문자열 분석 기반 유해 안드로이드 앱 검출

최광훈*, 박경득*, 고광만**, 박희완***, 윤종희****
*연세대학교 컴퓨터정보통신공학부 **상지대학교 컴퓨터정보공학부
***한라대학교 정보통신방송공학부 ****강릉원주대학교 컴퓨터공학과
e-mail: kwanghoon.choi@yonsei.ac.kr, qkremraos@hanmail.net,
kkman@sangji.ac.kr, heewanpark@halla.ac.kr, jhyoun@gwnu.ac.kr

Detection of Malicious Android Apps Using String Analysis

Kwanghoon Choi*, Kyeongdeuk Park*, Kwangman Ko**, Heewan Park***,
Jonghee Youn****
*Computer&Telecommunication Engineering Division, Yonsei University
**Dept. of Computing Information & Engineering, Sangji University
***School of Info. & Comm., Broadcasting Engineering, Halla University
****Dept. of CS & Engineering, Gangneung-Wonju National University

요 약

안드로이드 앱에서 접근할 수 있는 유해 사이트를 프로그램 분석 방법으로 검출하는 방법을 제안한다. 주어진 앱의 바이너리 코드를 자바바이트 코드로 역 컴파일하고 문자열 분석 방법을 적용하여 실행 중 사용 가능한 문자열 집합을 계산한 다음 유해 사이트 URL 문자열이 포함되어 있는지 확인하는 방법이다. 기존에는 앱을 직접 실행해서 특정 URL에 접속하는지 감시하는 동적 모니터링 방법인 반면, 제안한 방법은 앱을 실행할 필요가 없다. 앱스토어 관리에서 주기적으로 유해 앱 여부를 검사하는데 제안한 방법을 활용할 수 있다.

1. 서론

최근 스마트폰이 대중화되면서 다양한 종류의 앱이 개발되고 있다. 스마트폰 앱을 통해서 제조사가 개발한 기능에 사용자가 원하는 부가 기능을 자유롭게 추가하는 장점이 있다. 하지만 유해 콘텐츠를 제공하는 앱의 확산으로 청소년들이 성인물에 노출되는 역기능도 발생하고 있다 [1]. 애플 앱스토어의 경우 앱에 대한 내부 검수 절차가 마련되어 있지만, 안드로이드 마켓에는 누구나 자유롭게 앱을 올릴 수 있어 문제가 더욱 심각하다.

인터넷이 대중화되던 2000년대 초중반에도 유사한 문제가 발생했었고, 유해 웹 사이트 목록을 중앙 서버에서 관리하여 이 사이트를 방문하는 것을 동적으로 차단하는 방법으로 대응해왔다. 스마트폰의 경우도 이와 유사한 접근 방법으로 대응하고 있다 [2,3].

본 논문에서는 안드로이드 앱의 바이너리 코드를 입력받아 URL을 추출하여 유해 앱 여부를 가려내는 방법을 제안한다. 기존 방법과 차이점은 동적 모니터링을 사용하지 않고 정적 분석을 사용하는 점이다. 안드로이드 마켓, T스토어, Olleh마켓, U+ 앱마켓, Samsung Apps, LG SmartWorld와 같은 앱스토어에서 관리자가 자동으로 안드로이드 앱들을 분석할 수 있다. 기존의 동적 모니터링 방법으로는 가능하지 않다.

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 분석 기반 안드로이드 앱의 유해성 판단 방법

주어진 안드로이드 앱을 실행했을 때 유해 콘텐츠를 제공하는 외부 서버에 접속하면 이 앱은 유해하다고 이 논문에서는 정의한다. 유해 콘텐츠를 앱 내에 포함하는 경우는 드물다고 판단한다. 물론 이 정의에 포함되지 않는 종류의 유해한 앱도 있다. 예를 들어, 스마트폰의 개인 정보나 위치 정보를 외부로 유출하는 앱도 유해하다고 할 수 있지만 이런 종류의 유해성을 검출하는 주제는 본 논문의 범위를 벗어난다.

주어진 안드로이드 앱의 유해성을 분석하는 과정은 3단계로 구성되어 있다.

- 안드로이드 앱을 자바 바이트 코드로 역 컴파일 한다.
- 요약 해석 (abstract interpretation)으로 자바 바이트코드에서 다루는 문자열 타입의 변수에 가리킬 수 있는 가능한 문자열의 집합을 계산한다.
- URL을 표현하는 정규식 (regular expression) 패턴으로 자바 바이트코드에서 유해 사이트의 URL 문자열을 찾아 유해성을 판단한다.

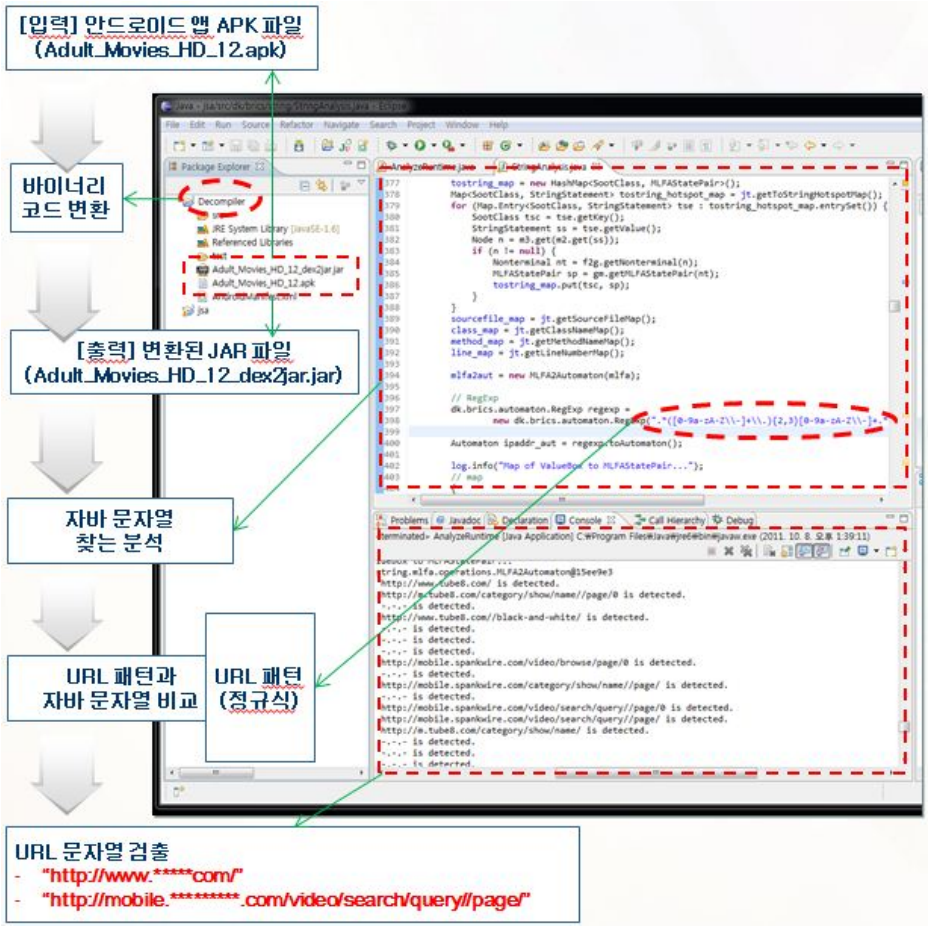

그림 1 안드로이드 유해 앱 분석 과정 예

첫 번째 단계는 역 컴파일 과정이다. 안드로이드 앱은 APK (Android application package) 형식의 파일이다. 사실 압축 파일 형식 ZIP과 동일하며 바이너리 코드, 리소스, XML 파일 등을 포함한다. 안드로이드 플랫폼은 달빅 가상기계 (Dalvik Virtual machine)를 통해 앱을 실행하는데 Dex라는 기계어로 구성되어 있다[4]. 역 컴파일 과정은 APK 파일에서 Dex 기계어 코드 (classes.dex)를 자바 바이트 코드 (classes.jar)로 변환한다. 오픈 소스로 사용 가능한 dex2jar 역 컴파일러[5]를 사용하였다.

두 번째 단계는 요약 해석 변환된 자바 바이트 코드에서 실행 중 잠재적으로 사용할 수 있는 문자열 집합을 계산한다. 자바 프로그램 (또는 자바 바이트 코드)에 대한 요약 해석 기반 문자열 분석 방법은 프로그램에 나타난 java.lang.String 타입의 모든 식으로부터 DFA (deterministic finite-state automaton)를 계산 한다[6]. 문자열 분석 방법을 설명하기 위해 다음의 자바 프로그램 예제를 살펴보자.

```
public String foo(int x) {
  StringBuffer b = new StringBuffer("I ate");
  if (x > 0) {
     b.append(x);
  } else {
     b.append("no");
  }
  b.append(" apple today");
  return b.toString();
}
```

위의 프로그램 맨 마지막 라인에서 문자열 타입을 갖는 식 b.toString()에 대해 계산된 DFA는 *"I ate" (<INT> | "no") " apple today"*이다. 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.

참고로 문자열 분석은 SQL Injection이라는 해킹을 막기 위한 방법으로 사용되었다[6].

세 번째 단계는 자바 프로그램에서 문자열 타입을 갖는 각각의 식으로부터 구한 DFA에 URL 패턴의 문자열이 포함될 수 있는지 비교한다. 구현에서 사용한 정규식 URL 패턴은 IETF RFC 1738 규격[7]을 참고하여 다음과 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 설명한 3단계 분석 과정을 주어진 안드로이드 앱에 적용해 발견한 URL이 유해 콘텐츠를 제공하는 서버의 주소인지 판단하기 위해서 별도의 URL 블랙리스트를 가정한다. 만일 리스트에 검출된 URL이 포함되어 있다면 분석 대상 안드로이드 앱에서 이 URL이 가리키는 서버에 접속할 수 있으므로 유해하다고 결론을 내린다.

안드로이드 앱에서 네트워크를 통해 외부 서버에 접속하기 위해서 WWW 웹 상의 리소스를 가리키는 URL을 표현하는 java.net.URL 클래스의 객체를 생성할 필요가 있다. URL 클래스의 객체 생성자는 서버 주소를 포함하는 문자열 타입의 매개 인자를 받는다. 따라서 문자열 분석을 통해 외부 서버에 접속하는 자바 프로그램을 효과적으로 필터링할 수 있다.

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

3. 결론

본 논문은 Dex 코드의 자바 바이트 코드로의 역 컴파일러와 자바 바이트 코드를 대상으로 하는 문자열 분석기를 조합하여 유해 콘텐츠를 제공하는 서버에 접속할 수 있는 유해 안드로이드 앱을 자동으로 찾아내는 방법을 제안하였다.

기존의 동적 모니터링에 의존하는 방법과 달리 안드로이드 앱을 실행하지 않고 URL 검출과 블랙 리스트 비교를 통해서 유해성 분석을 자동화 할 수 있다. 사용자 휴대폰에 모니터링을 위한 모듈이나 별도의 앱을 설치할 필요가 없다. 특히 앱 스토어에 제출된 앱을 관리하는데 유용하다. 본 논문에서 제안한 방법을 사용하면 앱 스토어에 올린 앱을 주기적으로 다운받아 분석 도구를 적용하여 유해 앱 검수를 자동화할 수 있다.

제안한 방법의 단점으로, 첫째 서버를 통하지 않고 유해 콘텐츠를 직접 포함하거나 자바 스크립트 또는 JNI 인터페이스를 통하는 등의 간접적인 방법을 통해 네트워크에 접근하면 이를 유해 안드로이드 앱으로 구분할 수 없다. 둘째, 유해 앱을 가려내는데 유해 서버 주속 목록이 결정적인 역할을 한다. 방대한 목록을 구축할수록 유해 앱 분석도 더 정확해질 것이다.

현재 논문에서 제안한 방법을 구현하고 수집한 샘플 유해 안드로이드 앱에 적용하여 문자열 분석을 통한 URL 검출이 효과적인지 실험을 진행 중이다. 실험 결과 검출된 문자열은 세 가지 타입으로 분류할 수 있었다. 첫째, 검출된 URL 문자열은 대부분 자바 바이트 코드에서 사용하는 문자열 상수로부터 비롯되었고 문자열 연산으로 조합되어 URL 문자열을 만드는 경우를 실험한 안드로이드 앱 샘플들에서는 찾지 못했다. 둘째, 검출된 URL 문자열 중 모바일 광고 서버를 가리키는 경우가 매우 빈번했다. 동일한 모바일 광고 서버가 여러 개의 안드로이드 앱들이 사용하는 경우도 눈에 띄었다. 마지막으로, URL 패턴에 URL 뿐만 아니라 안드로이드 인텐트 이름, 허가 이름, 콘텐트 프로바이더 컴포넌트 이름 등도 불필요하게 매치되어 검출되는 결과를 확인하였다.

본 논문에서 제안하는 방법은 Dex 코드를 자바 바이트 코드로 역 컴파일 하는 과정을 사용하는데, 이 과정에서 무결성 오류(integrity error)가 발생해서 분석을 진행하지 못하는 경우가 발생했다. Dex 코드와 자바 바이트 코드는 상당히 유사함에도 불구하고 역 컴파일의 어려움이 있다 [8]. Dex 코드를 직접 분석하거나 역 컴파일 방법의 완성도를 높이는 향후 연구를 통해 문자열 기반 유해 안드로이드 분석 방법을 개선할 수 있을 것이다.

참고문헌

[1] http://news.kbs.co.kr/society/2012/02/24/2440836.html
[2] 정덕기, 랜 환경에서의 URL 기반의 유해 사이트 접근 차단장치 및 그 방법, 출원번호 10-2009-0031370, 2009년.
[3] 송한경, 김미심, 유해차단 어플리케이션을 구비한 유해 차단 시스템 및 방법, 출원번호 10-2010-0066841, 2010년.
[4] Dalvik Technical Information, http://source.android.com/tech/dalvik/.
[5] Dex2Jar: Android Dex and Java class files manipulation tools, http://code.google.com/p/dex2jar/.
[6] Aske Simon Christensen, Anders Møller, and Michael I. Schwartzbach, “Precise Analysis of String Expressions,” Proc. 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/.