공인인증서 유출형 안드로이드 악성앱 탐지를 위한 Tainting 기법 활용 연구★

윤 한 재*, 이 만 희**

요 약

공인인증서는 개인을 증명하거나 통신간의 위변조 등을 방지하기 위하여 공인인증기관에서 발행하는 전자화된 정보로써 사이버 상의 인감도장이라 할 수 있다. 공인인증서는 암호화 된 파일의 형태로 PC 및 스마트폰에 저장되어 인터넷뱅킹 및 스마트뱅킹 서비스를 이용할 때 개인을 증명하기 위해 사용하기 때문에 외부로 유출될 경우 위험할 수 있다. 급증하는 안드로이드 기반 악성 어플리케이션 중 파일로 존재하는 공인인증서와 개인정보 등을 외부의 서버에 전송하는 악성 어플리케이션 또한 발견되고 있다. 본 논문은 공인인증서 탈취 악성코드를 사전에 판단하여 차단하기 위해 안드로이드 기반 동적 분석 도구인 DroidBox를 이용하여 공인인증서 외부 유출행위 여부를 판단하는 방안을 제안한다.

A Study on Tainting Technique for leaking official certificates Malicious App Detection in Android

Yoon Hanj Jae*, Lee Man Hee**

ABSTRACT

The certificate is electronic information issued by an accredited certification body to certify an individual or to prevent forgery and alteration between communications. Certified certificates are stored in PCs and smart phones in the form of encrypted files and are used to prove individuals when using Internet banking and smart banking services. Among the rapidly growing Android-based malicious applications are malicious apps that leak personal information, especially certificates that exist in the form of files. This paper proposes a method for judging whether malicious codes leak certificates by using DroidBox, an Android-based dynamic analysis tool.

Key words : Android, Certificates, Data leak, Android malware analysis

접수일(2018년 8월 16일), 게재확정일(2018년 9월 23일)

★ 본 논문은 2018년도 한남대학교 학술연구비 조성비 지원에 의하여 연구되었음.

* SKInfosec

** 한남대학교/컴퓨터공학과

1. 서 론

개인정보의 무분별한 수집과 적절하지 않은 관리로 인해 대규모 개인정보 유출 사건이 이어짐에 따라 우리나라는 2014년부터 개인정보보호법을 시행하고 있다. 안전행정부는 주민등록번호 수집 금지 제도 가이드라인을 통해 주민번호 대체 수단으로 i-PIN, 휴대폰, 공인인증서 등을 사용하도록 권장하고 있다[1]. 이중 공인인증서는 2014년 해외의 국내 전자상거래 활성화의 방해 요인으로 지적된 후, 2015년 금융위원회에서 인터넷뱅킹 및 인터넷 쇼핑에서 공인인증서 의무 사용이 폐지되었고 대체 인증 시스템이 도입되고 있다[2]. 그러나 이미 공인인증서를 기반 인증시스템이 잘 운영되고 있으며, 또한 일정 금액 이상은 여전히 공인인증서가 필요하므로 공인인증서를 통한 인증 서비스는 상당한 기간 동안 사용될 것으로 예상된다. 이런 상황에서 공인인증서가 주민번호 대체 수단으로 다시 권고됨에 따라 그 활용 및 보안성이 다시 주목받고 있다.

PKI(Public Key Infrastructure)를 기반으로 사용되는 공인인증서는 개인키인 SignPri.key 파일과 개인키에 대한 인증서인 SignCert.der 파일로 구성된다. 이처럼 파일로 존재하는 공인인증서는 이미 알려져 있는 디렉터리에 존재하기 때문에 접근이 쉬워 공인인증서 파일의 외부 유출의 가능성이 늘 존재한다. 공인인증서가 외부로 유출되면 전사적 대입 또는 사회 공학적 공격을 통하여 암호화되어 있는 개인키의 비밀번호를 알아낼 경우, 탈취된 개인키로 개인인증을 할 수 있어 금전적 손실이나 추가적 개인정보유출에 활용될 수 있다. 실제로 2016년 3월에는 중국 범죄조직으로부터 공인인증서를 포함한 개인정보를 받아 신용카드를 새로 발급받아 수억 원을 챙긴 사건이 발생하기도 했다[3].

이처럼 불법 탈취될 경우, 상당한 보안 위협이 존재하는 공인인증서가 악성코드를 통해 대거 유출되었다는 소식은 공인인증서를 개인 인증의 마지노선으로 생각하고 있는 우리나라 국민들에게 상당한 충격을 주었다. KISA에 의하면 2012년 8건이던 공인인증서 유출은 2013년 8,710건, 2014년 41,733건, 2015년 22,796건으로 급격히 빠른 속도로 증가하다 2016년 6월까지 6,815건으로 감소하였으나 여전히 심각한 위협이라고 할 수 있다[4].

공인인증서의 유출은 스마트뱅킹 사용이 활성화되면서 더욱 가속화될 것으로 예상된다. 실제로 국민은행 스마트 뱅킹 앱으로 위장하여 스마트폰에 저장된 공인인증서를 유출하고 사용자에게 보안카드번호 입력을 유도하는 악성 앱들이 발견되었다[5]. 따라서 스마트폰에서 공인인증서를 유출하는 앱을 사전에 분석하여 시그니처를 생성하여 스마트폰 백신에 업데이트할 수 있다면 스마트폰을 통한 공인인증서 유출을 상당히 경감할 수 있을 것으로 예상된다.

본 연구에서는 안드로이드 앱을 가상환경에서 미리 구동시키고 공인인증서를 유출하는지 탐지할 수 있는 시스템을 제안한다. 난독화 또는 암호화 등으로 정적 분석이 점점 어려워지고 있는 상황에서 동적 분석은 정적 분석과 상호보완적으로 악성 앱을 효과적으로 탐지할 수 있다. 동적 분석에서의 자료 유출 탐지를 위한 기존 연구들은 주로 행위 분석에 집중하였다[6],[7],[8],[9]. 즉, 시스템 호출의 내용 및 형태를 확인하여 해당 정보가 유출되는지 유추하거나, 특정 파일을 접근했는지 확인하는 방식이다. 하지만 이러한 행위분석의 약점은 유출 행위가 복잡하여 유출행위가 뚜렷이 드러나지 않아 미탐 또는 오탐이 발생하거나, 단순히 공인인증서 파일을 접근했다는 이유로 악성으로 탐지하는 오탐이 발생할 수 있다. 이를 근거로 백신 업데이트를 할 경우, 정상 앱을 악성으로 차단할 가능성이 있어 큰 혼란을 초래할 수 있다.

본 연구는 tainting 기법을 이용하여 미탐을 대폭 줄이는 동시에 오탐이 없는 공인인증서 유출 탐지 시스템을 구축하는 것을 목표로 한다. Tainting 기법은 관심 자료를 marking 또는 tainting을 한 후, 해당 자료가 다른 자료로 이동, 변환하더라도 비트 단위로 추적하여 최종적으로 해당 자료가 외부로 유출되는지 확인할 수 있다[10], [11], [12]. 특히 안드로이드 가상환경인 TaintDroid는 23가

지 개인정보가 유출되는지를 알려준다[13]. 하지만 TaintDroid에 추가적인 내용을 tainting하는 기법은 알려져 있지 않고 이를 이용하여 공인인증서의 유출을 탐지하는 방법은 연구된 바가 없다.

본 연구에서 제안 및 구현한 tainting 기반 공인인증서 유출 안드로이드 악성 앱 판단 시스템은 몇 가지 중요한 의미를 지닌다. 첫째, 기술적으로 TaintDroid를 수정하여 공인인증서 유출을 탐지하는 최초 연구로써 의미가 깊다. 둘째, 스마트폰에서 공인인증서 사용에 대한 우려가 높아가는 상황에서 공인인증서 유출 악성 앱을 사전에 오탐없이 판단할 수 있다. 마지막으로 본 시스템과 백신 서비스와 연동할 경우, 공인인증서 유출 악성 앱에 대한 대비를 실시간으로 수행할 수 있어 사람이 관여하지 않고도 공인인증서 유출 악성 앱에 대응할 수 있다.

2. 관련 연구

스마트폰에 있는 데이터를 외부로 유출하는 악성 어플리케이션이 증가함에 따라, 이를 정적 분석 또는 동적 분석으로 탐지하는 연구 또한 활발하게 진행되었다. 최영석 등은 LKM (Loadable Kernel Module)으로 구현된 시스템 콜 후킹 모듈을 커널에 삽입하여 시스템 콜을 분석함과 동시에 white-list를 기반으로 하여 허가되지 않은 어플리케이션의 접근 또는 IP주소에 연결을 차단함으로써 스마트폰 내부의 데이터 유출을 탐지 및 방지하는 연구를 진행하였다[14]. 하지만 이 경우, 특정 파일에 대한 접근이 필요한 정상적인 어플리케이션이 white-list에 등록되어있지 않은 경우 오탐을 발생시키고 정상적인 어플리케이션 모두를 white-list에 등록을 해야만 하는 불편함이 있으며, 공격자가 white-list에 등록되어 있는 IP 또는 DNS 등을 알고 있다면 스마트폰 내에 존재하는 민감 정보가 외부로 유출될 가능성이 있다.

김도래 등은 APK 파일을 디컴파일하여 API를 추출하고, 추출된 API간의 상호 의존성을 분석하여 개인정보 탈취 및 유출 노드사이의 최단거리를 계산하여 그 거리가 5 이하일 경우 정보 유출의 가능성을 가지고 있는 의심스러운 어플리케이션이라고 탐지하는 연구를 진행하였고, 이를 구현한 LeakDroid를 개발하여 그 효과를 증명하였다[15]. 하지만 공격자가 악성 어플리케이션을 개발할 때 개인정보 탈취 및 유출 노드사이의 거리를 고려하여 5 이상으로 개발할 경우 우회가 가능하다.

Di Cerbo 등은 APK 파일 안에 포함되어 있는 AndroidManifast.xml의 안드로이드 권한을 이용하여 개인정보를 외부로 유출하는 행위를 예측하는 연구를 진행하였고, Zhou 등은 API, 안드로이드 권한 등을 기반으로 시그니처를 생성했고 네이티브 코드와 동적 코드실행을 탐지하는 휴리스틱 탐지 기법을 제안하였으며 DroidRanger를 통하여 그 효과를 증명하였다[16][17].

위 연구 외에도 다양한 연구들이 많이 진행되고 있지만 공통적인 단점은 오탐이라고 할 수 있다. 특정 데이터를 외부로 유출하는 악성 어플리케이션을 분석 할 때 안드로이드 권한이나 API 등을 이용하여 분석할 경우, 정상적인 어플리케이션에서도 요구할 수 있는 권한이나 API로 이루어져 있기 때문에 악성 행위를 유추할 수는 있지만 오탐의 가능성이 항상 존재한다. 이 오탐 때문에 악성 앱의 자동 분석 결과는 분석가가 참조하는 자료로 쓰일 뿐, 판단의 결정적인 근거로 사용하기 어렵다. 하지만 tainting 기법을 이용한 데이터 유출 탐지는 해당 유출 행위 미발현 등으로 인한 미탐은 있을 수 있으나 오탐없는 100% 정탐이므로 유출 결과를 악성 앱 판단, 중지, 삭제에도 사용할 수 있다.

3. Tainting 기반 공인인증서 유출 탐지 시스템

본 논문에서 사용하는 TaintDroid는 가상 머신을 이용하여 안드로이드 어플리케이션을 동적 분석 할 수 있는 도구인 DroidBox를 이용한다. DroidBox는 안드로이드 4.1버전의 커널 소스에 TaintDroid 패치를 적용한 후 DroidBox만의 패치파일

을 다시 적용하여 Tainting 분석이 가능한 DroidBox를 빌드 한다. TaintDroid가 포함된 DroidBox는 Location, Contacts, MIC, Phone_number, Location_gps, Location_net, Location_last 등 총 23개의 데이터를 개인정보로 분류하고 taint한다[18].

TaintDroid가 공인인증서를 tainting 하기 위해서 세 가지를 수정해야한다. 첫째, 초기화된 안드로이드 커널은 공인인증서를 가지고 있지 않으므로 공인인증서 유출 악성코드는 특정 디렉터리에서 위치하는 공인인증서를 읽어오는 것을 실패하게 된다. 따라서 악성코드를 분석하기 전에 공인인증서를 특정 디렉터리에 삽입해야 한다. 둘째, TaintDroid 자료유출 태그는 공인인증서 항목이 정의되어 있지 않으므로 공인인증서 유출 악성코드인지 나타낼 수 없다. 이를 위한 태그를 추가해야한다. 마지막으로 수정이 필요한 부분은 TaintDroid에서 공인인증서를 읽을 때 태그를 붙이는 것이다. TaintDroid의 tainting 기능은 특정 정보에 태그 붙이기, 태그된 정보 지속적으로 따라가기, 태그된 정보 외부 유출(sink)될 때 탐지하기로 이루어지는데, 태그된 정보 따라가기와 외부 유출될 때 탐지 기능은 TaintDroid의 핵심 기능으로써 추가적으로 수정하지 않아도 모든 태그된 정보에 대해서 수행하기 때문에 태그를 하고 싶은 데이터에 태그만 붙이면 자동으로 유출 탐지가 된다.

3.1 공인인증서 삽입

정부24 홈페이지에 의하면 공인인증서는 한국증권전산만 “pkicert” 디렉터리를 사용하고 나머지 모든 인증서 발급기관은 “NPKI” 디렉터리를 사용한다. 하부 디렉터리에 “cn=본인이름” 이 생성되고 마지막으로 SignCert.der(전자서명용 인증서)와 SignPri.key(전자서명용 비밀키)가 존재하게 된다. 공인인증서 유출 악성코드 제작 해커라면 가장 일반적인 NPKI 디렉터리 하부의 인증서와 비밀키를 유출할 것이므로 일반적으로 저장되는 디렉터리에 두 파일을 생성하면 악성코드가 충분히 속을 수 있는 파일이 될 것이다. 그림 1은 이를 수행하는 공인인증서를 삽입하는 수정된 DroidBox.py 소스코드이다. DroidBox.py는 DroidBox가 분석대상 앱을 구동하기 위해 사용하는 스크립트인데, 분석대상 앱 구동 전 공인인증서 디렉터리 생성 및 파일 삽입 코드를 추가하면 을 보다 먼저 이 앱을 구동시키면 성공적으로 분석이 가능하다.

```
call(['adb', 'shell', 'mkdir', '/sdcard/NPKI/yessign/USER'])
call(['adb', 'push', '/home/DroidBox/signCert.der', '/sdcard/NPKI/yessign/USER/'])
call(['adb', 'push', '/home/DroidBox/signPri.key', '/sdcard/NPKI/yessign/USER/'])
```

(그림 1) 수정된 DroidBox.py

3.2 공인인증서 태그 선언

TaintDroid 자료유출 태그는 Location, Contacts, MIC, Phone_number, Location_gps, Location_net, Location_last 등과 같이 미리 선언된 태그를 사용하고 있다. 공인인증서 유출을 탐지하기 위해서는 해당 태그를 선언해야한다. 민감 정보를 리턴하는 메서드에서 민감 정보를 리턴하기 전에 선언한 태그를 추가하여 리턴한다. 이후 선언한 태그가 추가된 정보를 외부로 유출하면 TaintDroid는 민감 정보가 외부로 유출되었다고 판단한다. Taint.java 파일에 민감 자료를 판단하기 위한 태그들이 선언되어 있으며 그림 2와 같이 공인인증서 유출 탐지를 위한 태그 TAINT_CERTIFICATE를 추가하였다.

```
public static final int TAINT_CLEAR          = 0x00000000;
public static final int TAINT_LOCATION       = 0x00000001;
... omission ...
public static final int TAINT_CERTIFICATE
                              = 0x00800000; //test value
```

(그림 2) Taint.java에 정의된 tag

3.3 공인인증서 태그 삽입

TaintDroid가 태그 삽입에 사용하는 접근법은 Java API 또는 안드로이드 시스템의 메서드를 수정하여 접근된 데이터가 Taint 대상일 경우, 해당 데이터에 적절한 태그를 삽입한다. 태그가 삽입되

고 나면 TaintDroid의 태그 추적 기능에 의해 지속적으로 모니터링 되고 추후 파일 쓰기 또는 외부 유출시 탐지된다. 따라서 공인인증서 유출을 탐지하기 위해서는 공인인증서를 최초로 접근하여 읽어오는 메서드 또는 생성자를 찾고 적절히 수정하는 작업이 필요하다. 파일의 형태로 존재하는 공인인증서를 접근하는 방법으로 파일 읽기, 쓰기, 복사, 이동 등 다양한 행위를 수행할 수 있으므로 이런 행위를 위한 메서드 리스트가 필요하며, 표 1은 메서드와 생성자 리스트와 그 기능을 간략히 제시한다.

<표 4> 파일 접근 관련 메서드와 생성자 목록

Function	Classification
Create File Object	Constructor
Create FileInputStream Object	Constructor
Create InputStream Object	Constructor
Rename a file name	Method
Read data from a file	Method

상기 파일 접근 메서드를 수정하는 방법은 TaintDroid가 Taint를 위해서 필요한 메서드를 수정하는 방법을 사용한다. Taintdroid에서는 파일 읽기, 쓰기, 복사를 하기 위해 반드시 File 객체를 생성한 후에 InputStream 클래스를 상속받은 FileInputStream 객체를 생성하여 작업을 진행해야 한다. 본 연구에서는 Taint가 전파되도록 하기 위해 File 클래스의 생성자, File 클래스의 renameTo 메서드, FileInputStream 클래스의 생성자, InputStream 클래스의 생성자, InputStream 클래스의 read 메서드를 수정하였다.

Taint 전파의 첫 단계로 그림 3과 같이 File 객체가 생성될 때 인자 값으로 주어지는 'path'와 Taint를 추가하기 원하는 리스트를 미리 정의한 텍스트 파일을 비교하여 리스트 파일 안에 path가 존재할 경우 path에 태그를 추가하도록 하였다. 이후 다른 클래스에서 접근이 가능하도록 임의로 선언한 변수인 'publicPath' 변수에 기존의 생성자 결과 값을 동일하게 넣어주었다.

파일 복사의 다음 단계로 앞서 생성한 파일 객체에서 데이터를 읽기 위해 File 객체를 인자 값으로 사용하여 InputStream 클래스를 상속받은 FileInputStream 객체를 생성한다. 이때 그림 4와 같이 인자 값으로 사용되는 File 객체의 publicPath 변수에서 태그 값을 가져와 TAINT_CLEAR 값이 아닐 경우 FileInputStream 클래스에 임의로 선언한 'fileTaintingProp' 변수에 태그를 저장하고 이 값을 부모 클래스인 InputStream 클래스에 임의로 선언한 'fileTaintingProp' 변수에도 저장하도록 수정하였다.

```
public class File implements Serializable, Comparable<File> {
  public String publicPath = null;
  public File(String path) {
    if(!path.equals("/sdcard/list.txt")) {
      try {
        File taintingList = new File("/sdcard/list.txt");
        BufferedReader br = new BufferedReader(
          new FileReader(taintingList));
        String s;
        while((s = br.readLine()) != null) {
          Taint.log("file path : " + s);
          if(path.contains(s)) {
            Taint.addTaintString(path,
              Taint.TAINT_CERTIFICATE));
            Taint.log("directory tainting");
            break;
          }
        }
        br.close();
      } catch(Exception e) {
        Taint.log("File constructor Exception");
        e.printStackTrace();
      }  catch(StackOverflowError e){
        System.out.println("File constructor
          StackOverflowError");
        e.printStackTrace();
      }
    }
    Taint.log("path : " + path);
    Taint.log("tainting tag : " + Taint.getTaintString(path));

    this.publicPath = fixSlashes(path);
    this.path = fixSlashes(path);
  }
}
```

(그림 3) 수정된 File 클래스

마지막으로 그림 5와 같이 생성한 InputStream 클래스에 정의된 read 메서드를 이용하여 데이터를 읽을 때 FileInputStream 클래스의 생성자가 호출될 때 InputStream 클래스의 변수에 저장한 태그를 검사하여 TAINT_CLEAR 값이 아니라면 변수가 저장하고 있는 태그 값을 리턴 값인 buffer에 추가하도록 수정하였다.

```
public class FileInputStream extends InputStream implements Cl
oseable {
  public int fileTaintingProp = Taint.TAINT_CLEAR;
  public FileInputStream(File file) throws
    FileNotFoundException {
    if (file == null) {
      throw new NullPointerException("file == null");
    }
    this.fd = IoBridge.open(file.getAbsolutePath(),
      O_RDONLY);
    if(Taint.getTaintString(file.publicPath) !=
      Taint.TAINT_CLEAR) {
      this.fileTaintingProp =
        Taint.getTaintString(file.publicPath);
      super.fileTaintingProp = this.fileTaintingProp;
    }
    this.shouldClose = true;
    guard.open("close");
  }
}
```

(그림 4) 수정된 FileInputStream 클래스

```
public abstract class InputStream extends Object implements Clo
seable {
  int fileTaintingProp = Taint.TAINT_CLEAR;
  public int read(byte[] buffer) throws IOException {
    if(this.fileTaintingProp != Taint.TAINT_CLEAR ) {
      Taint.addTaintByteArray(buffer,
        fileTaintingProp);
    }
    return read(buffer, 0, buffer.length);
  }
}
```

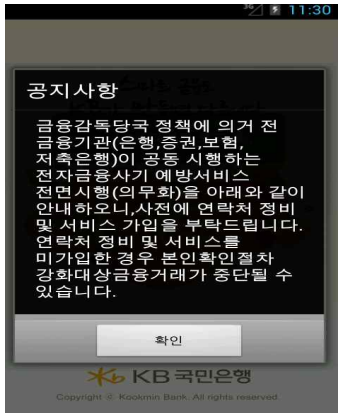
(그림 5) 수정된 InputStream 클래스

4. 실 험

본 장에서는 Tainting 기반 공인인증서 유출 탐지 시스템과 기존 분석 방법과의 차이를 공인인증서 유출 악성 어플리케이션을 분석하며 비교한다.

4.1 공인인증서 유출 악성 어플리케이션

효용성 검증을 위해 사용한 공인인증서를 유출 악성 어플리케이션 kb.apk 이다. kb.apk는 국민은행 어플리케이션으로 위장한 악성 어플리케이션으로 실행 시 그림 6, 그림 7, 그림 8과 같이 본인확인절차가 필요하다는 메시지를 출력한 후, 공인인증서 비밀번호, 개인정보, 보안카드번호를 차례로 물어본다. 이 정보와 함께 공인인증서가 저장된 NPKI 디렉터리를 압축하여 ZIP 형태로 외부의 C&C서버에 전송한다.



(그림 6) kb.apk 실행 화면

(그림 7) 공인 인증서 선택 화면

(그림 8) 공인인증서 암호 입력 화면

악성 앱을 동적 분석함에 있어 어려운 점은 최종적으로 공인인증서를 외부로 전송할 때 탐지해야하는데 해당 앱이 전송하는 외부 서버는 다운되었거나 차단되어 실제로 외부로 전송하지 못한다. 따라서 본 연구에서는 kb.apk를 디컴파일한 소스코드를 분석하여 유사한 행동을 하는 악성 앱을 제작하였다. 본 연구에서 중점적으로 다루는 공인인증서 유출과 관련된 코드부분과 유사한 NPKI.apk를 작성하였고, 이를 이용하여 본 연구의 효용성을 검증한다.

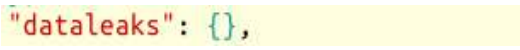
NPKI.apk의 핵심 행위는 안드로이드 기반 스마트폰에 설치 및 실행이 되면 Environment.getExternalStorageState()를 통하여 스마트폰에 내장 메모리가 마운트 되어 있는지 확인하고 Environment.getExternalStorgae.Directory()를 통하여 외장 메모리의 주소를 가져온다. 이때 안드로이드 에뮬레이터의 경우 외장 메모리의 위치는 /sdcard 이다. 이후 외장 메모리에 NPKI 디렉터리의 유무를 확인한 후 NPKI 디렉터리가 존재하면 이를 압축하여 외부의 서버로 전송한다.

4.2 분석 결과

4.2.1 수정 전 DroidBox

본 장에서는 직접 개발한 공인인증서 유출 악성 어플리케이션을 동적 분석 도구 중 하나인 수정하지 않은 DroidBox를 사용하여 분석을 진행한 결과를 다룬다.

(그림 9) 수정 전 DroidBox 분석 결과 - sendnet



(그림 10) 수정 전 DroidBox 분석 결과 - dataleaks

직접 개발한 공인인증서 유출 악성 어플리케이션을 수정 전 DroidBox를 이용하여 분석한 결과 그림 9와 같이 sendnet 항목의 분석 결과 인증서 파일을 외부 서버로 유출하는 것에 성공하였으나, 그림 10과 같이 dataleaks 항목에서는 어떠한 행위도 검출되지 않는 것을 확인할 수 있다. 이는 DroidBox에서 인증서 관련 디렉터리(NPKI) 또는 파일(SignCert.der, SignPri.key)을 taint하고 있지 않기 때문이다.

4.2.2 제안 시스템의 분석 결과

다음은 앞서 설명한 방법으로 Taintdroid를 수정한 후 DroidBox에 적용하여 분석한 결과이다.

(그림 11) 수정 후 DroidBox 분석 결과 - sendnet

(그림 12) 수정 후 DroidBox 분석 결과 - dataleaks

그림 11과 그림 12는 수정된 DroidBox로 직접 개발한 공인인증서 유출 악성 어플리케이션을 분석 한 결과의 일부이다. 수정 전 DroidBox에서는 인증서의 외부유출이 dataleaks 항목에 검출되지 않았지만, 수정한 DroidBox에서는 공인인증서와 다른 정보를 압축한 파일인 sign.zip 파일을 외부로 유출하는 것이 검출되는 것을 확인할 수 있다.

5. 결 론

스마트폰의 보급률이 증가하며 스마트폰을 기반으로 하는 악성 어플리케이션 또한 증가하기 시작하였다. 특히 스마트폰 내부에 존재하는 개인정보를 외부로 유출하는 악성코드 중에서도 국내에서 금융, 전자 거래 등에 쓰이고 있는 공인인증서를 노리는 악성 어플리케이션 또한 다양하게 발견되었다. 공인인증서는 사이버상의 인감도장이라 할 수 있기 때문에 외부로 유출 될 경우 악용 될

가능성이 있다. 본 논문에서 DroidBox를 수정하여 스마트폰 내에 존재하는 공인인증서 디렉터리인 NPKI 디렉터리를 taint하여 공인인증서를 외부로 유출하는 악성 어플리케이션을 탐지 할 수 있다는 것을 증명하였다. 향후에는 DroidBox에 taint 항목을 추가하여 조금 더 다양한 자료에 대한 유출 탐지에 대하여 연구 할 계획이다.

참고문헌

[1] 개인정보보호 종합포털-주민등록번호 수집 금지 제도 가이드라인, https://www.privacy.go.kr/nns/ ntc/selectBoardArticle.do?nttId=5006

[2] 금융위원회 고시 제 2015-7호, http://www.fsc.go.kr/know/law_most_view.jsp?menu=7410100&bbsid=BBS0079&no=30700

[3] YTN, http://www.ytn.co.kr/_ln/0103_201603220904243529

[4] 미래창조과학부, theminjoo.kr/fileDn.do?seq=17260

[5] http://www.dailysecu.com/news/articleView.html?idxno=3637

[6] Youngseok Choi, Sunghhon Kim, Dong Hoon Lee, “Study to detect and block leakage of personal informaion : Android-platform environment,” Journal of The Koread Institute of Information Security & Cryptology(JKIISC), Vol.23, No.4, August 2013

[7] Dorae Kim, Yongsu Park, “Detection of Privacy Information Leakage for Android Applications by Analyzing API Inter-Dependency and the Shortest Distance,” Journal of KIISE, Vol. 41, NO. 9, pp. 707-714, September, 2014.

[8] Francesco Di Cerbo, Andrea Girardello, Florian Michahelles, Svetlana Voronkova, “Detection of Malicious Applications on Android OS,” Proceedings of the 4th international conference on Computational forensics, IWCF'10, pp 138-149, November 2011.

[9] Yajin Zhou, Zhi Wang, Wu Zhou, Xuxian Jiang, “Hey, you, Get Off of My Market; Detecting Malicious Apps in Official and Alternative Android Markets,” Proceedings of the 19th Annual Network&Distributed System Security Symposium. February 2012.

[10] Vaibhav Rastogi, Yan Chen, William Enck, “AppsPlayground: Automatic Security Analysis of Smartphone Applications”, Proceedings of the third ACM conference on Data and application security and privacy, pp. 209-220, 2013.

[11] Min Zheng, Mingshen Sun, John C.S. Lui, “DroidTrace: A Ptrace Based Android Dynamic Analysis System with Forward Execution Capability”, Wireless Communications and Mobile Computing Conference, 2014.

[12] Martina Lindorfer, Matthias Neugschwandtner, Lukas Weichselbaum, “ANDRUBIS - 1,000,000 Apps Later: A View on Current Android Malware Behaviors”, Building Analysis Datasets and Gathering Experience Returns for Security, 2014.

[13] Willian Enck, Peter Gilbert, Byung-Gon Chun, Landon P.Cox, Jeayeon Jung, Patrick McDaniel and Anmol N.Sheth, “TaintDroid: An Informatin-Flow Tracking System for Realtime Privacy Monitoring on Smartphones,” OSDI vol. 10. pp.255-270, October, 2010.

[14] oungseok Choi, Sunghhon Kim, Dong Hoon Lee, “Study to detect and block leakage of personal informaion : Android-platform environment,” Journal of The Koread Institute of Information Security & Cryptology(JKIISC), Vol.23, No.4, August 2013

[15] Dorae Kim, Yongsu Park, “Detection of Privacy Information Leakage for Android Applications by Analyzing API Inter-Dependency and the Shortest Distance,” Journal of KIISE, Vol. 41, NO. 9, pp. 707-714, September, 2014.

[16] Francesco Di Cerbo, Andrea Girardello, Florian Michahelles, Svetlana Voronkova,

"Detection of Malicious Applications on Android OS," Proceedings of the 4th international conference on Computational forensics, IWCF'10, pp 138-149, November 2011.

[17] Yajin Zhou, Zhi Wang, Wu Zhou, Xuxian Jiang, "Hey, you, Get Off of My Market:Detecting Malicious Apps in Official and Alternative Android Markets," Proceedings of the 19th Annual Network& Distributed System Security Symposium. February 2012.

[18] Google code - DroidBox, https://code.google.com/p/DroidBox

[저 자 소 개]

윤 한 재 (Han-jae Yoon)
2016년 2월 한남대학교 컴퓨터공학과 학사
2018년 2월 한남대학교 컴퓨터공학과 석사
2018년 1월 ~ 현재 SKInfosec
<관심분야> 네트워크/시스템/스마트폰 보안, 고성능 시스템
email : hanjae.karoha@gmail.com

이 만 희 (Man-hee Lee)
1995년 경북대학교 컴퓨터공학과 공학사
1997년 경북대학교 공학석사
2008년 Texas A&M 대학교 컴퓨터공학과 공학박사
1997년~2003년 한국과학기술정보연구원 연구원
2008년~2009년 Cisco Systems, San Jose
2010년~2011년 국가보안기술연구소 선임연구원
2012년~현재 한남대학교 부교수
<관심분야> 네트워크/시스템/스마트폰 보안, 고성능 시스템, 컴퓨터교육
email : manheelee@gmail.com