Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.31, NO.1, Feb. 2021 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2021.31.1.73

안드로이드 저장소 취약점을 이용한 악성 행위 분석 및 신뢰실행환경 기반의 방어 기법*

김 민 규,[1†] 박 정 수,[2] 심 현 석,[2] 정 수 환[3‡]
[1,2,3]숭실대학교(학생, 대학원생, 교수)

Analysis of Malicious Behavior Towards Android Storage Vulnerability and Defense Technique Based on Trusted Execution Environment*

Minkyu Kim,[1†] Jungsoo Park,[2] Hyunseok Shim,[2] Souhwan Jung[3‡]
[1,2,3]Soongsil University(Undergraduate student, Graduate student, Professor)

요 약

휴대폰 이용 시 앱 또는 웹 기반 어플리케이션을 이용하여 파일 다운로드 시, 다운로드 되는 파일들은 어플리케이션 마다 특정 디렉토리에 저장하도록 기본 경로가 설정되어 있다. 파일 관리자를 비롯하여 저장소로 접근이 필요한 여러 어플리케이션들은 여러 기능들과 서비스를 제공하기 위해, 저장소의 읽기 및 쓰기 권한을 요구한다. 이는 다운로드 경로에 직접 접근하여 사용자가 저장해놓은 수많은 중요 파일들에 직접 접근할 수 있게 됨을 의미한다. 본 논문에서는 이러한 다운로드 된 파일들의 저장 공간의 보안 취약점을 이용한 공격 가능성을 증명하기 위해 암호화를 위장한 파일 탈취 어플리케이션 기능을 개발하였다. 암호화를 진행한 파일은 암호화됨과 동시에 백그라운드에서는 해커에게 E-mail을 통해 전송된다. 개발한 어플리케이션을 악성 분석 엔진인 VirusTotal을 이용하여 검사한 결과, 74개의 엔진 모두에서 악성 앱으로 탐지되지 않았다. 최종적으로 본 논문에서는 이러한 저장소 취약점을 보완하기 위한 신뢰실행 환경 기반의 방어 기법과 알고리즘을 제안한다.

ABSTRACT

When downloading files using an app or web-based application on the user's mobile phone, the path is set to be saved in the pre-defined default directory. Many applications requiring access to storage, including file managers, require a write or read permission of storage to provide numerous functions and services. This means that the application will have direct access to the download folder where the numerous files downloaded. In this paper, to prove our feasibility of attack using the security vulnerabilities mentioned above, we developed a file hacking function disguised as an encryption function in the file management application. The file that encrypted will be sent to hackers via E-mail simultaneously on the background. The developed application was evaluated from VirusTotal, a malicious analysis engine, was not detected as a malicious application in all 74 engines. Finally, in this paper, we propose a defense technique and an algorithm based on the Trusted Execution Environment (TEE) to supplement these storage vulnerabilities.

Keywords: Android application, Trusted Execution Environment, Privacy leak

Received(10. 15. 2020), Modified(12. 10. 2020), Accepted(12. 14. 2020)

* 이 논문은 2020년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임 (No. 2019-0-00477, 가상화된 신뢰실행환경을 이용한 안드로이드 보안 프레임워크 기술 개발)

* 본 논문은 2020년도 한국정보보호학회 하계학술대회에 발표한 우수논문을 개선 및 확장한 것임

† 주저자, rlaalsrb4028@naver.com

‡ 교신저자, souhwanj@ssu.ac.kr(Corresponding author)

I. 서 론

파일 관리를 위해 유용하게 사용되고 있는 파일 관리자 어플리케이션으로는 파일 관리자 어플리케이션, 아스트로 파일 관리자 어플리케이션 등등 여러 가지가 존재한다. 파일관리자 어플리케이션은 휴대폰의 Storage Read / Write permission 획득을 통해 저장 공간에 접근하여, 존재하는 폴더들과 파일들을 관리할 수 있도록 한다. 또한, 이 파일들에 접근하여 List 구조로 사용자에게 User Interface (UI)를 제공하고, 삭제, 전송, 이동, 이름 변경 등의 기능을 사용자에게 제공한다. 따라서 파일의 편리한 관리를 위해 많은 사람들이 파일 관리자 어플리케이션을 사용한다.

여기서 문제점은, 우리가 사용 하고 있는 휴대폰에서는 웹(또는 어플리케이션)을 이용한 파일 다운로드 시, 다운로드 폴더에 모든 파일들이 저장되도록 기본 저장 경로가 설정되어있다. 즉 다운로드 경로를 사용자가 직접 변경하지 않는다면, 사용자가 다운로드 했던 모든 파일들은 다운로드 폴더에 모두 저장되게 된다는 취약점이 존재하게 된다. 결국 악성 파일 어플리케이션 의 경우에는, 다운로드 폴더의 중요 파일들이 해커에게 쉽게 노출된다. 또한, 웹 뿐만 아니라 특정 어플리케이션을 이용하여 파일을 다운로드를 할 경우에도 내부 저장소로의 접근 권한 획득이 필요하게 된다. 예를 들어, 구글 웹사이트 어플리케이션을 통해 받은 파일들 또한 다운로드 폴더에 저장된다. 이러한 안드로이드 저장 공간 취약점을 이용하여 파일 탈취가 쉽게 가능하다.

이번 논문에서는 AES 암호화기법을 이용하여 암호화 기능을 구현하고, 동시에 다운로드 폴더의 암호화를 시도한 파일을 탈취하는 악성 파일 관리 어플리케이션의 동작을 구현하였다. 암호화 기능을 사용시, 해당 파일은 암호화가 되는 동시에 암호화를 시도한 파일을 이메일을 통해 전송받도록 하였다. 본 논문의 어플리케이션은 루팅이 되지 않은 정상적인 안드로이드 기기에서 성공적으로 동작하였다.

이에 대한 방어기법으로 저장 공간의 보안성 향상을 위해 다운로드 되는 파일들이 Trusted Execution Environment (TEE)상의 메모리 공간에 저장되도록 하는 알고리즘을 제시한다. TEE는 현재 안드로이드 체계에서 사용되고 있는 정보 보안 기술로, 일반 실행환경과 하드웨어적으로 분리되는 신뢰실행환경을 제공한다. 본 논문에서 개발한 악성 어플리케이션의 동작은 기존 안드로이드 저장 공간상에서 이루어졌기 때문에 동작 할 수 있었으며 파일들의 다운로드 경로와 파일 저장 공간을 신뢰 메모리 공간으로 설정하면 저장 공간으로부터의 탈취를 방지 가능하다. 또한, 신뢰할 수 있는 어플리케이션에 한해 Trusted Application (TA)화 하여 기타 신뢰할 수 없는 어플리케이션들의 저장 공간 접근을 제한해야 한다.

본 논문은 2장에서 배경지식을 설명하며, 3장에서는 구현 방법과 동작에 대해 설명할 것이다. 이후 4장에서는 구현된 악성 앱에 대한 분석이 이루어지며, 5장에서 이러한 악성 행위에 대한 방어 기법을 제안한다. 마지막으로 6장에서 최종 결론을 통해 끝맺는다.

II. 배경 지식

2.1 안드로이드 샌드박스

안드로이드는 각 안드로이드 어플리케이션에 User-ID (UID)를 지정하고 자체 프로세스에서 실행하는 샌드박스 저장 방식을 사용한다. 이러한 안드로이드 샌드박스의 구조는 “Fig. 1.”과 같다. 현재 안드로이드 체계에서 어플리케이션들은 자신만의 샌드박스 구조를 가지고 있으며 기타 정보들 또한 자신의 독립적인 샌드박스 내부에 저장된다. 또한, 기본적으로 어플리케이션들은 추가적인 권한 없이는 서로 상호작용 할 수 없게 되어있는 구조이고, 이 때문에 보안성을 향상시킨다. 때문에 외부로의 상호작용이 더 필요한 경우에는 추가적인 접근 권한을 획득하여 동작하는 구조를 이루고 있다. 현재 안드로이드에서는 디바이스 내부 데이터들이 쉽게 액세스 되지 않도록

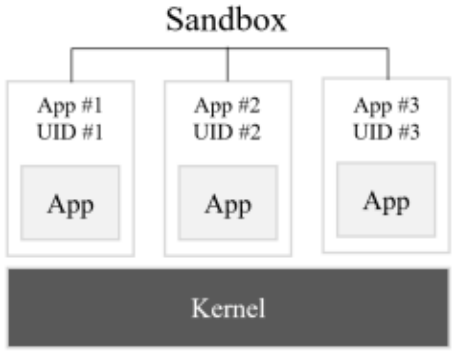


Fig. 1. Android sandbox structure

보안 수준을 설정하는 것을 권장한다. 본 논문에서 다루는 파일 관리자의 경우 저장 공간에 직접 접근 권한을 얻게 되므로, 샌드박스 내부가 아닌 저장 공간으로의 접근이 가능하다[1]. 추가적으로 웹을 이용하는 SNS, Google 등의 다양한 어플리케이션들도 특정 파일을 다운로드 할 경우에, 저장 공간으로의 접근 권한을 추가적으로 획득하여 다운로드 폴더에 저장한다.

2.2 안드로이드 악성 어플리케이션 분석 기법

현재 Google에서는 안드로이드 기기의 악성 어플리케이션으로부터 기기를 보호하기 위해 **Google Play Protect[2]** 서비스를 지원하는데, 이 **Google play protect** 서비스는 머신러닝 알고리즘으로 개발되어 멀웨어를 차단하는 기능을 한다[3]. 현재 Google에서는 악성코드에 감염된 어플리케이션을 발견 시, 구글 플레이 스토어에 해당 어플리케이션이 등록되는 것을 허용하지 않으며, 만약 설치 후 감염사실이 확인 될 경우 사용자에게 경고 알림을 보내고 삭제하며, 주기적으로 기기를 검사한다. 필요 시에는 사용자가 직접 **Google Play Protect** 기능을 해제할 수 있으나, 기기는 다양한 악성 어플리케이션으로부터 위험에 노출된다. 그러나 본 논문에서 다루는 악성 파일 어플리케이션의 경우, **Google Play Protect** 기능으로부터 악성 어플리케이션으로 탐지되지 않는다. 본문에서 다루는 어플리케이션의 경우에는 평상시 네트워크를 사용하거나, 특정 URL에 접근하거나 등의 백그라운드 동작을 하지 않기 때문에 이러한 보안망의 탐지로부터 벗어날 수 있었으며, 설치 이후 기기에 남아있어도 악성 어플리케이션으로 탐지되지 않는다.

2.3 신뢰실행환경 구조

Trusted Execution Environment (TEE)는 현재 안드로이드 체계에서 제공하는 신뢰 실행 환경이며, "Fig. 2. "와 같다. TEE는 안드로이드 운영체제 상에서의 기존 커널로부터 독립된 환경을 제공하여 컨텐츠들의 보안 수준을 향상시킨다[4]. 또한 TEE에서 데이터를 실행하고 처리함으로써 안전한 환경에서 실행을 가능하게 한다. 이 신뢰공간의 활용을 통해 모바일 디바이스에 설치될 수 있는 악성 어플리케이션들과 안드로이드 보안 취약점들로부터 저장 공간의 컨텐츠들이 보호된다. 이러한 방법을 통해, 보다 높은 보안 수준을 요구하는 결제 시스템, 인터넷 뱅킹, 생체 보안 등의 다양한 기능들에 사용가능하다.

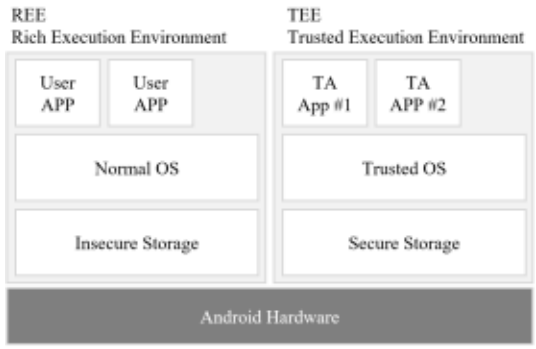


Fig. 2. Structure of TEE in Android

본 논문에서 개발한 파일 탈취 기능은 기존 OS상에서 저장소 접근 권한만으로 쉽게 동작하였다. 본 논문에서는 파일 저장 및 처리에 있어서 TA와 TEE 공간 활용을 통한 비신뢰 어플리케이션들과 기타 동작들에 대한 제한 필요성 및 다운로드 되는 파일들의 TEE 활용 알고리즘을 제안한다.

III. 악성 행위 구현

이 섹션에서는 다루고자 하는 어플리케이션의 동작을 위한 구현 기법을 소개한다. 구현한 파일 탈취 동작은 위에 첨부한 "Fig. 3."과 같다. 본 논문에서는 암호화기능을 위장한 탈취를 구현하였다.

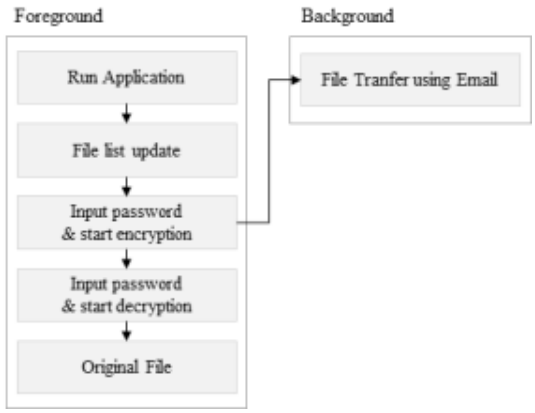


Fig. 3. Malicous Application Structure Design (Attack Flow)

"Fig. 3."에서는 포그라운드 기능, 백그라운드 기능으로 나누어져 있으며, 각각의 기능 명세는 다음과 같다. 포그라운드에서는 내부 저장소로 접근 권한을 얻으며 암호화 및 복호화작업을 진행한다. 백그라운드에서는 암호화기능과 동시에 해커에게 암호화를 시도한 파일이 SMTP를 이용한 메일 전송방식으로 전송된다.

3.1 파일 유출 기능 구현

현재 안드로이드 스튜디오에서는 파일 및 디렉토리 관련 변수 및 함수 기능을 지원한다. 다운로드 폴더 및 내부 저장 공간에 접근권한을 가진 악성 파일 관리자 어플리케이션은 (*getExternalStoragePublicDirectory*) 저장 공간 내부 폴더의 파일들에 대한 접근이 가능해지며, 파일 리스트를 만들어 사용자에게 보여주게 된다. "Fig. 3."의 오른쪽 그림에 이 어플리케이션의 백그라운드 동작(탈취)을 나타내었다. 이를 통해 encryption event 발생 시, 암호화 시도한 파일을 이메일을 통해 전송하여 탈취한다.

메일 전송 시 필요한 정보들이 있는 aar파일을 import하여 메일 전송방식을 사용하였다. 본 논문에서 사용한 방식을 제외하고도 안드로이드 스튜디오에서는 다양한 메일 전송 방식을 제공한다.

추가적으로 메일 전송 기능 구현을 위해 SMTP 방식을 이용한 라이브러리를 구현하였다. 또한 탈취 동작을 위해 *INTERNET_PERMISSION* 권한을 요구한다. SMTP 방식의 인증을 위해서는 해당 이메일 발신자의 계정 정보가 필요하기 때문에 Google 계정을 생성하여, 낮은 단계의 보안 수준을 가지도록 계정 설정을 변경했다. 그 후 계정의 ID와 password를 비롯한 메일 발신자의 정보를 입력받으며, 파일과 함께 파라미터로 사용하여 최종적으로 탈취를 완료한다.

3.2 파일 로딩 및 암호화, 복호화 기능 구현

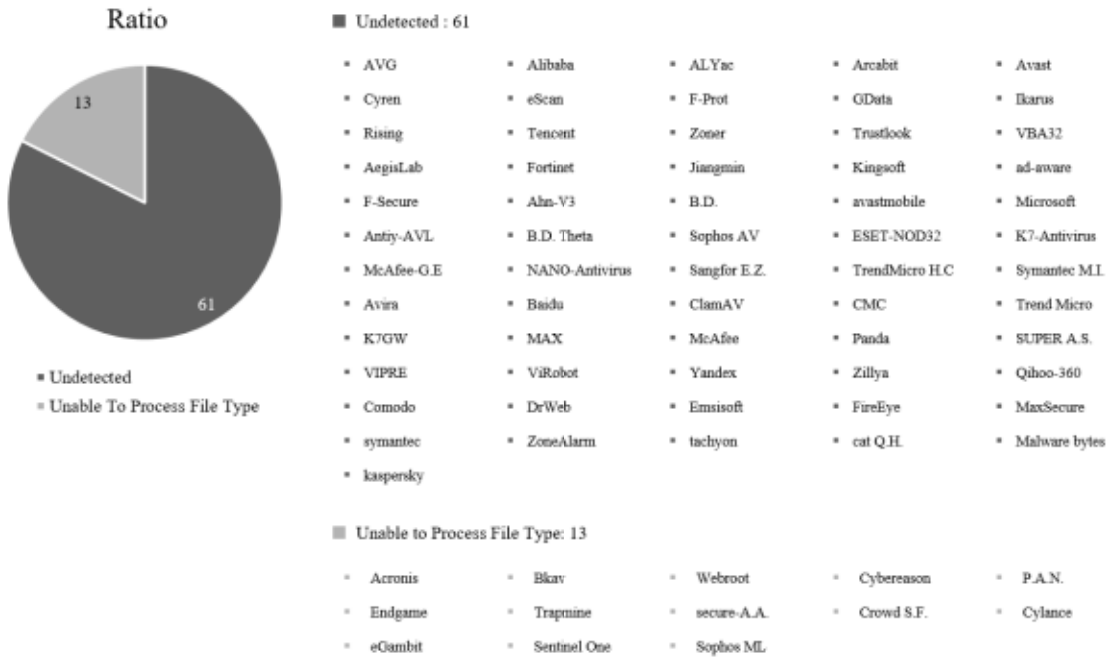
"Fig. 3."의 왼쪽 영역은 어플리케이션의 정상 동작인 암호화이다. 암호화 시에는 AES암호화 방식을 사용하였고, 16byte 키의 암호를 입력받아 사용하였다[5]. 파일에는 다양한 형식의 파일 타입이 존재한다. jpeg, hwp, pdf 등의 다양한 파일 종류가 존재하는데, 이들은 각각 고유의 파일 구조를 가지며 내부 구조는 바이너리 배열 및 바이트 구조로 되어있다. 따라서 파일을 암호화하기 전에, 암호화 알고리즘에 넣기 위해 바이트화(*FileToByte*) 한다. 암호화 이벤트 발생 시, EditText 인터페이스를 통해 16byte 키의 암호를 입력받아 암호화 알고리즘의 패스워드로 사용한다. 그 후 암호화된 바이트 배열을 *FileOutputStream*을 이용하여 암호화된 파일을 생성한다. 복호화 시에는 암호화 시 사용한 똑같은 16byte 키의 암호를 입력받고, 암호화 된 파일을 상대로 복호화를 진행한다. 그 후 복호화 된 바이트 배열을 새로운 파일로 만들어내어 원래의 파일과 내용이 같음을 확인하였다.

IV. 악성 행위 평가

현재 안드로이드 체계에서는 2절에서 설명한 바와 같이, 안드로이드의 사용 증가와 동시에 다양한 보안 취약점을 이용한 악성 어플리케이션들 또한 증가하고 있다는 문제점을 가지고 있다[6][7]. 파일 관리자 어플리케이션의 경우, 최초 실행 시 저장소 읽기 및 쓰기 권한인 *READ_EXTERNAL_STORAGE* 및 *WRITE_EXTERNAL_STORAGE* 권한을 얻는다[8]. 위 두 권한이 허용되면 해당 어플리케이션은 내부 저장 공간으로의 접근이 가능하며, 따라서 타 어플리케이션 및 웹에서 다운로드 된 파일들에 접근이 가능하다.

웹 및 일부 어플리케이션으로부터 받은 파일들은 모두 기기 저장 공간의 다운로드 폴더로 다운로드 된다. 본 논문에서는 다운로드 폴더의 파일들을 상대로 암호화기능을 제공함과 동시에 백그라운드에서는 파일 탈취기능을 구현하였다. 공격 대상의 파일은 SMTP를 통해 암호화를 시도한 파일의 저장경로와 파일을 첨부파일의 형태로 전송하여 탈취하였다. 해당 어플리케이션은 루팅이 되지 않은 정상적인 안드로이드 디바이스 상에서 성공적으로 동작하였다.

추가적으로 본 논문에서 다룬 어플리케이션의 경우 Google에서 제공하는 **Google Play Protect**와 악성 어플리케이션 검사 엔진을 제공하는 사이트인 **VirusTotal[9]**로부터 악성 어플리케이션으로 탐지되지 않았다. Google Play store에서는 디바이스에 설치된 어플리케이션들의 Google Play Protect 엔진 검사 결과를 확인할 수 있으며, 해당 어플리케이션이 지속적으로 악성 어플리케이션으로 탐지가 되지 않음을 확인하였다. 또한 **VirusTotal**에서의 검사 결과, 총 74개의 모듈 중 **Acronics,**



* (Each acronym represent as follows: B.D. = Bit Defender, cat Q.H. = cat QuickHeal, Crowd S.F. = Crowd Strike Falcon, McAfree G.E. = McAfree GW-Edition, P.A.N. = Palo Alto Networks, Symantec M.I. = Symantec Mobile Insight, SUPER A.S. = Super Anti Spyware, Secure A.A. = Secure Age Apex, Sangfor E.Z = Sangfor Engine Zero, TrendMicro H.C = TrendMicro HouseCall)

Fig. 4. Result from VirusTotal. All listed engine cannot detect the application as malware.

SecureAge를 비롯한 13개의 모듈로부터는 해당 어플리케이션의 검사를 지원하지 않아 검사가 불가능하였으며, 나머지 61개의 항목으로부터는 악성 어플리케이션으로 탐지되지 않았다. 결과는 "Fig. 4. "과 같이 나타내었다. VirusTotal에서 악성 어플리케이션으로 판단하지 않은 이유는 행위적 모델을 통한 악성코드 감지를 수행할 경우에 발생할 수 있는 False-Positive 오류를 최소화하기 위하여, Signature 기반의 탐지 룰을 활용하기 때문이다.

이러한 보안 취약점을 이용하여, 본 논문에서는 파일 관리자의 암호화기능과 동시에 암호화를 시도한 파일을 쉽게 탈취해 내는 방법을 설명하였다. 파일 암호화 및 복호화 결과 파일은 "Fig. 5."에 나타냈다. hwp 확장자 파일을 대상으로 탈취를 진행하였으며 성공적으로 동작했다. 다른 종류의 파일 타입 또한 탈취까지 성공적으로 동작했다.

또한, 다양한 메신저 어플리케이션, SNS 어플리케이션 등등 다양한 어플리케이션들은 평소 동작 시에는 내부 저장 공간으로의 접근 권한이 필요하지 않다. 하지만 파일을 첨부하거나 다운로드 하는 등의 저장 공간 접근이 필요한 동작을 하는 경우에는 내부 저장 공간에 대한 접근 권한을 얻게 된다[10]. "Fig. 6."에는 메일을 통한 파일 탈취 결과를 볼 수 있다. 메일 제목에는 파일의 사용자 기기에서의 절대 경로로 보내도록 설정했으며 최종적으로 암호화를 위장한 탈취를 성공했다. 이는 파일에 접근하는 모든 기능들, 예를 들어 이동, 복사, 삭제 등의 다양한 기능들을 통해 탈취 기능 및 악성 기능을 위장시킬 수 있다는 점을 나타낸다. 따라서 이러한 취약점을 이용하여 안드로이드 내부 저장 공간의 파일 탈취가 쉽게 가능하다는 점을 통해, 안드로이드 체계에서는 저장 공간 보안성 향상을 위해 백그라운드 상에서 익명의

```
-rw-r--r--  1 system system 11264 Oct  3 11:54
Test.hwp
-rw-r--r--  1 system system 11280 Oct  8 04:25
Test.hwp_enc.hwp
-rw-r--r--  1 system system 11264 Oct  8 04:26
Test.hwp_enc.hwp_ori.hwp
```

Fig. 5. Result of encryption and decryption on hwp extension file

Fig. 6. Result of File deception via E-mail

프로세스가 파일 암호화를 시도하는 등 저장 공간에 접근하는 기능에 대해서는 주기적인 모니터링[11]과 접근 제한이 이루어져야 한다.

V. 제안 방어 기법

5.1 머신러닝 기반 악성코드 분석 기법

안드로이드 악성코드 분석 기법 중 하나인 정적분석은 대상 어플리케이션의 실행 없이 데이터를 이용하여 악성 행위의 탐지를 진행한다[12]. 또 하나의 기법인 동적 분석은 실행 중인 어플리케이션으로부터 특성을 추출하는 방식이며, 네트워크 트래픽, 배터리 사용량, IP 주소 등의 정보들이 분석에 사용된다.

현재 머신러닝 기반의 악성코드 분석 기법으로는 권한 기반 안드로이드 악성코드 탐지[3][13], API call time interval을 활용한 머신러닝 기반의 악성코드 탐지[14] 등 다양한 연구가 진행되고 있다. 그 중에서 권한 기반의 안드로이드 악성 코드 탐지의 경우, APK에 대한 권한 및 MD5값, Software Development Kit (SDK) 버전 정보, 라이브러리 정보 등을 추출하여 머신러닝에 학습시킨다.

반면 이러한 권한 기반의 악성 행위 분석 기법은 정상 앱들 또한 비슷한 권한 셋으로 인한 오탐 문제와, 분석 시 소요되는 리소스 및 오버피팅 문제로 인해 정상적인 방어가 불가능하다.

현재 이러한 머신러닝 기반 악성코드 분석의 한계점으로 인해, 오버피팅의 발생 방지를 위한 연구인 PFESG(Permission Based Android Malware Feature Extraction Algorithm)가 진행되었다[15]. 이는 안드로이드 권한 기반 멀웨어 추출 방법으로, 사용률이 높은 권한들을 사용하여 분류 모델의 오버피팅 발생을 방지한다.

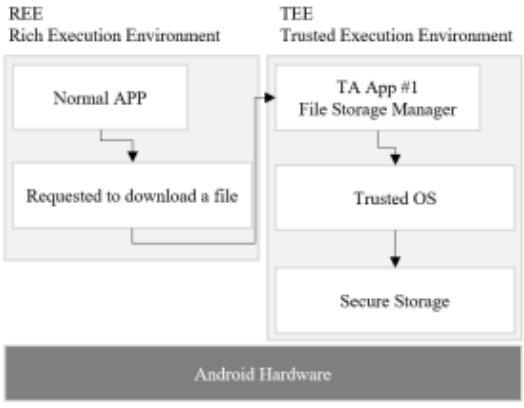


Fig. 7. Proposed structure of defense technique using TEE

5.2 신뢰실행환경을 이용한 방어 기법

기존 안드로이드 체계에서 앱 또는 웹을 이용하여 저장 공간 권한 획득 후 파일 다운로드 시 직접적으로 내부 저장소 다운로드 폴더 내부에 밀집되어 저장되는 것을 확인하였다. 본 논문에서 개발한 어플리케이션을 제외하고도 다양한 어플리케이션들이 저장 공간 접근권한 획득을 통해 다운로드 폴더 및 공통 저장 공간에 쉽게 접근이 가능했다. 그러나 이는 심각한 저장 공간상의 취약점이므로 이를 방지하기 위해 TEE 신뢰 공간 기반의 파일 다운로드 및 저장에 관한 알고리즘을 제시하며, 이는 “Fig. 7. ”에 나타내었다. 기존 안드로이드 저장 공간의 다운로드 경로와 다르게 신뢰된 어플리케이션들 및 웹의 파일 다운로드 경로를 TEE상의 저장 공간으로 설정한다. 디바이스가 전달받은 파일 다운로드 요청을 처리하기위해 TEE상의 저장 공간으로 전달하며, 저장 공간에서의 신뢰된 어플리케이션인 파일 관리 및 저장소 관리 어플리케이션이 요청들을 처리 하여 보다 안전한 저장 공간에 컨텐츠들을 보관한다[16][17]. 이를 통해 신뢰할 수 없는 어플리케이션들 및 저장 공간 취약점을 이용한 악성 동작으로부터 저장 공간의 중요 파일들을 안전하게 저장할 수 있다.

VI. 결 론

안드로이드의 저장 공간 취약점을 이용하여 기기 저장 공간의 접근을 통해 파일을 탈취해냈다. 파일은 암호화를 위장한 메일 전송방식으로 탈취해냈으며. **Google Play Protect** 및 **Virustotal** 사이트

에서 어플리케이션 검사 결과 악성 어플리케이션으로 탐지되지 않았다. 이러한 안드로이드 저장 공간의 취약점을 보완하기 위해 프로세스의 저장 공간 접근 동작에 대해서는 백그라운드 상에서의 주기적인 모니터링과 추가적인 보안 기능 및 접근 제한이 이루어져야 한다.

또한 악성 어플리케이션들의 이러한 저장 공간 접근 제한을 위해 파일들의 저장 경로 및 처리에 대한 해결책을 제시하였다. 어플리케이션 신뢰도에 따른 TA관리 및 TEE환경 사용을 통해 저장 공간에서의 파일 관리와 보안성 향상이 이루어져야 하며 민감하고 중요한 파일들에 대한 안드로이드 저장 공간 상에서의 추가적인 관리 기능이 필요하다.

References

[1] Jae Keun Lee and Eul Gyu Im. "A Study on the Exception About Android Sandbox systems," Proceedings of Symposium of the Korean Institute of communications and Information Sciences, pp.784-785, 2012.

[2] Google, Android - Google Play Protect, "https://www.android.com/intl/ko_kr/play-protect/," last accessed Oct 2020.

[3] Seongeun Kang, Nguyen Vu Long and Souhwan Jung. "Android Malware Detection Using Permission-Based Machine Learning Approach," Journal of the Korea Institute Information Security & Cryptology, vol.28, no.3, pp.617-623, 2018.

[4] Mohamed Sabt, Mohammed Achemlal and Abdelmadjid Bouadballah. "Trusted Execution Environment: What It Is, and What It Is Not," 2015 IEEE Trustcom/BigDataSE/ISPA, vol.1, IEEE, 2015.

[5] Tayde, Suchita and Seema Siledar. "File Encryption, Decryption Using AES Algorithm in Android Phone," International Journel of Advanced Research in computer science and software engineering, vol.5, no.5, pp.550-554, 2015.

[6] Kimberly Tam, Ali Feizollah, Nor Badrul Anuar, Rosli Salleh and Lorenzo Cavallaro. "The evolution of android malware and android analysis techniques," ACM Computing Surveys (CSUR) vol. 49, no.4, pp.1-41, 2017.

[7] Saba Arshad, Abid Khan, Munam Ali Shah and Mansoor Ahmed. "Android malware detection & protection: a survey," International Journal of Advanced Computer Science and Applications vol.7. no.2. pp.463-475. 2016.

[8] Jin a Kang and Hyoungshick Kim. "A Study on Application Permissions for Android Devices," Journal of the Korean Information Science Society, pp.808-810, 2013.

[9] Google, VirusTotal, "https://www.virustotal.com/," last accessed Oct 2020.

[10] Jongmun Jeong, Hoon Lee and Mintae Hwang. "Selective Management of System-level Access Permission in Android-based Application," The Korea Institute of Information and Communication Engineering, vol.20, no.1, pp.87-93, 2016.

[11] Jun Li, Lidong Zhai, Xinyou Zhang and Daiyong Quan. "Research of android malware detection based on network traffic monitoring," 2014 9th IEEE Conference on Industrial Electronics and Applications, IEEE, 2014.

[12] Payet, Étienne and Fausto Spoto. "Static analysis of Android programs," Information and Software Technology vol.54, no.11, pp.1192-1201, 2012.

[13] Zarni Aung and Win Zaw. "Permission-based android malware detection," International Journal of Scientific & Technology Research, vol.2, no.3, pp,228-234, 2013.

[14] Young Min Cho and Hun Yeong Kwon. "Machine Learning Based Malware

Detection Using API Call Time Interval," Journal of the Korea Institute of information Security & Cryptology, vol.30, no.1, pp.51-58, 2020.

[15] Chengcheng Wang and Yuquing Lan. "PFESG: Permission-based Android Malware Feature Extraction Algorithm," Proceedings of the 2017 VI International Conference on Network, Communication and Computing, pp.106-109, 2017.

[16] Jin Soo Jang, Sunjune Kong, Minsu Kim, Daegyeong Kim and Brent Byunghoon Kang. "SeCReT: Secure Channel between Rich Execution Environment and Trusted Execution Environment," NDSS, 2015.

[17] Konstantin Rubinov, Lucia Rosculete, Tulika Mitra and Abhik Roychoudhury. "Automated partitioning of android applications for trusted execution environments," 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE), IEEE, 2016.

〈저자소개〉

김 민 규 (Minkyu Kim) 학생회원
2021년 2월: 숭실대학교 전자정보공학부 IT융합전공 졸업
〈관심분야〉 IT융합, 시스템 보안, 네트워크 보안

박 정 수 (Jungsoo Park) 학생회원
2013년 2월: 숭실대학교 정보통신공학과 졸업
2015년 2월: 숭실대학교 전자공학과 석사
2015년 3월~현재: 숭실대학교 융합소프트웨어학과 박사과정
〈관심분야〉 모바일 보안, 클라우드 보안, 인증, 악성코드 분석

심 현 석 (Hyunseok Shim) 학생회원
2019년 2월: 숭실대학교 전자정보공학과 졸업
2019년 3월~현재: 숭실대학교 정보통신융합학과 석사과정
〈관심분야〉 AI 보안, 모바일 보안, 흐름 분석, 개인정보 보호

정 수 환 (Souhwan Jung) 종신회원
1985년 2월: 서울대학교 전자공학과 졸업
1987년 2월: 서울대학교 전자공학과 석사
1996년 6월: University of Washington 박사
1988년~1991년: 한국통신 전임 연구원
1997년~현재: 숭실대학교 전자정보공학부 교수
〈관심분야〉 AI 보안, 모바일 보안, 클라우드 보안, 네트워크 보안