 모델을 위한 테스트 커버리지(Test Coverage) 개념의 필요성을 제기하고 뉴런 커버리지(Neuron Coverage)의 개념을 제안하였다. 또한 동일한 응용 목적을 갖는 두 개 이상의 심층 신경망 모델이 있을 때 같은 입력에 대해 모델 간 출력의 차이를 유발하는 테스트 데이터 생성(Difference Inducing Test Data Generation)을 통해 데이터 변형 알고리즘을 제안하였다. 제안된 알고리즘은 변형된 테스트 데이터가 학습 모델이 기존 학습 데이터를 처리하기 위해 사용한 뉴런들 이외의 뉴런을 사용케 하여 뉴런 커버리지를 높이는 전략을 사용한다. [2]에서는 다양한 뉴런 커버리지와 차이 유발 테스트 데이터 생성 방법을 제안하였다. 자세한 내용은 3절에서 설명한다.

3. 배경 지식

테스트 데이터를 학습된 심층 신경망 모델에 입력했을 때, 신경망의 뉴런에서 출력되는 결과 값을 기준으로 뉴런의 행동을 두 가지 부류로 나눌 수 있다. 첫 번째는 결과 값이 예상되는 범위에 속할 경우이며, 이것을 메인 케이스(main case) 영역에 위치해 있다고 한다. 두 번째는 결과 값이 예상되지 않는 범위에 속할 경우이며, 이것을 코너 케이스 영역에 위치해 있다고 하며, 본 논문에서 목표로 두고 있는 것이 비교적 많은 뉴런이 코너 케이스 영역에 해당하는 행동을 보이는 데이터를 찾는 것이다.

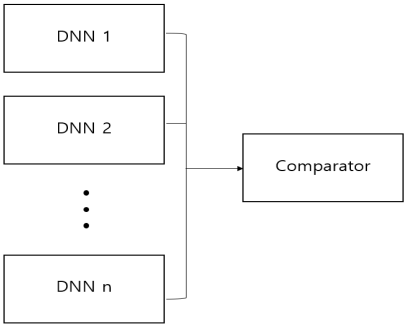


그림 34 DNN의 결과 비교

심층 신경망 모델이 예측하지 못하는 테스트 데이터를 찾는 과정은 먼저 같은 학습 데이터로 학습한 다수의 같은 구조의 모델이 필요하다. 이는 그림 1과 같이 같은 심층 신경망 모델들이 테스트 데이터 입력에 따라 나오는 결과 값이 서로 다른 경우를 예측이 빗나간 경우로 정의하기 위함이다. 다음은 테스트 데이터를 변형시키는 과정으로써 3.1절에서는 뉴런 커버리지, 3.2절에서는 차이를 유발하는 테스트 데이터 생성을 자세히 소개한다. 그리고 3.3절에서는 본 논문의 대상 시스템인 버그 담당자 자동 배정 시스템에 대해서 소개한다.

3.1 뉴런 커버리지

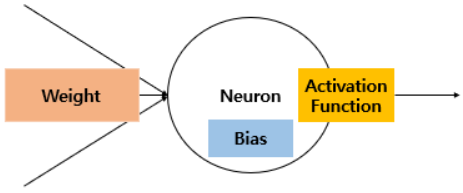


그림 35 뉴런의 구조

그림 2에서 볼 수 있듯이 일반적으로 심층 신경망에서의 뉴런은 입력 데이터, 가중치(Weight)와 바이어스(Bias)를 이용하여, 그들을 활성화 함수(Activation Function)을 통해 결과 값을 도출하는 구조를 가지고 있다.

학습된 심층 신경망 모델의 경우 가중치와 바이어스는 고정되어 오직 입력 데이터만 뉴런의 결과 값에 영향을 줄 수 있다. 입력 데이터에 의해 뉴런이 반응할 때, 즉 결과 값이 달라질 때 뉴런이 활성화되었다고 말한다.

$$Coverage = ActivatedNeuron\ /\ \ (1)$$

수식 1은 [1]에서 제안한 뉴런 커버리지에 대한 것이며, 이는 전체 뉴런에 대한 활성화된 뉴런의 비율을 나타낸다. 뉴런 커버리지의 값이 높을수록 심층 신경망 모델은 입력된 데이터를 예측 분류하기 위해 더 많은 뉴런을 활성화시킨다는 것을 의미하며, 이는 해당 모델의 예측 가능한 입력의 범위를 대략적으로 나타내기도 한다.

심층 신경망의 잘못된 예측을 유도하는, 코너 케이스 영역에 해당하는 행동을 보이는 테스트 데이터를 찾기 위해 뉴런 커버리지를 높이는 데이터를 찾는다. 이는 학습 시 모델에서 사용하지 않았던 뉴런을 사용하는 데이터가 예상과는 다른 행동을 보일 가능성이 크기 때문이다.

본 논문에서 사용할 기존 뉴런 커버리지의 종류는 Threshold Neuron Coverage(TNC) [1]와 K-Multi section Neuron Coverage(KMNC), Strong Neuron Boundary Coverage(SNBC) [2]이다. 이들은 서로 뉴런의 활성화를 판단하는 기준을 다르게 하였다. TNC는 단순히 뉴런의 활성화 판단을 뉴런의 결과 값이 미리 설정된 임계 값보다 큰 것으로 결정짓는다. 다시 말해 뉴런의

결과 값이 임계 값보다 크다면 활성화된 뉴런에 추가 시킨다. KMNC는 심층 신경망 모델을 학습 시 얻어지는 뉴런들의 결과 값을 사용하여 뉴런마다 최댓값과 최솟값으로 범위를 설정하고 그 범위를 K개로 나눈다. 주어진 입력 데이터에 의해 뉴런의 결과 값이 나눠진 구간에 속한다면, 그 구간이 활성화 되었다고 간주한다. 따라서 KMNC는 다른 뉴런 커버리지와 다르게 수식 2와 같이 계산할 수 있다.

$$KMNC = ActivatedSection / (K * \quad (2)$$

SNBC는 심층 신경망 모델을 학습 시 얻어지는 뉴런들의 결과 값을 이용하여 뉴런마다 최댓값을 설정한다. 뉴런의 결과 값이 얻어진 최댓값보다 크다면 해당 뉴런이 활성화되었다고 간주한다.

KMNC는 메인 케이스 영역에 해당하는 행동에 대한 커버리지이고, SNBC는 코너 케이스 영역에 해당하는 행동에 대한 커버리지라 할 수 있다.

3.2 차이 유발 테스트 데이터 생성

차이 유발 테스트 데이터 생성은 기존 테스트 데이터를 변형하며 심층 신경망 모델이 예측하지 못하는 데이터를 생성하는 과정이다.

먼저 테스트 데이터가 차이를 유발하는지 판단하기 위해 같은 학습 데이터로 학습된 복수개의 심층 신경망 모델을 준비한다. 설정된 데이터 변형 방법으로 입력 데이터를 변형 시키고, 변형시킨 데이터를 다시 준비된 모델들의 입력으로 가한다. 가해진 입력 데이터에 대한 모델의 뉴런 커버리지와 모델의 비용(cost)를 계산하고, 뉴런 커버리지와 비용을 같이 증가시키는 방향으로 테스트 데이터 변형을 반복한다. 이때 뉴런 커버리지와 비용을 증가시키는 방향이란, 변형된 테스트 데이터로 인한 모델의 최종 비용과 활성화 되지 않은 뉴런의 출력 값을 최대로 하는 방향을 뜻한다. 즉, 수식 3을 최대화 하는 것이다. 본 논문에서 ω는 [1]에서 사용한 값 0.1을 사용한다.

$$f(x) = \text{최종비용} + w * \text{비활성화뉴런의결과값} \quad (3)$$

변형 반복 과정 중 심층 신경망 모델들의 예측이 엇갈리면 생성을 종료한다. 이 일련의 과정은 모든 테스트 데이터에 대해 적용되며, 변형은 설정된 반복 수 만큼 적용된다.

본 논문에서 제안하는 방법과 비교하기 위해 사용할 기존의 차이 유발 테스트 데이터 생성 방법은 Fast Gradient Sign Method(FGSM) [2], Gradient Ascent Method(GAM) [1]이다. 이들은 변형하는 과정 중 모델의 비용을 이용하는 방법에 차이를 두었다.

$$x^* = x + \varepsilon * sign(f(x)) \quad (4)$$

각 데이터 변형 방법은 다음과 같다. FGSM은 수식 3을 이용한 수식 4로 변형을 진행한다. 수식 4에서 x는 현재 테스트 데이터를 의미하며 x^*는 변형된 테스트 데이터를 의미한다. GAM은 단순히 수식 3으로 변형을 진행한다.

3.3 버그 담당자 자동 배정 시스템

버그 관리 프로세스에서 새로이 보고된 버그를 해결하기 위해 적합한 담당자를 찾는 과정을 버그 선별(Triage)이라 한다. 우리는 본 논문에서 제안하는 변형 방법을 버그 담당자 자동 배정 시스템 [3]에 적용한다. 해당 시스템은 심층 신경망을 이용하여 기존의 버그 리포트를 해결한 개발자 패턴을 학습하고 새로운 버그 리포트가 주어졌을 때 적절한 개발자를 추천하게 한다. 버그 리포트는 보고자의 이름, 버그에 대한 설명(bug description), 버그 발생 환경, 버그의 심각도, 스택 추적(stack trace) 등으로 이루어져있다. 버그 담당자 자동 배정 시스템에서 사용하는 심층 신경망은 버그 리포트 구성 요소 중 버그에 대한 설명을 입력으로 사용한다.

버그에 대한 설명을 심층 신경망의 입력으로 사용하기 위해 워드 임베딩(Word embedding)을 진행한다. 워드 임베딩을 수행하기 전 특수용어와 불용어(stop words)을 제거하여 버그에 대한 설명에서 의미 있는 단어로만 구성하였다. 워드 임베딩의 학습 방법으로 Word2Vec을 사용했으며, 중심 단어로부터 주변 단어를 예측하는 Skip-gram 방식을 사용했다.

심층 신경망의 학습은 입력 데이터인 버그 리포트, 버그에 대한 설명은 해당 버그를 고친 개발자의 아이디로 라벨링(labeling)하여 지도 학습(supervised learning)방식으로 진행했다. 학습과 테스트에 사용할 데이터는 오픈소스 프로젝트인 Eclipse JDT의 2013년 2월부터 2015년 2월까지의 기간의 버그 리포트 1340개를 사용하였다. 해당 리포트에 나타난 담당자의 수는 16명이다.

본 논문에서는 [3]에서 실험한 기계학습 알고리즘 중 가장 높은 성능을 보인 CNN 모델을 대상으로 적용하였다.

4. 접근 방법

본 절에서는 텍스트 데이터에 적합한 테스트 데이터 변형 방법에 대해 제안한다. [1, 2]에서 제시한 알고리즘과 뉴런 커버리지를 사용하되, 명암 변경, 가림막 등 이미지 데이터를 기반으로 진행되던 변형 방법을 텍스트 데이터에 맞게 변경하였다.

텍스트 데이터의 변형을 위해 버그 리포트에 나타난 단어들을 사전적 의미가 같은 동의어로 변형시키는 방법으로 진행한다. 변경된 단어는 워드 벡터 테이블(word vector table)에 존재하지 않을 수 있기 때문에, 우리는 GoogleNews-vectors-negative300을 사용하였다.

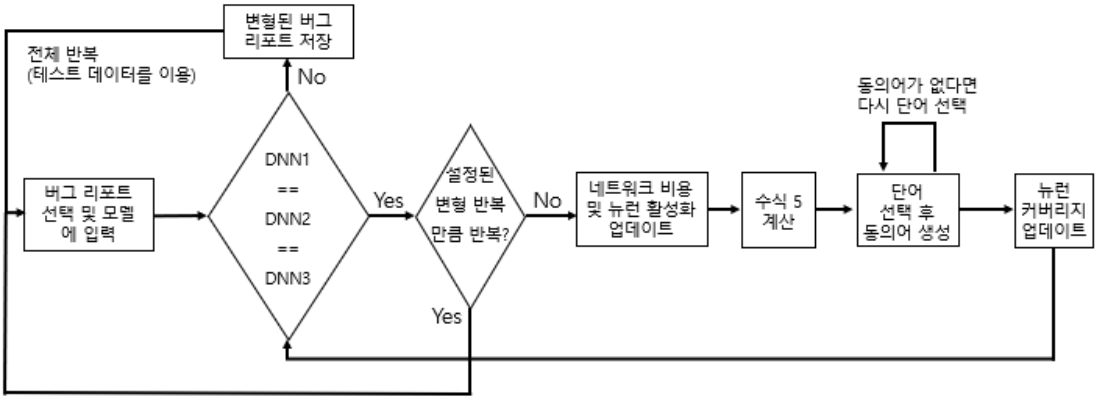


그림 36 차이 유발 테스트 데이터 생성 전체 알고리즘

$$x^* = x + \lfloor \max(L(W,\beta)) \rfloor \quad (5)$$

수식 5에서 x는 변형하기 전 데이터, x^*는 변형한 후 테스트 데이터, $L(W,\beta)$은 x를 모델의 입력으로 가했을 때 얻어지는 최종 비용을 의미한다. 해당 수식은 심층 신경망 모델의 비용을 이용한 테스트 데이터의 변형에 이용된다. 우리는 우선 모델을 잘못된 예측으로 유도해야 하고, 이미지 데이터에 비해 많은 변형이 필수로 요구되므로 $L(W,\beta)$에서 최대인 값을 이용했다. 또한 단어인 텍스트 데이터를 변형시키기 때문에, $L(W,\beta)$에서 최대인 값을 바닥(floor) 함수를 이용해 정수로 만들었다. 최종적으로 생성된 정수만큼 단어들을 선택해 동의어로 변형시킨다. 이렇게 변형된 단어들은 심층 신경망 모델로 하여금 입력 버그 리포트에 대해 적합하지 않은 담당자를 배정하도록 유도한다. 전체적인 알고리즘은 그림 3과 같다.

5. 평가 방법 및 실험 결과

5.1 평가 방법

본 논문은 텍스트 기반의 심층 신경망 모델이 예측하지 못하는 데이터를 찾는 과정에서 기존 데이터를 변형시키는 방법을 제안했다.

따라서 본 논문이 제안한 방법이 기존 방법에 비해 효율적인지 평가하기 위해 먼저 TNC, KMNC, SNBC [1, 2]와 같은 뉴런 커버리지 수치를 이용해 기존 이미지를 대상으로 한 차이 유발 테스트 데이터 생성인 FGSM, GAM과 비교하여 보여주는 것으로 평가한다. 그리고 설정하는 변형 반복과 전체 반복을 다르게 하여 반복수가 변형에 어떠한 영향을 주는지 SNBC를 이용하여 실험했다.

5.2 실험 결과

먼저 본 논문에서 제안한 데이터 변형 방식과 기존의 데이터 변형 방식을 TNC, KMNC, SNBC를 이용하여 비교한다. 본 논문에서 제안한 방법은 “Proposed”라 표기한다.

표 1 차이 유발 데이터 생성 방법 비교

	TNC	KMNC	SNBC
GAM	0.949	0.947	0.139
FGSM	0.948	0.948	0.165
Proposed	0.951	0.949	0.217

표 1은 세 개의 생성 기법을 적용해서 데이터를 만들었을 때 다양한 뉴런 커버리지 측정값을 나타낸다. 이는 변형 반복을 10회로 두고, 전체 반복을 30회로 두었을 때 실험 결과이다.

KMNC는 메인 케이스에 해당하는 모델의 행동을 계산하는 뉴런 커버리지이다. 표를 보면 제안한 방법과 기존의 방법으로 생성한 데이터가 대략 95%p 정도로 학습된 심층 신경망 모델이 예상할 수 있는 범주, 메인 케이스를 커버했다고 볼 수 있다.

SNBC는 코너 케이스에 해당하는 모델의 행동을 계산하는 것인데, 제안한 방법이 GAM, FGSM과 같은 기존 방법보다 대략 3~7%p 정도 차이를 보이고 있다. 이는 제안한 방법이 데이터를 생성하는 동안 기존 방법보다 학습된 심층 신경망의 뉴런의 3~7%p 가 코너 케이스에 해당하는 결과 값을 도출했다는 것을 의미한다. 즉, 비교적 더 많은 예상하지 못하는 범주에 속한 데이터를 생성했다고 말할 수 있다.

```
the jdt lexer Lashkar-e-Taiba hexadecimal floatingpoint misprint without for exemplar give_birth
valid misprint javac correctly report_card error misprint error hexadecimal numbers_pool
mustiness desegregate least matchless hexadecimal finger doubling the jdt give similar for malformed
literals missing exponent jdt gives error message invalid float literal digits xp00 eclipse testslexnum10f
test java2 xp00 error x0 p number
```

```
the jdt lexer Lashkar-e-Taiba hexadecimal floatingpoint misprint without for exemplar give_birth
valid misprint javac correctly report_card mistake misprint mistake hexadecimal numbers_pool
mustiness desegregate least matchless hexadecimal feel double the jdt give similar for malformed
literals missing exponent jdt gives error message invalid float literal digits xp00 eclipse testslexnum10f
test java2 xp00 error x0 p number
```

그림 4 버그 리포트의 원본과 변형 결과 1

```
if plugin provides file possible enable tracing plugin via
general tracing preference this bug covers adding plugin
preference page make easier debug problems runtime without restarting options
page org eclipse jdt ui eclipse
```

```
if plugin supply file potential enable trace plugin via
cosmopolitan trace predilection this bug screen add plugin
predilection page brand easy debug problem runtime without restart option
page org eclipse jdt ui eclipse
```

그림 5 버그 리포트의 원본과 변형 결과 2

표 2 변형 반복과 전체 반복 횟수 변화에 따른 SNBC 값

변형 반복 / 전체 반복	5	10	15	20
10	0.065	0.073	0.098	0.123
30	0.145	0.182	0.196	0.217
50	0.235	0.252	0.259	0.263

표 2는 변형 반복수와 전체 반복수를 조정하면서 제안한 방법을 적용할 때 측정된 SNBC의 값을 비교한 실험 결과이다. 첫 번째 열은 변형 반복수, 첫 번째 행은 전체 반복수를 의미한다.

전체적으로 변형 반복이나 전체 반복수를 늘렸을 때 SNBC는 증가하고 있으며, 이는 더욱 많은 코너 케이스 영역에 해당하는 행동을 보이는 데이터가 생성되었다는 것을 의미한다. 변형 반복수를 늘렸을 때와 전체 반복수를 늘렸을 때를 비교해 보았을 때, 비교적 전체 반복수를 늘려 다양한 데이터를 변형을 가한 것이 변형 반복수를 늘려 데이터에 많은 변형을 가한 것보다 SNBC 증가폭이 높다는 것을 알 수 있다.

그림 4, 5는 본 논문에서 제안하는 방법으로 심층 신경망 모델에 입력 데이터인 버그 리포트를 변형한 결과이다. 그림의 상단 버그 리포트는 원본이고, 하단 버그 리포트는 변형된 것이다. 또한 변형된 단어는 빨간색 박스로 표시하였다. 그림 4는 심층 신경망 모델 3개가 각각 12 , 14 , 14번째 담당자로 다른 예측을 했으며, 그림 5는 각각 8 , 8 , 0번째 담당자로 다른 예측을 했다.

6. 결론

심층 신경망 기술에 대한 수요가 많아지면서 검증의 필요성이 부각되어 많은 모델의 차이를 유발시키는 데이터를 생성하는 연구가 진행되었다. 이미지를 변형하여 심층 신경망이 예측하지 못하는 데이터를 생성하는 기존 연구와는 달리 본 연구는 텍스트를 변형하여 예측하지 못하는 데이터를 생성한다. 제안한 방법은 사용하는 심층 신경망 모델의 입력인 버그 리포트의 유의미한 단어를 동의어로 바꾸는 것으로 심층 신경망 모델이 예측하지 못하는, 그리고 코너 케이스 영역에 해당하는 행동을 보이는 데이터를 생성하는 것이다.

기존 연구의 데이터 생성 방법보다 본 연구에서 제안한 것이 뉴런 커버리지의 관점에서 약 3~7%p 좋은 성능을 보이는 것을 확인할 수 있었다. 또한 데이터의 변형 반복수, 그리고 전체 반복수를 조정하며 실험하여 전체 반복이 변형 반복보다 SNBC 증가폭이 더 높다는 것을 알 수 있었으며, 반복수와 뉴런 커버리지가 어느 정도 비례관계에 있다는 것을 알 수 있었다.

참 고 문 헌

[1] K. Pei, Y. Cao, J. Yang, S. Jana, “DeepXplore: automated whitebox testing of deep learning systems,” In Proc. of SOSP, ACM, pp. 1-18, Oct, 2017.

[2] Lei Ma, Felix Juefei-Xu, Jiyuan Sun, Chunyang Chen, Ting Su, Fuyuan Zhang, Minhui Xue, Bo Li, Li Li, Yang Liu, Jianjun Zhao, and Yadong Wang, “Deepguage: Comprehensive and multi-granularity testing criteria for gauging the robustness of deep learning systems,” In Proc. of ASE, pp. 120-13, 2018.

[3] 박해성, 김수빈, 이찬근, 채병훈, 딥러닝 모델 기반의 버그 담당자 자동 배정 시스템 성능 측정, 한국 소프트웨어 공학 학술대회 2018

[4] Houssem Ben Braiek, Foutse Khomh, "On Testing Machine Learning Programs,", CoRR, abs/1812.02257, 2018, URL: https://arxiv.org/abs/1812.02257

[5] Francesco Leofante, Nina Narodytska, Luca Pulina, Armando Tacchella, "Automated Verification of Neural Networks: Advances, Challenges and Perspectives," CoRR, abs/1805.09938, 2018, URL: http://arxiv.org/abs/1805.09938.

이 민 수
2019년 중앙대학교 컴퓨터공학부 학사
2019년~현재 중앙대학교 컴퓨터 공학과 석사과정.

이 찬 근
정보과학회논문지
제 44 권 제 3 호 참조

논문지 논문 모집 (Call for Papers)

한국정보과학회 소프트웨어공학 소사이어티에서는 매년 4회에 걸쳐 '소프트웨어공학 소사이어티 논문지'를 발간하고 있습니다. 이 논문지에는 소프트웨어공학 전반에 걸친 연구 성과 및 산업계 동향을 소개하고 있습니다. 이에 다음과 같은 소프트웨어공학 주제에 관련된 논문을 모집하고 있으니 학계와 산업계의 여러분들의 적극적인 논문투고를 바랍니다.

◈ 논문 주제

○ 소프트웨어 설계, 아키텍처 및 프로덕트라인
○ 요구공학
○ 소프트웨어 품질 및 테스트
○ 프로젝트 관리 및 프로세스
○ 소프트웨어 정형 기법 및 검증
○ 임베디드, 모바일, 웹기반 소프트웨어 개발
○ 기타 소프트웨어 응용 (국방, 자동차, 의료, 조선, IoT, 빅데이터 등의 분야)

◈ 논문심사

○ 투고된 논문은 편집위원회에서 심사 선정하며, 필요 시 외부 심사위원을 위촉하여 심사를 합니다. 제출된 논문은 반환하지 않습니다.
○ 심사료 및 게재료: 없음

◈ 논문 제출

○ 한국정보과학회 논문지 투고 양식(www.kiise.or.kr)을 사용하며, 논문의 분량은 10장으로 제한합니다.
○ 논문지 투고규정에 따라 작성된 심사용 논문파일과 함께 논문제목, 저자, 소속, 초록을 편집위원장 또는 편집위원에게 이메일로 송부하시기 바랍니다.

◈ 문의처 (편집위원회)

○ 편집위원장 : 이선아 교수 (경상대학교, 055-772-1377, saleese@gnu.ac.kr)
○ 편집이사 : 배경민 교수 (POSTECH 054-279-2256, kmbae@postech.ac.kr)
○ 편집이사 : 홍 신 교수 (한동대학교, 054-260-1409, hongshin@handong.edu)

투 고 요 령

1. 소프트웨어공학소사이어티 논문지에 실리는 원고는 주제 논문, 일반 논문, 산업체 기고 등으로 구분하며 다음과 같은 분야에 대하여 모집한다.

   가. 소프트웨어공학 및 그 응용분야에 대한 연구결과
   나. 강좌 및 관련 교육사항 소개 (목적, 과정, 일정, 대상, 특징)
   다. 소프트웨어 도구 및 방법론 소개 (가격, 특징, 종류, 적용사례)
   라. 소프트웨어 산업에 대한 학계, 업계의 주요 관심사
   마. 기타 관련 사항

2. 투고자는 원칙적으로 본 소사이어티의 회원으로 한다. 다만 공동 또는 초청 기고자는 예외로 한다.

3. 논문은 원칙적으로 한글로 작성한다.

4. 원고는 한글(hwp), 워드(MS Word), PDF 형식 중 하나를 택하여 작성하며, 그림과 표를 포함하여 10쪽 이내로 한다.

5. 논문 내용에 직접 관련이 있는 문헌에 대해서는 이들 문헌에 관련이 있는 본문 중에 참고 문헌 번호를 쓰고 그 문헌을 참고문헌 난에 인용 순서대로 기술한다. 참고문헌은 학술지의 경우 저자, 제목, 학술지명, 권, 호, 쪽수, 발행 연도의 순서로, 단행본은 저자, 서명, 쪽수, 발행처, 발행 연도의 순서로 기술한다.

6. 작성된 논문은 논문파일과 함께 논문제목, 저자, 소속, 초록을 편집위원장 또는 특집 주제 담당 편집위원에게 이메일로 제출한다.

7. 기타 자세한 사항은 한국정보과학회 논문지 투고 요령을 따른다.

소프트웨어공학소사이어티 논문지 제28권 제1호 (통권 102호)

발 행 일 ‖ 2019년 6월 30일

발 행 인 ‖ 이병정

편 집 인 ‖ 이선아

발 행 처 ‖ 사단법인 한국정보과학회 소프트웨어공학소사이어티

연 락 처 ‖ 서울특별시 동대문구 서울시립대로 163(전농동) 서울시립대학교
정보기술관 202호 컴퓨터과학부 이병정
전 화 : 02-6490-2451, 팩 스 : 02-6490-2444
홈페이지 : http://www.sigsoft.or.kr/

Journal of Software Engineering Society

VOLUME 28, NUMBER 1, June 2019

Korean Institute of Information Scientists and Engineers

Software Engineering Society