스토리지의 관점에서 살펴본 구글 세이프티넷

윤주성○ 원유집
한양대학교 컴퓨터·소프트웨어 학과
yjs05@hanyang.ac.kr, yjwon@hanyang.ac.kr

Google Safety-Net : aspect of storage.

Juseong Yun○ Youjip Won
Department of Computer and Software, Hanyang University

요 약

스마트폰의 등장으로 인해 사용자의 IT 생활패턴이 바뀌었다. 인터넷 서핑, 이메일, 뉴스부터 은행업무까지 스마트폰을 통해서 해결되고 있다. 스마트폰의 사용 용도가 다양해짐에 따라 기기의 보안 또한 중요하졌다. 구글은 세이프티넷을 통해 안드로이드 기기의 보안을 체크하도록 하였다. 하지만 기기의 보안을 위해 각종 설정과 로그들을 저장하기 위해 막대한 양의 입출력이 발생하게되었다. 본 논문은 구글의 세이프티넷을 역공학하여 세이프티넷이서 사용되는 데이터베이스 중 가장 많은 양의 접근횟수를 보이는 snet_files_info 데이터베이스의 동작 원리를 분석함으로서, 세이프티넷의 효율성에 대해 분석해본다.

1. 서 론

프로세서와 스토리지와 같은 모바일 장치의 하드웨어의 성능개선과 초고속 무선 통신과 같은 주변 인프라가 발전함에 따라 세계 스마트폰 시장은 급격하게 성장하였다. 전세계 스마트폰 판매량은 2015년을 기준으로 전통적인 피처폰의 판매량을 뛰어 넘었다 [1]. IOS와 안드로이드가 각각 2007년, 2008년에 시장에 발표되었으므로, 스마트폰은 탄생한지 약 7년만에 피처폰을 대체한 것이다. 스마트폰의 대중화는 사용자들의 IT 생활패턴을 바꾸었다. 온라인 비디오와 영화감상, 이메일과 뉴스 조회부터 온라인 쇼핑에 이르기 까지 다양한 영역에서 스마트폰이 사용되고 있다. 2016년 미국인의 51%가 모바일뱅킹을 사용하는 것으로 나타났으며, 국내의 경우 87.1%가 스마트폰으로 인터넷뱅킹을 하는 것으로 나타났다. 스마트폰이 컴퓨터를 대체하기 시작한 것이다.[2-4]

스마트폰이 점차 다양한 용도로 사용됨에 따라 스마트폰의 보안은 더욱더 중요해졌다. 구글은 2013년 안드로이드 디바이스에서 보안과 관련된 디바이스 정보들과 유저 설정, 주요 이벤트들의 발생 로그들을 수집하는 세이프티넷(SafetyNet)을 추가하였다. 세이프티넷은 구글 플레이서비스 어플리케이션에서 관리되는 서비스중 하나로, 네트워크나 어플리케이션에서 발생하는 시스템 위협을 탐색하고 보호하는 기능을 제공한다. 당시 세이프티넷은 출처를 알 수 없는 어플리케이션이 설치된 경우 동작하여 디바이스를 보호하도록 설계되었다. 하지만 2015년 모든 버전의 안드로이드와 680만개의 어플리케이션을 대상으로 조사를 한 결과, 멀티태스킹기능을 통해 다른 어플리케이션을 감청할 수 있으며, 심지어 악성코드까지도 심을 수 있다는 연구 결과가 발표되었다 [5]. 이에 대해 구글은 보다 강화된 보안을 제공하기 위해, 2015년 SafetyNet의 기능을 API를 통해 제공함으로서 어플리케이션 프로그래머가 디바이스의 상태를 점검하여 주요 데이터를 스스로 보호할 수 있도록 하였으며, 2016년에 세이프티넷이 모든 어플리케이션에 대하여 검사를 하도록 변경하였다 [6].

이러한 지속적인 보안강화로 안드로이드 유저의 정보를 안전하게 보호하는 것은 성공하였지만, 디바이스의 보안을 유지하기 위해 시스템 로그들을 주기적으로 저장하기 때문에 막대한 양의 입출력이 스토리지에 발생하고 있다. 지난 2015년 04월 25일부터 2015년 05월 31 까지 36일동안 LG Nexus 5기기의 입출력 로그를 수집하여 분석한 결과 SafetyNet에서 사용하는 db중 하나인 snet_files_info에서 전체 입출력 접근 횟수의 43%가 발생한는 것이 발견되었다 [7]. 구글의 세이프티넷의 구현이 상당히 비효율 적일수 있다는 것이다.

본 논문은 안드로이드기기의 세이프티넷을 역공학하여 가장 입출력 횟수가 많았던 snet_files_info 데이터베이스의 동작 알고리즘을 분석해봄으로서 세이프티넷 구현의 효율성을 점검해 본다.

2. 안드로이드의 구조

Java를 통해 작성된 안드로이드의 어플리케이션은 컴파일 과정을 거쳐 그림 1과 같은 APK의 형식으로 구성된다. AndroidMenifest.xml은 안드로이드 어플리케이션이 사용할 수 있는 디바이스의 장치 권한과 어플리케이션에 함께 링크될 라이브러리정보와 같은 어플리케이션의 메타데이터가 저장된다. res 디렉토리는 어플리케이션에서 사용될 이미지와 사운드, UI의 배치정보를 저장한

STRING PRIMARY KEY	INTEGER	BLOB	INTEGER	INTEGER	STRING	INTEGER	INTEGER	INTEGER	STRING
full_path	time_ms	sha256_digest	is_setuid_root	is_symlink	symlink_target	file_permissions	file_owner	file_group	se_linux_security_context

표 1 snet_files_info.db의 스키마

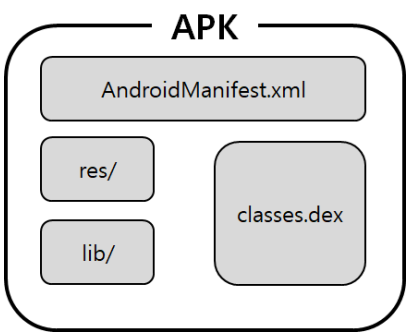

그림 1 안드로이드 apk의 구조

xml 파일과 같은 어플리케션 리소스들을 저장하는 디렉토리이며 마지막으로 lib 디렉토리에는 외부에서 작성된 자바 jar 라이브러리나, 자바 네이티브 코드로 작성된 so 라이브러리들이 저장된다. classes.dex파일은 어플리케이션의 java 코드가 안드로이드의 가상머신에서 동작할 수 있도록 컴파일된 바이너리 코드이다. dex 포맷은 안드로이드의 달빅 가상머신에서부터 사용되어 안드로이드 5(롤리팝)부터 적용된 안드로이드 런타임 (ART)에서도 사용되고 있다.

한편, 로그인관리, 지도정보, 세이프티넷과 같은 안드로이드 주요 서비스들은 안드로이드 4 (아이스크림샌드위치)에서 도입된 google play services를 통해 제공된다. google play services는 그림 2와 같이 플랫폼에서 항시 대기하여 어플리케이션에 빠른 서비스를 제공 하며 안드로이드 플랫폼과 분리되어 플랫폼 버전에 독립적으로 최신 서비스를 제공할 수 있다 [8].

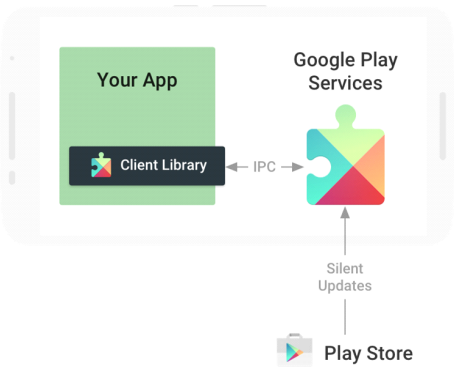

그림 3 구글 플레이서비스의 동작 [8]

3. Apk의 디컴파일

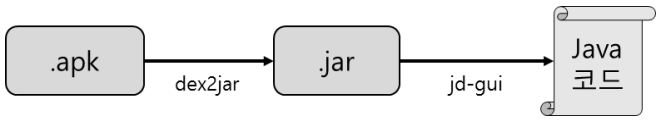

그림 4 apk 역공학 과정

본 논문은 SafetyNet의 효율성을 점검하기위해 Snet.jar 파일과 google play services 어플리케이션을 역공학하였다. google play services는 com.google.android.gms라는 패키지명으로 제작되었으며 어플리케이션 extract 앱을 사용하여 apk 파일을 추출할 수 있다. 추출한 apk는 그림 3의 과정을 통하여 java코드로 변환하였다. SafetyNet의 기능은 snet.jar에 구현되어있다. snet.jar 파일의 압축을 해제하면 dex 포맷의 코드들이 나오게 된다. 우리는 snet.dex 파일도 그림3의 과정을 거쳐 java코드로 변경시켜주었다.

4. SafetyNet의 동작 분석

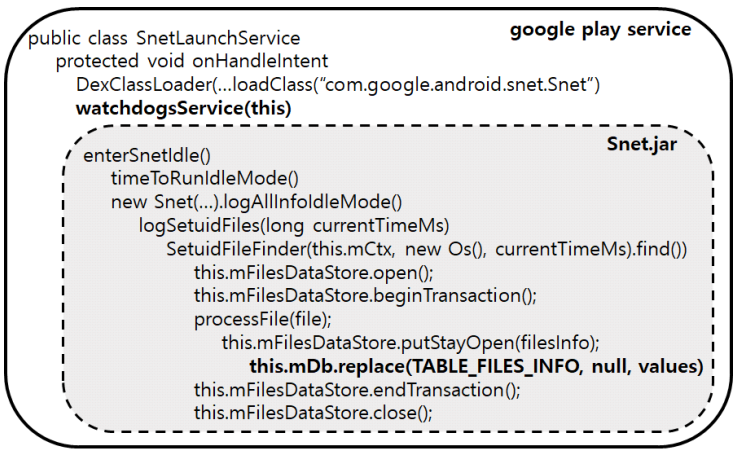

그림 2 snet_files_info.db의 업데이트 경로

역공학을 통해 분석한 snet_files_info.db의 업데이트는 매우 비효율적이었다. 표1과 같이 snet_files_info.db는 기기의 안정성을 점검하기 위해 사용되는 db로 파일의 접근권한과 같은 메타데이터 정보와 변조를 검조하기 위한 hash를 저장하는 테이블 1개만 가지고 있다. 이 테블이 갱신되는 경로는 그림4와 같다. 그림4에서 볼 수 있듯이 play service에서 Snet에 관련된 기능을 사용하면 SnetLaunchService 클래스를 생성한다. SnetLaunchService 클래스가 생성되면 내부의 인텐트를 통해 DexClas

sLoader클래스를 사용하여 동적으로 Snet.jar을 로딩한 후, enterSnetIdle() 메소드를 호출하게 된다. enterSnetIdle() 메쏘드는 그림4의 호출과정을 거쳐 processFiles을 호출한다. 위 동작은 Snet이 처음 생성될 때 watchdogsService 메소드에 등록되어 주기적으로 반복되게 된다. SetuidFileFinder()는 Sqlite 트랜잭션으로 메타데이터를 업데이트하는 메소드이다. 파일의 메타데이터를 업데이트하는 과정의 의사코드로 나타내면 아래의 그림 5와 같다. 1번 라인과 같이 디렉토리들을 저장하는 스택을 팝하며 디렉토리를 2번 라인의 listFiles() 메소드로 파일 리스트를 가져온다. 파일이 존재하는 동안 for루프를 반복하여 디렉토리으 ahems 파일에 대해 5번 라인의 processFile을 호출한다. process file은 10번에서 19번 라인과 같이 파일의 메타데이터를 FilesInfo 클래스에 저장하여 20번의 mFilesDataStore.push() 를 사용하여 db에 저장한다. mFilesDataStore.push는 결과적으로 SqliteOpenHelper 클래스의 replace() 메소드를 호출하여 db에 저장된 레코드를 새로이 갱신하게 된다. 만약 현재 파일이 디렉토리라면 22번 라인을 통하여 디렉토리 스택에 푸시하여 1번 라인의 루프에서 사용할 수 있도록 한다.

```
1.  while (!this.mDirectoriesToSearch.empty()) {
2.      File[] files = ((File) this.mDirectoriesToSearch.pop()).listFiles();
3.      if (files != null) {
4.          for (File file : files) {
5.              processFile(file); //db에 metadata를 update
6.          }
7.      }
8.  }

9.  private void processFile(File file) {
10.     if (!this.mOs.isSymlink(file)) {//db 레코드를 구조체에 저장
11.         if (file.isFile() && this.mOs.isSetuidRoot(file)) {
12.             byte[] sha256 = Utils.getSha256(file);
13.             FilesInfo filesInfo = new FilesInfo();
14.             filesInfo.filename = file.getAbsolutePath();
15.             filesInfo.present = true;
16.             filesInfo.sha256 = sha256;
17.             filesInfo.isSetuidRoot = true;
18.             filesInfo.foundAtTimeMs = this.mCurrentTimeMs;
19.             this.mSetuidFiles.add(filesInfo);
20.             this.mFilesDataStore.putStayOpen(filesInfo);
21.         } else if (file.isDirectory()) {
22.             this.mDirectoriesToSearch.push(file); //디렉토리 푸시
23.         }
24.     }
25. }
```

그림 5 snet_files_info.db에 저장될 레코드를 생성하는 알고리즘

즉 수 ms마다 디바이스에 저장된 모든 파일을 검색하여 메타데이터를 업로드하고 있는 것이다.

5. 결론

안드로이드는 기기의 보안을 유지하기위해 세이프티넷을 도입하였다. 세이프티넷은 기기의 보안을 유지하기 위해 주기적으로 기기를 체크하는 것으로 알려졌다. 하지만 과거의 연구를 통해 안드로이드의 전체 입출력의 43%에 달하는 입출력이 세이프티넷에 의해 발생함이 보고되었다 [7].

본 논문은 역공학을 통해 snet_files_info.db의 입출력이 발생하는 원인을 분석하였다. 분석결과 snet_files_info.db는 파일의 보안변화여부를 감지하기위해 기기내 모든 파일의 메타데이터를 수 ms마다 저장한다는 사실을 밝혀냈다.

6. 사사

본 연구는 한국연구재단 기초연구실 지원사업(No. 2017R1A4A1015498), IITP 전문연구실 지원사업(2018-0-00549)과 미래창조과학부 및 정보통신기술진흥센터의 정보통신•방송 연구개발사업의 일환으로 수행하였음.[R0601-16-1063, ICT 장비용 SW 플랫폼 구축]

참고문헌

[1] statista, “Feature phone and smartphone shipments worldwide from 2008 to 2020 (in 1,000 units)”, https://www.statista.com/statistics/225321/global-feature-phone-and-smartphone-shipment-forecast/

[2] statista, “Most popular mobile internet activities according to internet users worldwide as of 2nd half 2017, by device”, https://www.statista.com/statistics/249761/most-popular-activities-carried-out-on-mobile-internet-devices/

[3] statista, “Share of mobile banking users among mobile phone owners in the United States from 2009 to 2016”, https://www.statista.com/statistics/244414/percentage-of-us-mobile-phone-users-who-use-mobile-banking/

[4] KISA, “2016년 인터넷이용실태조사 최종보고서“, https://isis.kisa.or.kr/board/?pageId=060100&bbsId=7&itemId=817&searchKey=&searchTxt=&pageIndex=1

[5] Ren, Chuangang, et al., "Towards Discovering and Understanding Task Hijacking in Android." USENIX Security Symposium. 2015.

[6] Google, “Android Security 2016 Year in Review”, https://source.android.com/security/reports/Google_Android_Security_2016_Report_Final.pdf

[7] Kim, Myungsik, et al., "Dumb design, dumber usage of mobile database." Consumer Electronics (GCCE), 2015 IEEE 4th Global Conference on. IEEE, 2015.

[8] Google, "Overview of Google Play Services", https://developers.google.com/android/guides/overview