정보처리학회논문지 A
제19-A권 제4호(통권 제139호) 2012년 8월

The KIPS Transactions : Part A

VOL. 19-A, NO. 4, (SERIAL NUMBER139), August 2012

KIPS Korea Information Processing Society

제19-A권 제4호(통권 제139호) 2012년 8월

사단법인 한국정보처리학회

정보처리학회논문지 A

The KIPS Transactions : Part A

VOL. 19-A, NO. 4, August 2012

사단법인 한국정보처리학회
KIPS Korea Information Processing Society

http://dx.doi.org/10.3745/KIPSTA.2012.19A.4.187

유해 사이트를 접속하는 안드로이드 앱을 문자열 분석으로 검사하는 시스템

최 광 훈[†] · 고 광 만[††] · 박 희 완[†††] · 윤 종 희[††††]

요 약

안드로이드 기반 스마트폰 앱의 바이너리 코드를 오프라인 상에서 분석하여 유해 사이트 목록에 포함된 서버에 접속하는지 여부를 판단하는 시스템을 제안하고, 실제 앱에 대해 적용한 실험 결과를 제시한다. 주어진 앱의 바이너리 코드를 Java 바이트 코드로 역 컴파일하고, 문자열 분석을 적용하여 프로그램에서 사용하는 모든 문자열 집합을 계산한 다음, 유해 매체물을 제공하는 사이트 URL을 포함하는지 확인하는 방법이다. 이 시스템은 앱을 실행하지 않고 배포 단계에서 검사할 수 있고 앱 마켓 관리에서 유해 사이트를 접속하는 앱을 분류하는 작업을 자동화할 수 있는 장점이 있다. DNS 서버를 이용하거나 스마트폰에 모니터링 모듈을 설치하여 차단하는 기존 방법들과 서로 다른 단계에서 유해 앱을 차단함으로써 상호 보완할 수 있는 방법이 될 수 있다.

키워드 : 안드로이드 앱, 자바 프로그래밍언어, 달빅 바이트코드, 문자열 분석, 프로그램 분석, 컴파일러

A String Analysis based System for Classifying Android Apps Accessing Harmful Sites

Kwanghoon Choi[†] · Kwangman Ko[††] · Heewan Park[†††] · Jonghee Youn[††††]

ABSTRACT

This paper proposes a string analysis based system for classifying Android Apps that may access so called harmful sites, and shows an experiment result for real Android apps on the market. The system first transforms Android App binary codes into Java byte codes, it performs string analysis to compute a set of strings at all program points, and it classifies the Android App as bad ones if the computed set contains URLs that are classified because the sites provide inappropriate contents. In the proposed approach, the system performs such a classification in the stage of distribution before installing and executing the Apps. Furthermore, the system is suitable for the automatic management of Android Apps in the market. The proposed system can be combined with the existing methods using DNS servers or monitoring modules to identify harmful Android apps better in different stages.

Keywords : Android App, Java, Dalvik Bytecode, String Analysis, Program Analysis, Compiler

1. 서 론

최근 스마트폰이 대중화되면서 다양한 종류의 앱이 개발되고 있다. 스마트폰 앱을 통해서 제조사가 개발한 기본 기능에 사용자가 원하는 개별 부가 기능을 자유롭게 추가할 수 있다.

유해 콘텐츠를 제공하는 앱도 동시에 확산되고 있어 이 앱을 통해 청소년들이 유해 매체물에 노출되는 부작용이 발생하고 있다. 여성가족부에서 실시한 중고생 15,000여 명을 대상으로 하는 설문 조사에 따르면 휴대폰을 통해 성인물을 본 학생 수 비율은 2009년 7.3%에서 2010년 7.5%였지만, 국내에서 스마트폰이 대중화되는 시점인 2011년에 12.3%로 큰 폭으로 증가하였다[1].

인터넷이 대중화되던 2000년대 초중반에도 유사한 문제가 발생한바 있다. 유해 콘텐츠를 제공하는 앱 대신 유해 콘텐츠를 제공하는 웹 사이트가 증가하고 있었고, 스마트폰 대신 인터넷에 접속 가능한 개인용 컴퓨터를 통해 쉽게 유해 웹 사이트를 접속할 수 있었다. 앞서 설명한 문제와 유사한, 청소년들이 유해 매체물에 노출되는 부작용이 발생한 바 있다.

※ 이 논문은 2012년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No.2012-0004546).
† 정 회 원 : 연세대학교 컴퓨터정보통신공학부 조교수
†† 종신회원 : 상지대학교 컴퓨터정보공학부 교수
††† 정 회 원 : 한라대학교 정보통신방송공학부 조교수
†††† 정 회 원 : 강릉원주대학교 컴퓨터공학과 강의전담교수
논문접수 : 2012년 4월 9일
수 정 일 : 1차 2012년 6월 7일
심사완료 : 2012년 7월 16일
* Corresponding Author : Jonghee Youn(jhyoun@gwnu.ac.kr)

개인용 컴퓨터를 통해 유해 웹 사이트에 접근하는 것을 막는 방법으로는 크게 2가지가 사용되고 있다. 첫째, 인터넷 서비스 업체(ISP, Internet Service Provider)에서 운영하는 DNS 서버를 통한 특정 사이트를 차단하는 방법이다. 이 방법의 장점은 DNS 서버에서 유해 사이트 목록을 일단 구축해놓으면 해당 인터넷 서비스 업체를 사용하는 모든 개인용 컴퓨터에 쉽게 적용 가능하다. 단점은 개인용 컴퓨터에서 다른 DNS 서버를 지정해서 우회하는 방법으로 유해 사이트에 접속이 가능한 점이다.

둘째, 개인용 컴퓨터에 인터넷 접속을 실시간으로 모니터링하는 프로그램을 설치함으로써 이 컴퓨터의 사용자가 유해 사이트에 접속하는 것을 방지하는 방법이다. DNS 기반 방법의 약점을 보완할 수 있는 장점이 있으나 모든 컴퓨터에 이 프로그램을 설치해야하는 관리 문제가 있다. 또한 인터넷 사용 모니터링 프로그램을 임의로 삭제하지 못하도록 막는 방법도 고려해야 한다.

스마트폰 앱을 통한 유해 사이트 접속을 막기 위해 서로 유사한 방식으로 대응하고 있다. 첫째, DNS 서버를 통한 유해 사이트 차단 방법은 스마트폰 앱 환경에서도 개인용 컴퓨터 환경과 동일하게 적용할 수 있다. 인터넷 유해 사이트 접속을 방지하기 위해 구축한 유해 사이트 목록을 그대로 활용할 수 있다. 하지만 스마트폰에서 사용하는 DNS 서버를 변경하여 유해사이트로 우회 접속하는 것을 막지 못하는 단점이 여전히 남아 있다.

둘째, 스마트폰에 인터넷 사용을 실시간으로 모니터링하는 모듈을 설치하는 방법이다 [2,3]. 개인용 컴퓨터 환경과 달리 스마트폰 환경에서 이 방법을 사용하기에는 두 가지 제약점이 있다.

- 기술적인 관점: 유해 사이트 목록을 저장하기 위한 저장 용량 문제와 실시간 모니터링을 위해 필요한 CPU와 네트워크 장치 사용으로 인한 전력 소모를 고려해야 한다. 유해 사이트 목록을 제공하는 지정된 서버로 부터 목록을 받아 로컬 저장소에 저장할 수 있지만 개인용 컴퓨터에 비해 그 용량이 제한적이다. 서버에 접속하여 유해 사이트 여부를 확인하는 경우 속도 저하로 인해 스마트폰의 사용성에 영향을 준다.
- 개발 및 배포 관점: 개인용 컴퓨터의 운영체제와 달리 스마트폰 운영체제의 경우 보안 등의 이유로 시스템과 연관된 모듈을 통합하기에 상대적으로 제한적이다. 스마트폰에서 인터넷 모니터링 앱을 구현하기 위해 스마트폰 운영체제와의 밀접한 연동이 필요하고 이를 위해 일반 앱 개발을 위해 제공되는 API 외의 기능을 지원하는 스마트폰 제조사의 협력이 필요하다. 그리고 실시간 모니터링 모듈이 사용자에 의해 임의로 조작되지 않도록 보안이 필요하다.

본 논문에서 Android 기반 스마트폰 앱의 바이너리 코드를 오프라인 상에서 분석하여 유해 사이트 목록에 포함된 서버에 접속 가능성 여부를 판단하는 시스템을 제안한다. 연구의 독창성과 필요성은 다음과 같다.

첫째, 앱을 스마트폰에 설치하기 전, 즉 사전 배포 단계에서 유해 사이트에 접속하는 앱을 차단할 수 있다. 제안하는 시스템은 앱을 스마트폰에 설치해서 실행하지 않고 컴파일러 기술을 통해 앱의 바이너리 코드를 분석하는 정적 방법을 사용한다. DNS 서버를 사용하거나 실시간 모니터링을 통한 기존의 동적 방법과의 큰 차이점이다. 사전 배포 단계에서 앱을 차단하는 방법은 실행 단계에서 앱을 차단하는 기존의 동적 방법들을 상호 보완할 수 있을 것이다.

둘째, 제안한 시스템을 활용하면 Android 기반 앱의 유해 사이트를 접근하는 앱 여부를 판단하는 검수 과정을 자동화할 수 있다. 구글이 운영하는 안드로이드 마켓뿐만 아니라 통신 사업자들이 운영하는 T스토어, Olleh마켓, U+ 앱마켓과 제조사들이 운영하는 Samsung Apps, LG SmartWorld와 같은 앱스토어의 관리자가 본 연구에서 제안한 시스템을 활용하여 유해 사이트에 접속하는 안드로이드 앱들을 자동으로 필터링하여 검수 업무를 도울 수 있다.

기존의 동적 방법으로 검수 과정을 자동화하려면 스마트폰에 앱을 설치해서 앱을 실행하는 과정을 자동화해야하는데 마치 사용자가 앱을 사용하는 것처럼 흉내 내어 앱의 화면을 전환하고 버튼을 누르거나 텍스트를 입력하는 과정을 자동화하는 것은 다른 기술적 문제를 야기한다.

세째, 서두에서 설명한 유해 사이트를 접속하는 Android 앱을 검출하기 위해, 정적 분석 중 한가지인 문자열 분석(String Analysis) 방법을 Android 앱 바이너리 코드에 적용하여 URL 문자열을 찾는 아이디어를 처음으로 도입하였다.

문자열 분석이란 프로그램에 나타난 문자열 타입 (Java의 경우 String 타입)의 모든 식(Expression)에 대해 프로그램을 실행했을 때 각 식의 계산 결과로 나올 수 있는 모든 가능한 문자열들의 집합을 구하는 프로그램 분석 방법이다 [6,10,11].

Java 언어로 작성된 Android 앱에서 네트워크를 통해 서버에 접속하려면 반드시 URL 클래스 객체를 만들어야 한다. 이때 문자열 타입 (String)의 사이트 주소를 지정하도록 라이브러리가 설계되어 있다. 이러한 특징으로 인해 문자열 분석을 적용하면 Android 앱에서 접속할 수 있는 모든 사이트 주소들을 구할 수 있다.

지금까지 문자열 분석은 프로그램에 의해 동적으로 생성되는 SQL 문이 유효한지를 프로그램을 실행하지 않고 검사하기 위해 사용되어왔다. 예를 들어, 아래의 Java 프로그램

```
public void printAddresses(int id)
{
   Connection c =
      DriverManager.getConnection("st.db");
   String q = "SELECT * FROM address";
   if (id!=0) q = q + "WHERE studentid=" + id;
   ResultSet rs =
      c.createStatement().executeQuery(q);
   while(rs.next())
      System.out.println(rs.getString("addr"));
}
```

[6]은 학생 고유 번호를 입력 받아 데이터베이스 관리 시스템에 주소를 요청한다.

위와 같이 동적으로 질의문을 만드는 경우 Java 프로그램을 컴파일할 때는 에러가 발생하지 않지만 질의문을 실행 중에 에러가 발생한다. 왜냐하면 "address"와 "WHERE" 사이에 공백이 없어서 구문 에러인 질의문을 생성하기 때문이다. 문자열 분석 방법을 통해 이러한 에러를 실행하지 않고 미리 검출할 수 있다. 이 외에도 웹 문서, XML 쿼리문(XPath), JavaScript 식 등을 동적으로 생성하는 프로그램의 안전성을 검사하기 위한 방법으로도 사용할 수 있다.

네째, 제안한 시스템은 기존의 동적 방법과 함께 사용하여 서로 보완하는 방법으로 사용할 수 있다.

- DNS 서버 기반 유해사이트 차단 방법은 쉽게 우회할 수 있는 문제점이 있고,
- 실시간 모니터링 기반 방법을 리소스 제한적인 스마트폰 환경에 적용하려면 기술적 어려움이 많고, 개발을 위해 스마트폰 운영체제를 탑재하는 제조사의 협력이 필수적이며, 개개인의 스마트폰에 모니터링 모듈을 설치해야하는 운영상의 어려움이 있다.
- 동적 방법은 일단 앱을 스마트폰에 설치해야 적용 가능하고, 앱 마켓에 업로드된 많은 앱을 반복해서 테스트하는 과정을 자동화하기 어렵다.

다섯째, 분석 대상으로 Android 기반 앱을 선택한 이유는 Android 앱 마켓의 현재 운영 방식이 모든 개발자가 특별한 앱 검수 절차 없이 자유롭게 앱을 등록할 수 있기 때문에 논문에서 고려하는 성격의 유해 앱이 많기 때문이다. iOS 기반 앱의 경우 앱스토어에 등록하기 위해 내부 검수 절차를 반드시 통과해야하므로 유해 앱을 배포 단계에서 차단할 수 있다.

본 논문에서 제안한 시스템을 Android 마켓에서 다운로드 받은 실제 앱에 적용하여 실험하였다. Android 앱의 Dalvik 바이트코드를 먼저 동일한 의미의 Java 바이트코드로 변환하고 그 다음 기존의 문자열 분석 방법을 변환된 코드에 적용하여 잠재적으로 접속할 수 있는 모든 유해 사이트 목록을 찾았다.

Dalvik 바이트코드에 문자열 분석 방법을 직접 적용하지 않은 이유는 기존 Java 바이트코드에 대한 풍부한 분석 도구들을 활용하기 위해서이다.

실험 결과 18개의 유해 사이트를 접속하는 앱들 중 11개를 자동으로 검출할 수 있었다. 나머지 7개의 앱에 대해 유해성 여부를 판단하지 못한 이유는 크게 두 가지로 분류할 수 있다.

- 첫째, Android 앱이 사실상 HTML과 JavaScript를 실행하는 웹 기반으로 구성되어 있어 JavaScript 프로그램에서 유해 사이트를 접속하는 형태이다. 현재 시스템은 Dalvik 바이트코드에서 변환된 Java 바이트코드만을 분석하도록 구성되어 있는데 JavaScript 코드에 문자열 분석을 적용하도록 확장함으로써 보완 가능하다.
- 둘째, 기존의 Java 바이트코드 분석 도구들을 활용하기 위해 Dalvik 바이트코드를 Java 바이트코드로 변환하는 역컴파일러를 사용하도록 시스템을 구성하였으나 역컴파일러의 버그로 인해 문자열 분석을 아예 적용하지 못한 사례가 발생하였다. 이 문제는 Dalvik 바이트코드 수준에서 직접 문자열 분석을 적용하도록 시스템을 구성함으로써 해결 가능하다.

실험 결과를 통해 Android 앱 마켓에서 유해 사이트를 접속하는 앱을 검수하는 과정을 자동화할 수 있음을 확인하였다. 현재 구축한 시스템을 활용하면 적어도 50%이상의 유해 앱을 자동으로 분류할 수 있었다.

앱 마켓에 새로 업로드되는 앱의 개수가 앞으로 지속적으로 증가할 것이고, 기존의 앱에 대해서도 버전이 올라가면서 관리 대상 앱의 수는 끊임없이 증가할 것이다. 따라서 제안한 시스템과 같이 앱의 검수 과정을 자동화하는 방법이 반드시 필요할 것이다. 그리고 실험을 통해 드러난 두 가지 문제점들을 앞서 기술한 바와 같이 개선한다면 더 높은 비율의 유해 앱을 자동으로 분류할 수 있을 것으로 판단된다.

2장에서 문자열 분석 기반 유해 안드로이드 앱 검출 방법을 설명하고, 3장에서 실험 결과를 요약하고, 4장에서 제안한 방법의 한계와 개선 방향을 논의하고, 5장에서 논문의 결론을 내린다.

2. 문자열 분석으로 유해 사이트를 접속하는 안드로이드 앱을 검사하는 방법

본 논문에서 관심을 갖는 안드로이드 앱의 유해성을 아래와 같이 가정한다.

- 안드로이드 앱을 실행했을 때 유해 매체물을 제공하는 서버 사이트에 접속하면 이 앱은 유해하다고 정의한다.
- 유해 매체물을 제공하는 사이트의 목록은 주어져있다고 가정한다.

첫 번째 항목과 같이 안드로이드 앱의 유해성을 정의한 이유는, 스마트폰에서 제한적인 리소스를 활용하는 특성상 유해 매체물을 앱의 로컬 데이터에 모두 포함시키지는 않고 지정된 서버를 접속하여 필요한 데이터를 다운로드 받는 형태일 것으로 판단되기 때문이다.

두 번째 항목은 논문에서 제공하는 유해 사이트를 접속하는 안드로이드 앱 검사 방법이 특정 목록에 의존하지 않음을 기술하고 있다. 따라서 기존의 인터넷 서비스 공급자가 DNS 기반 유해 사이트 접근 차단 방법에 사용하던 목록을 활용한다고 가정한다.

스마트폰 앱처럼 다운로드 받아 실행하는 코드의 유해성을 논의할 때 다양한 다른 형태의 유해성이 있다. 예를 들어, 스마트폰의 개인 정보나 위치 정보를 허가 받지 않고 외부로 유출하는 앱도 유해하다고 분류할 수 있다[13, 14].

이런 종류의 유해성을 검출하는 주제는 본 논문의 범위를 벗어난다.

안드로이드 앱의 유해성을 분석하는 과정은 그림 1과 같다. 본 논문에서 사용한 분석 방법은 4단계로 구성되어 있다.

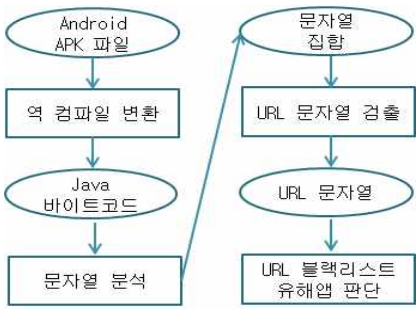


그림 1. 안드로이드 앱 유해성 분석 과정
Fig. 1. An Analysis Process for Detecting Android Apps Accessing Harmful Sites

- Dalvik 바이트 코드로 구성된 안드로이드 앱을 Java 바이트 코드로 역 컴파일 한다.
- 요약 해석 (abstract interpretation)으로 Java 바이트코드에서 다루는 문자열 타입의 변수가 갖을 수 있는 모든 가능한 문자열의 집합을 계산한다.
- URL을 표현하는 정규식 (regular expression) 패턴으로 Java 바이트코드에서 URL 문자열을 찾는다.
- 주어진 유해 사이트 URL 블랙 리스트에 검출한 URL이 포함되어 있는지 확인하여 앱의 유해성을 판단한다.

2.1 Java 바이트코드로의 변환

첫 번째 단계는 Java 바이트 코드에 대한 분석 도구를 활용하기 위해 안드로이드 앱의 Dalvik 바이트 코드를 역 컴파일 변환하는 과정이다.

안드로이드 앱은 APK (Android application package) 형식의 파일이다. 압축 파일 형식 ZIP과 동일하며 바이너리 코드, 리소스, XML 파일 등을 포함한다. 바이너리 코드는 Dalvik 바이트 코드 (Dex 코드)로 작성되어 있다. 안드로이드 플랫폼은 달빅가상기계 (Dalvik Virtual machine)에서 Dalvik 바이트 코드로 작성된 명령어를 실행하는 방식으로 앱을 실행한다[4]. Dalvik 바이트 코드는 레지스터 기반 명령어 집합으로 구성되어 있고 상수 풀 (constant pool)을 최적화하는 특징을 지닌다.

오픈 소스 dex2jar [5,15] 역 컴파일러를 사용하여 APK 파일에 포함된 Dalvik 바이트 코드 (classes.dex)를 동일한 의미의 Java 바이트 코드 (classes.jar)로 변환한다.

사실 Dalvik 바이트 코드를 직접 분석하는 방법도 가능하지만, Java 바이트 코드를 분석하는 기존의 다양한 도구를 활용할 수 있는 장점이 있다. 반면에 사용한 역 컴파일러의 버그로 인해 문자열 분석을 적용하지 못하는 문제점이 발생하기도 하였다. 4장에서 역 컴파일 과정 도입에 대한 장단점을 논의 한다.

2.2 문자열 집합 계산

두 번째 단계에서 변환된 Java 바이트 코드로부터 실행 중 잠재적으로 사용할 수 있는 모든 문자열 집합을 계산한다. Java 바이트 코드에 대한 요약 해석 기반 문자열 분석 방법은 프로그램에 나타난 java.lang.String 타입의 모든 식에서 정규식을 구한다. 이 정규식으로 표현된 문자열 집합은 해당 식을 실행한 결과 얻은 문자열을 모두 포함한다[6]. 주어진 식에서 구한 정규식으로부터 원하는 패턴의 문자열이 포함되어있는지 여부를 확인할 수 있다.

문자열 분석 방법을 설명하기 위해 다음의 Java 프로그램 예제를 살펴보자.

```
public String foo(int x) {
  /* Line 1: */  StringBuffer b =
                    new StringBuffer("I ate");
  /* Line 2: */  if (x > 0) b.append(x);
  /* Line 3: */  else b.append("no");
  /* Line 4: */  b.append(" apple today");
  /* Line 5: */  return b.toString();
}
```

이 프로그램에서 5개의 문장에서 문자열 타입의 식을 각각 포함하고 있다. 참고로 StringBuffer 타입은 변경 가능한 String 타입을 표현하고, 각 문장에 포함된 변수 b에 문자열이 저장될 수 있다.

- 분석을 시작하기 전에 변수 b를 공집합 ({ })으로 초기화하고 각 Java 문장의 의미를 고려하여 변수 b가 어떻게 변경될지를 계산한다.
- Line 1에서 b에 저장될 수 있는 문자열 집합은 { “I ate” }이다.
- Line 2와 3에서 조건식을 결정하는 변수 x의 값은 실행 전에 알 수 없으므로 참이 되는 경우와 거짓이 되는 경우를 모두 고려한다. 따라서 Line 2에서 b에 저장되는 문자열 집합은 { “I ate” <INT> }가 되고, Line 3에서 { “I ateno” }가 된다. 이때 <INT>는 임의의 정수로부터 변환된 문자열을 뜻한다.
- Line 4에서 b에 저장되는 문자열 집합은 { “I ate” <INT> “apple today”, “I atenoapple today” }가 된다.
- 마지막으로 Line 5에서 b에 저장되는 문자열 집합은 Line 4에서의 b에 저장되는 문자열 집합과 동일하다.

이 문자열 집합을 동일한 의미를 갖는 정규식으로 아래와 같이 표현할 수 있다.

```
"I ate" (<INT> | "no") " apple today"
```

즉, “I ate” 문자열이 먼저 나타나고 숫자가 나타나거나 “no” 문자열이 출현한 다음 “ apple today”로 끝난다. <INT>는 “17” 또는 “-3”과 같은 정수에 대한 문자열을 뜻한다.

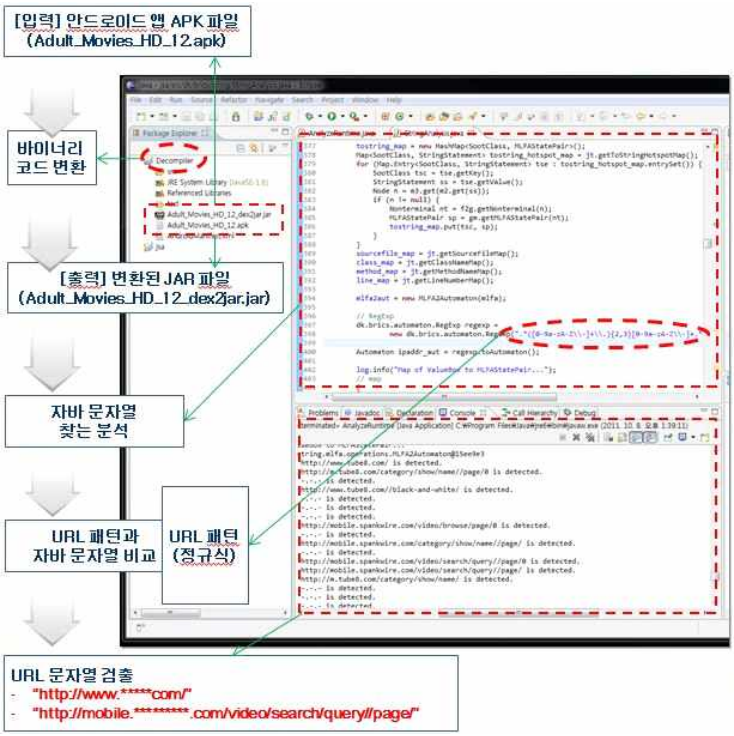


그림 2. 안드로이드 유해 앱 분석 과정 예
Fig. 2. An Example of the Android App Analysis

문자열 분석 과정을 설명하기 위해 Java 프로그램의 예를 들었지만 제안한 시스템에서는 Java 바이트코드를 분석한다.

2.3 URL 문자열 필터링

세 번째 단계는 Java 바이트 코드에서 문자열 타입을 갖는 각각의 식으로부터 구한 정규식을 참고해서 URL 패턴의 문자열이 포함될 수 있는지 비교한다.

Java 언어로 작성된 Android 앱에서 네트워크를 통해 서버에 접속하려면 반드시 URL 클래스 객체를 만들어야 한다. 이때 문자열 타입 (String)의 사이트 주소를 지정하도록 라이브러리가 설계되어 있다. 이러한 특징을 활용하여 문자열 분석으로 찾은 문자열 집합에 URL 패턴의 문자열이 있는지 확인함으로써 Android 앱에서 접속하는 모든 사이트 주소들을 구할 수 있다.

구현에서 사용한 정규식 URL 패턴은 IETF RFC 1738 규격[7]을 참고하여 아래와 같이 정의한다.

```
.*([0-9a-zA-Z\-]+\.){2,3}[0-9a-zA-Z\-]+.*
```

이 정규식이 표현하는 집합에 속하는 문자열은 점(.)으로 분리되는 숫자, 대소문자 알파벳, 바(-)로 이루어진 임의의 단어를 적어도 3개나 4개를 포함한다. 예를 들어, “http://www.yonsei.ac.kr:1234”, “192.168.0.1”, “ftp://cair-archive.kaist.ac.kr/public/”, “www.naver.com”와 같다.

앞서 문자열 분석 방법을 설명하기 위해 예로 든 Java 프로그램으로부터 계산한 b에 저장되는 문자열 집합을 표현하는 정규식과 URL 패턴을 기술하는 정규식은 서로 공통 문자열을 가지고 있지 않음을 쉽게 확인할 수 있다.

정규식으로 표현된 두 문자열 집합들의 공통 문자열을 효율적으로 찾기 위해 정규식을 DFA (Deterministic Finite Automata, 유한오토마타)로 변환한 다음 두 DFA의 교집합이 되는 DFA를 구하여 이 DFA가 받아들이는 문자열 집합이 공집합인지 확인한다 [6].

2.4 블랙리스트 기반 유해 URL 판단

네 번째 단계는 안드로이드 앱으로부터 검출한 URL을 유해 사이트 주소 목록에 포함되어 있는지를 기준으로 분석한 앱의 유해성을 판단한다. 이때 사용하는 유해 사이트 주소 목록은 주어져 있다고 가정한다. 기존의 인터넷 사업자에서 유해 사이트 접속을 차단하기 위해 구축한 DNS 서버에서 사용한 유해 사이트 주소 목록을 그대로 활용한다고 가정한다.

그림 2는 본 논문에서 제안한 방법을 구현하여 샘플 안드로이드 앱에 적용하여 URL을 검출하는 예이다.

3. 실험 결과

이 장에서 31개의 샘플 안드로이드 앱을 수집하여 제안한 분석 시스템에 적용한 실험 결과를 제시한다. 먼저 실험에서 사용한 샘플 앱의 속성을 요약한다.

- 전체 샘플들 중 유해 안드로이드 앱은 18개
- 앱의 클래스 수는 평균 334개 (최소 4개~최대 1,718개)
- 전체 앱은 10,340개의 클래스 포함
- 변환된 Java 바이트 코드 (Jar 파일)의 평균 크기는 425.7KByte (최소 4KB ~ 최대 2,115KB)

유해성 여부는 수동으로 앱을 실행하여 살펴본 결과에 따라 판단하였고 앱이 접근하는 사이트 주소를 확인한 다음 이 실험을 위해 유해 사이트 블랙 리스트를 구축하였다. 따라서 제안한 시스템에서 해당 사이트 주소를 검출한다면 유해 앱으로 분류할 수 있다.

3.1 문자열 분석 결과

본 논문에서 구축한 시스템에서 사용한 문자열 분석 프로그램 [6]은 클래스를 기본 단위로 분석한다. 안드로이드 앱(APK 파일)로부터 역 컴파일한 Java 바이트 코드 (JAR 파일)에 일반적으로 여러 개의 클래스가 포함되어 있으므로 각 클래스 별로 문자열 분석을 적용한다. 예를 들어 샘플 앱으로 선택한 Melon.apk 파일을 분석한 결과를 표 1에서 보여준다.

표 1. 샘플 앱에 대한 문자열 분석 결과 예
Table 1. An Example of the String Analysis for a Sample Android App

클래스	검출된 문자열
com.iloen.melon.MusicBrowserActivity	• http://www.melon.com/intro1.jsp • android.intent.action.VIEW
com.iloen.melon.R$attr	없음
com.iloen.melon.R$drawable	없음
com.iloen.melon.R$string	없음
com.iloen.melon.R	없음

문자열 분석 프로그램에 의해 발견한 URL로 유해 안드로이드 앱을 가려내는 정확성에 대해 실험하였다. 전체 샘플 안드로이드 앱들 중 유해 앱은 18개이다. 앱을 직접 사용해본 결과를 근거로 분류하였다.

- 문자열 분석으로 찾은 유해 앱의 수는 11개
- 유해성을 판단하지 못한 앱의 수는 7개

문자열 분석으로 유해 앱을 찾지 못한 경우를 분석한 결과 안드로이드 앱이지만 사실상 HTML/JavaScript기반 웹 응용프로그램으로 구성되어 있어 URL을 검출하지 못하는 경우를 발견하였다. 문자열 분석을 Java 바이트 코드에 대해 적용하는 것뿐만 아니라 JavaScript 코드를 포함하는 리소스 파일도 함께 분석하는 방법을 고려해볼 수 있다.

구현한 시스템에서 문자열 분석으로 검출된 문자열의 특성을 살펴보면 Java 바이트 코드에 포함된 하나의 문자열 상수에 URL 문자열이 나타나는 형태이다. 문자열 연산을 사용하여 작은 문자열들을 조합해서 URL 문자열을 만드는 경우를 실험에서 사용한 안드로이드 앱 샘플들에서는 찾지 못했다. 이와 같은 사실을 확인하기 위해 Dalvik 바이트코드에서 상수 문자열을 추출하는 도구를 Dalvik 바이트코드 디스어셈블러(Smali) [9]로 구현하여 실험에 사용한 안드로이드 앱에 적용한 결과 문자열 분석으로 검출한 URL을 동일하게 검출할 수 있음을 확인할 수 있었다.

3.2 유해 사이트를 접근하는 안드로이드 앱 검출에 소요된 시간

문자열 분석 기반 유해 안드로이드 앱 검출 분석에 걸린 시간은 전체 31개의 APK 파일을 분석하는데 대략 8시간 소요되었다.

- 전체 31개 APK 분석 시간은 28,761초
- 1개 클래스 당 분석 시간은 평균 2.8초
- 1개 APK를 분석하는 평균 시간은 927.8초

실험 환경은 인텔 코어 i3 CPU 550 3.2GHz, 윈도우즈 7 운영체제, Cygwin DLL version 1.7.8이다.

실험 결과 검출한 문자열들 중 유해 사이트로 분류되지 않은 URL 문자열이 여러 샘플 앱에서 빈번하게 검출되기도 하였다. 예를 들어 모바일 광고 서버 사이트를 가리키는 URL 문자열이 여러 개의 안드로이드 앱들에서 공통적으로 다수 검출되었다. 빈번하게 나타나는 화이트 리스트에 속하는 URL 문자열을 유해 콘텐츠를 제공하는 서버의 블랙리스트와 비교한 결과를 캐쉬로 보관해서 나중에 그 URL 문자열이 다시 검출되면 캐쉬를 통해 효율적으로 블랙 리스트에 포함 여부를 판단하는 방법도 고려해 볼 수 있다.

URL 패턴에 네트워크 사이트를 가리키는 URL 뿐만 아니라 안드로이드에서 특별하게 사용하는 문자열들로서 매치되어 검출되는 사례가 빈번했다. 안드로이드 인텐트 이름, 허가 이름, 콘텐트 프로바이더 컴포넌트 이름은 URL 형태를 지니고 있다. 예를 들어, “content://com.android.htmlfileprovider”는 Android 응용 프로그램 간 데이터를 공유하기 위해 제공하는 컴포넌트의 이름이다. 이와 같은 Android에서 사용하는 특별한 유형의 문자열을 적절히 필터링함으로써 문자열 분석으로 검출한 불필요한 문자열들의 수를 상당히 줄일 수 있을 것이다.

4. 논의 사항

4.1 문자열 분석과 문자열 추출 방법의 비교

문자열 분석은 단순 상수 문자열뿐만 아니라 접합 연산 (+, concatenation), 추가 연산 (append), 대체 연산 (replace), 제어 흐름을 모두 고려하여 가능한 문자열의 집합을 계산할 수 있다.

실험에서 사용한 샘플 안드로이드 앱을 분석한 결과 얻은 URL 문자열은 모두 Dalvik 바이트 코드에 그대로 포함된 단순 상수 문자열로부터 비롯된 것이었다. 따라서 문자열 분석에서 처리할 수 있는 다양한 연산을 고려한 복잡한 방

법이 사실상 필요하지 않았다. 물론 Android 앱내에서 URL 문자열을 “http” + “://” + “www.goo” + “gle.com”과 같은 식을 통해서 만들어 사용한다면 단순 문자열 상수를 추출하는 방법을 통해서 위의 URL 문자열을 검출할 수 없다.

단순 문자열 추출은 프로그램을 한 번만 스캔하면 되지만, 문자열 분석 방법은 (특히 프로그램 상에 반복문이 포함된 경우에) 최종 분석 결과를 얻을 때까지 반복하는 과정이 포함되어 있어 분석 시간이 훨씬 오래 걸린다. 실제 시스템을 구축할 때 이러한 점을 고려해서 문자열 분석과 단순 문자열 추출을 혼용해서 사용하여 효율과 분석도의 정확성을 조절해볼 수 있다.

4.2 Dalvik 바이트코드를 Java 바이트코드로 변환

본 논문에서 제안한 시스템은 Dalvik 바이트코드를 Java 바이트 코드로 변환하여 문자열 분석을 적용하는 형태로 설계하였다. 이렇게 설계한 이유는 문자열 분석 프로그램이 Java 바이트 코드를 분석하는 도구 (Soot [12])를 사용하기 때문이다. Soot는 Java 바이트코드를 다루기 위한 중간 언어를 제공하고 최적화하기 적절한 레지스터 기반 변형된 Java 바이트코드와 다양한 최적화 방법을 제공한다.

Dalvik 바이트코드를 직접 다루는 문자열 분석을 설계한다면 역 컴파일 과정이 불필요할 것이다. 하지만 Dalvik 바이트코드와 Java 바이트코드는 상당히 많은 공통점을 지니고 있으므로 Java 바이트코드를 다루기 위해 개발된 도구를 모두 버리고 Dalvik 바이트코드 용으로 다시 개발하는 것 또한 바람직하지 않다. 이러한 관점에서 Dalvik 바이트코드를 Java 바이트코드로 안정되게 역 컴파일하는 방법이 중요하다.

문자열 분석 과정 중 에러가 발생한 경우의 원인을 분석하였다. 샘플 안드로이드 앱에 포함된 전체 10,340개 클래스를 분석하는 중에 2090개 클래스에서 에러가 발생했다. 이 에러들 중 77.7%에 해당하는 1624개 에러가 Java 바이트 코드의 무결성(integrity) 문제가 발생한 것 때문이었다. 무결성 문제란, 예를 들어 Java의 원시(Primitive) 데이터를 NULL에 할당하거나, 참조 데이터를 저장하는 Java 바이트코드 명령어 (Astore)를 원시 데이터와 사용하거나, 분기된 제어 흐름이 합해지는 곳에서 양쪽의 스택 타입이 다르거나, 메써드를 찾을 수 없는 등의 문제가 발생했다.

Java 바이트 코드의 무결성 문제가 발생한 원인 중 하나는 Dalvik 바이트코드를 Java 바이트 코드로 역 컴파일 하는 과정에서 비롯될 수 있다. Dalvik 바이트코드에서 Java 바이트 코드로의 역 컴파일 할 때의 고려해야할 사항이 있다[8]. 첫째, Dalvik 바이트코드 명령어에서 최적화를 위해 타입 정보를 느슨하게 사용하는 경우가 있다. 예를 들어 참조 객체 주소의 NULL과 정수 0을 32비트 0으로 동일하게 표현한다. Dalvik 바이트코드 명령어에서 사용된 0의 타입이 정수인지 클래스 타입인지를 알기 위해서 명령어의 앞뒤 문맥을 살펴서 타입을 결정해야 한다. 둘째, 레지스터 기반 Dalvik 바이트코드에서 스택 기반 Java 바이트코드로 변환할 때 스택으로 레지스터를 단순히 흉내내도록 변환을 하는 경우 불필요한 load/store 명령어가 증가하게 된다.

4.3 Android 외의 스마트폰 플랫폼에 적용

Android 외의 스마트폰 플랫폼에 제안한 시스템을 적용하기 위해 여러 가지 기술적 어려움이 있다. 첫째, 각 스마트폰 플랫폼에서 사용하는 개발 언어와 실행 환경이 달라서 하나의 도구로 여러 플래폼의 앱에 적용할 수 없다. 둘째, 일반적으로 앱을 구성하는 바이너리 코드가 ARM용 기계어인 경우 자유도로 인해 문자열 분석을 적용하기가 어렵다. Android 앱의 경우 Dalvik 가상기계에서 실행하는 모델 덕분에 실제 기계어보다 상위 레벨의 가상 기계어에 분석을 적용할 수 있었다. 예를 들어 아이폰 앱의 경우 ARM용 기계어 코드로 구성되어 있어 분석하기 쉽지 않다. 또한 iOS의 특성상 아이폰 앱의 바이너리 코드는 개발자의 고유 키로 인코딩되어 있어 다루기 어렵다.

5. 결 론

본 논문에서 Android 앱이 유해 사이트를 접속하는지 여부를 배포 단계에서 사전에 검사하는 시스템을 제안하였고, Android 마켓의 샘플 앱을 대상으로 실험한 결과를 제시하였다. 유해 사이트를 접속하는 앱을 차단하는 기존의 동적 방법들은 실행 단계에서 적용되고 논문에서 제안한 방법은 배포 단계에서 적용되므로 상호 보완할 수 있다.

첫째, 기존의 DNS 서버를 통해 유해 사이트 접속을 차단하거나 또는 스마트폰에 실시간 모니터링 모듈을 탑재하는 동적 차단 방법은 앱 배포 이후 실행 단계에서 검출 하는 방법으로 접근 방법의 큰 차이점이 있다. 제안한 방법은 기존의 동적 방법들과 달리 안드로이드 앱을 실행하지 않고 URL 검출과 블랙 리스트를 비교하는 방식이므로 배포 단계에서 차단할 수 있는 장점이 있다. 또한 사용자 휴대폰에 모니터링을 위한 모듈을 별도로 설치할 필요도 없다.

둘째, Android 스마트폰 사용의 확대로 새로운 앱도 크게 증가할 것이고 기 개발된 앱의 경우 버전 업그레이드될 것이므로 앱이 유해 사이트를 접속하는지 여부를 확인하는 작업이 수동으로 처리하기에 어려워질 것이다. 이러한 일련의 검수 작업을 자동화할 필요가 있다. 제안한 방법은 사람의 개입 없이 Android 앱을 분석할 수 있는 시스템으로 이러한 요구 사항에 적합하다. 마켓에 올린 앱을 주기적으로 다운받아 분석 도구를 적용하여 유해 앱 검수를 자동화할 수 있다.

향후 연구로 Dalvik 바이트코드를 Java 바이트코드로 안전하게 역컴파일 변환하는 방법에 대한 것과 Dalvik 바이트코드에 문자열 분석 방법을 직접 적용하는 주제가 있다. Java 바이트코드에 대한 분석 및 최적화 도구들이 많이 개발되어 왔으므로 역컴파일 변환으로 Dalvik 바이트코드를 간접적으로 분석하고 최적화할 수 있다면 Android 앱을 분석하거나 최적화하는데 큰 도움이 될 것이다. 또한 Dalvik 바이트 코드에 직접 문자열 분석 방법을 구현한다면, 역컴파일 변환 후 Java 바이트코드에 문자열 분석 방법을 적용한 방법과 서로 장단점을 비교할 수 있을 것이다.

두 번째 주제로, JavaScript 프로그램에 문자열 분석을 적용해 유해 사이트를 접속하는지 여부를 판단하도록 문자열

분석의 적용 범위를 확장하는 것이다. 스마트폰 플랫폼 별 다른 개발 환경 때문에 동일한 기능을 중복해서 개발하는 문제를 해결하기 위해 JavaScript와 HTML5를 사용한 앱 개발 방식이 주목받고 있다. 안드로이드 앱과 JavaScript 프로그램에 문자열 분석을 함께 적용하는 방법을 통해 유해 사이트 접속에 대한 판단의 정확도를 높일 수 있을 것이다.

참 고 문 헌

[1] Ministry of Gender Equality & Family, A Comprehensive Survey Report on Young People's Contact with Harmful Environment, November, 2011.

[2] Deokgi Jung, Cutoff Apparatus for URL-based Harmful Site Access in LAN Environment and Method Thereof, Pub. No.KR10-2009-0031370, 2009.

[3] Hangyeon Song and Misim Kim, System and Method for Blocking Harmfulness Equipped Blocking Application Against Harmful Website and Application, Pub. No.KR10-2010-0066841 2010.

[4] Dalvik Technical Information, http://source.android.com/tech/dalvik/.

[5] Tools to work with android .dex and java .class files, http://code.google.com/p/dex2jar/.

[6] Aske Simon Christensen, Anders Møller, and Michael I. Schwartzbach, "Precise Analysis of String Expressions," in Proceedings of 10th International Static Analysis Symposium (SAS), LNCS, Vol.2694, Springer-Verlag, June, 2003.

[7] IETF, Uniform Resource Locators, RFC 1738.

[8] Damien Octeau, William Enck, Patrick McDaniel, "The ded Decompiler," Technical Report NAS-TR-0140-2010, The Pennsylvania State University, September, 2010.

[9] Smali: an Assembler/Disassembler for Android's dex format, http://code.google.com/p/smali/.

[10] Tae-Hyoung Choi, Oukseh Lee, Hyunha Kim, and Kyung-Goo Doh, "A Practical String Analyzer by the Widening Approach," in Proceedings of the Fourth ASIAN Symposium on Programming Languages and Systems, LNCS, Vol.4279, pp.374-388, Springer, Sydney, Australia, 2006.

[11] Kyung-Goo Doh, Hyunha Kim, and David A. Schmidt, "Abstract Parsing: Static Analysis of Dynamically Generated String Output Using LR-Parsing Technology," in Proceedings of the 16th International Symposium on Static Analysis, pp.256-272, Los Angeles, CA, August, 2009.

[12] Vallee-Rai, Raja and Gagnon, Etienne and Hendren, Laurie J. and Lam, Patrick and Pominville, Patrice and Sundaresan, Vijay, "Optimizing Java Bytecode Using the Soot Framework: Is It Feasible?," in Proceedings of the 9th International Conference on Compiler Construction, pp.18-34, Springer, London, UK, 2000.

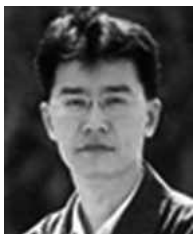
[13] William Enck, Damien Octeau, Patrick McDaniel, and Swarat Chaudhuri, "A Study of Android Application Security," in Proceedings of the 20th USENIX Conference on Security, pp.21-21, USENIX Association, Berkeley, CA, USA, 2011.

[14] Jinseong Jeon, Kristopher K. Micinski, Jeffrey A. Vaughan, Nikhilesh Reddy, Yixin Zhu, Jeffrey S. Foster, and Todd Millstein, "Dr. Android and Mr. Hide: Fine-grained Security Policies on Unmodified Android," Technical Report CS-TR-5006, Dept. of Computer Science, University of Maryland, College Park, 2011.

[15] Wontae Sim, Jong-Myoung Kim, Jae-cheol Ryou, and Bongnam Noh, "Android Application Analysis Method for Malicious Activity Detection," Journal of the Korea Institute of Information Security and Cryptology, Vol.21, No.1, February, 2011.

최 광 훈

e-mail : kwanghoon.choi@yonsei.ac.kr
1994년 한국과학기술원 전산학과(학사)
1996년 한국과학기술원 전산학과(공학석사)
2003년 한국과학기술원 전산학과(공학박사)
2003년~2005년 JAIST, Researcher
2005년~2006년 Tohoku Univ., Researcher
2006년~2010년 LG전자 Mobile Communication 연구소, 책임연구원
2010년~2011년 서강대학교 컴퓨터공학과 BK21연구교수
2011년~현 재 연세대학교 컴퓨터정보통신공학부 조교수
관심분야 : 모바일 소프트웨어, 프로그래밍언어, 컴파일러, 프로그램 분석, 소프트웨어 검증 등



고 광 만

e-mail : kkman@sangji.ac.kr
1991년 2월 원광대학교 컴퓨터공학과 (공학사)
1993년 2월 동국대학교 컴퓨터공학과 (공학석사)
1998년 2월 동국대학교 컴퓨터공학과 (공학박사)
1998년 3월~2001년 8월 광주여자대학교 컴퓨터과학과 전임강사
방문연구 : QUT(2003, 호주), UQAM(2008, 캐나다), UC Irvine(2010, 미국)
2001년 9월~현 재 상지대학교 컴퓨터정보공학부 교수
관심분야 : 프로그래밍언어론, 컴파일러, 모바일 컴퓨팅 등

박 희 완

e-mail : heewanpark@halla.ac.kr
1997년 동국대학교 컴퓨터공학과(학사)
1999년 한국과학기술원 전산학과(공학석사)
2010년 한국과학기술원 전산학과(공학박사)
2004년~2007년 삼성전자 무선사업부 책임연구원
2010년~2011년 ETRI 부설연구소 선임연구원
2011년~현 재 한라대학교 정보통신방송공학부 조교수
관심분야 : 프로그램 난독화, 역공학, 악성코드 분석, 소프트웨어 워터마킹, 정적 및 동적 분석 등

윤 종 희

e-mail : jhyoun@gwnu.ac.kr
2003년 경북대학교 전자전기공학부(학사)
2011년 서울대학교 전기컴퓨터공학부(박사)
2011년~현 재 강릉원주대학교 컴퓨터공학과 강의전담교수
관심분야 : Embedded systems, Optimizing compiler, software optimizations and computer architecture, MPSoC, Mobile Cloud Computing