정 보 보 호 학 회 지
제 29 권 제6호, 2019. 12

안드로이드 FDE·FBE 복호화 연구 동향

서 승 희*, 이 창 훈**

요 약

Full Disk Encryption(FDE)과 File Based Encryption(FBE)는 파일 디스크를 암호화하는 방식으로 안드로이드에서는 연락처, 문자 등의 사용자 데이터가 저장되는 데이터 파티션(/data)에 적용된다. FDE는 파티션 전체를 하나의 키로 암호화하는 방식이나 FBE는 2개 이상의 키로 파티션을 나누어 암호화한다. 이러한 FDE와 FBE는 기기 분실 및 도난 시 개인 정보 유출 피해를 방지할 수 있으나, 디지털 포렌식 수사 과정에서 증거 데이터 수집 및 분석을 어렵게 한다. 따라서 디지털 포렌식 관점의 FDE. FBE 분석 및 복호 방안에 관한 연구가 필요하다. 본 논문은 기존 FDE와 FBE의 복호 및 안전성 연구를 정리하고, 매년 FBE·FDE가 보완되어 탑재되는 새로운 안드로이드 버전에 발맞춘 꾸준한 분석의 필요성을 시사한다.

Ⅰ. 서 론

안드로이드는 지난 5년간 국제시장 점유율이 가장 높은 모바일 운영체제[1]로 최근 개인 정보 보호에 대한 인식이 높아짐에 따라 다양한 보안 기술이 적용되어 왔다. 특히 사용자 데이터를 비정상적인 물리적 접근으로부터 보호하기 위해, 안드로이드 킷캣(Android 4.4)부터 Full Disk Encryption(FDE)을 적용하기 시작했다.

FDE는 안드로이드 시스템에서 연락처, 문자, 시스템 환경 설정 정보, 애플리케이션 데이터 등 사용자의 개인 정보가 주로 저장되는 사용자 데이터 파티션(/data) 전체를 암호화하여 디스크에 쓰는 방식이다[2]. 이는 안드로이드 롤리팝(Android 5.0)부터 디스크 암호키 관리를 Trusted Execution Environment(TEE)에 의존하기 시작하면서 보안성이 크게 강화되었다.

하지만 FDE는 사용자 데이터에 접근하기 위해서는 파티션을 통으로 복호화해야 하므로 디바이스 부팅 후 사용자가 크리덴셜을 입력하기 전 상태인 다이렉트 부트(Direct Boot)모드[3]에서 착신 제한, 알람 미작동 등의 문제가 발생할 수 있다[4].

구글은 이와 같은 문제점을 보완하고 데이터 입출력(I/O)의 시간 효율을 높이기 위해 파일 시스템 수준의 암호화 방식인 File based Encryption(FBE)을 안드로이드 누가(Android 7.0)부터 지원하기 시작했다.

하지만 안드로이드의 보안 기술 강화로 인한 사용자 데이터에 대한 FBE·FDE 적용은 경찰을 비롯한 법집행기관의 디지털 포렌식 수사에 어려움을 초래하고 있다. 예를 들어, 구동이 불가할 정도로 파손된 스마트폰, 태블릿의 경우, 물리적으로 데이터를 복원하더라도 사용자 데이터 파티션이 암호화되어 있으므로 분석 및 증거 수집이 어렵다.

모바일 기기는 사용 빈도가 높고 안드로이드 시장 점유율의 지속적인 증가 추세를 고려할 때[1] 특정 사실의 입증을 위한 사용자의 행위 데이터가 저장되어 있을 가능성이 크므로, 안드로이드 사용자 데이터 파티션에 대한 FBE·FDE의 복호화 연구는 중요하다. 또한 매년 새로 출시되는 안드로이드 버전에 FBE·FDE가 보완되어 탑재되므로 이에 발맞춘 꾸준한 연구가 필요하다.

본 논문은 FBE·FDE의 특징과 기존 안전성 분석 및 복호화 연구를 소개하고자 한다.

Ⅱ. Full Disk Encryption (FDE)

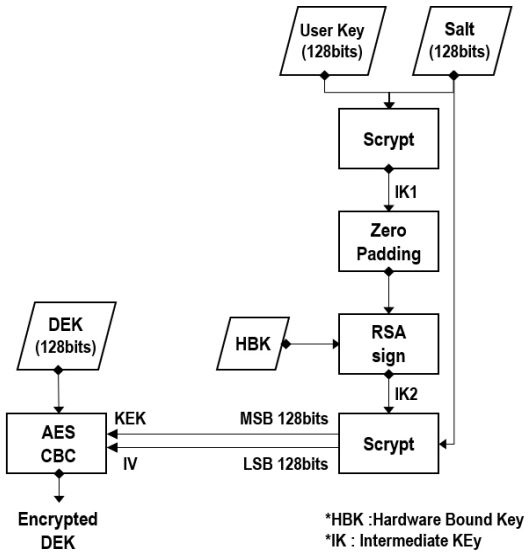
FDE는 랜덤하게 생성된 하나의 128 bits 키를 사용하여 AES-CBC로 사용자 데이터 파티션(/data) 전체를 암호화한다[5]. 이때 사용되는 암호키를 Disk Encryption Key(DEK)라고 하고, 암호화를 마친 DEK는 Key Encryption Key(KEK)로 암호화되어 Crypto

* 서울과학기술대학교 컴퓨터공학과 암호학 및 정보보안 연구실(s.eley.seo@gmail.com)
** 서울과학기술대학교 컴퓨터공학과(chlee@seoultech.ac.kr)

Footer에 저장된다. Crypto Footer는 디스크 암복호화에 필요한 데이터를 저장하는 구조체로 /metadata(구글), /data(삼성) 뒤 또는 /encrypt(엘지)에 위치한다. 그림 1은 Android 5.0 이상 버전의 FDE에서 DEK가 암호화되는 과정을 나타낸다[5].

그림 1에서 부팅 시 입력받는 사용자 키가 KEK를 유도하는 데 관여함을 알 수 있다. 사용자 키는 안드로이드 디바이스의 화면 잠금 키와 동일하고 PIN, 패스워드, 패턴, 지문, 얼굴, 홍채 등으로 설정될 수 있다.

FDE는 Android 4.4~9까지 지원되고 있으며 Android 5.0부터 RSA Sign 과정이 추가되며 보안이 강화되었다. RSA Sign에 필요한 키는 Trust Execution Environment(TEE) Processor로만 접근할 수 있는 메모리 영역에 존재하는 Hardware Bound Key로 생성되고, 키의 생성은 TEE에서 이루어진다. TEE는 데이터의 무단 변조를 방지하기 위해 분리된 커널에서 운용되는 프로세싱 환경으로 메모리 일부에 대한 메인 프로세서의 접근을 제한하여 실행 코드와 데이터의 신뢰성 및 무결성을 보장한다.



(그림 1) Disk Encryption Key(DEK) 암호화 과정 (Android 5.0 ~)

2.1. Userkey Brute Force

Oliver Kunz[2]는 Online, Offline, Semi-offline으로 나누어 Android 4.4.1과 Android 5.0이 설치된 Nexus 6에서 FDE에 대한 Brute Force을 시도하였다.

2.1.1. Online Brute Force

Online Brute Force는 구동되고 있는 안드로이드 기기 내에서 사용자 키 전수조사를 통해 디스크를 복호화한다. 일반적으로 사용자가 사용자 키를 입력하는 방법과 유사한 방식으로 사용자 키를 전수조사한다.

구글에서 제공하는 개발자 도구인 Android Debug Bridge(ADB)를 이용하여 전수조사 공격을 자동화할 수 있다. input 명령어의 옵션을 통해 사용자 키를 자동으로 입력하고 사용자 키 인증이 완료될 시 로깅되는 문자열 mLockScreenShown이 탐지되면 키 입력을 중단한다. PIN, 패스워드, 패턴으로 지정된 사용자 키의 전수조사를 위한 input 명령어 옵션의 활용 방법은 표 1과 같다.

ADB 명령을 안드로이드 기기에서 실행하기 위해서는 USB 디버깅 인가가 필요하며, 이를 회피하는 방안으로 USB OTG(On-The-Go), 터치 화면을 자동으로 누르는 기계 등이 제안된 바 있다.[5, 7]

(표 1) Online Brute Force에서 사용자 키 입력을 위한 ADB Input 명령어 옵션

옵션	설명	사용자 키 입력 방법
tab	입력한 스크린 좌표에서 손가락 터치 시뮬레이션	후보 PIN 번호와 대응되는 키패드의 스크린 좌표 입력
text	포커스된 Textfiled에 문자열 파라미터 입력	Keyevent 옵션으로 Textfiled를 포커스 한 후 후보 패스워드 키 문자열 입력
swipe	좌표 1에서 좌표 2로 터치스크린을 문지르는 행위 시뮬레이션	9개의 dot의 스크린 좌표를 조합한 후보 패턴 키 입력

```
253  /*
254   * Calculates the timeout in milliseconds as a function of the failure
255   * counter 'x' as follows:
256   *
257   * [0, 4] -> 0
258   * 5 -> 30
259   * [6, 10] -> 0
260   * [11, 29] -> 30
261   * [30, 139] -> 30 * (2^((x - 30)/10))
262   * [140, inf) -> 1 day
263   *
264   */
```

(그림 2) Android 6.0 Timeout 규정

Android 5.0까지 사용자 키 인증 과정에서 5번 실패 시 패널티로 30초의 Timeout이 부과되었으나, Android 6.0이상에서는 Timeout 규정이 강화되어 현실적으로 Online Brute Force가 불가능하다. 자세한 Android 6.0 이상 버전의 Timeout 규정은 그림 2와 같다[8].

2.1.2. Offline Brute

Offline Brute Force 은 안드로이드 기기의 디스크에 물리적 접근 및 복사가 가능한 것을 전제로 기기가 작동이 불가할 때, /data와 Crypto Footer를 이미징하여 별도의 컴퓨터에서 FDE 복호화를 시도한다.

안드로이드 파일 시스템은 ext4로 /data의 첫 1024 bytes가 Null로 채워지고, 오프셋 0x1080에 값이 Ext4 시그니처(0xEF53)가 저장된다[9]. 따라서 전수조사 시 첫 1024bytes와 시그니처를 확인하여 복호화 여부를 확인할 수 있다.

이 공격 기법은 Online Brute Force보다 20배 이상 빠르나, RSA Sign 과정이 추가된 Android 5.0 이상부터는 적용이 어렵다.

2.1.3. Semi-offline Brute Force

Semi offline Brute Force는 Android 5.0 이상에서 적용이 어려운 Offline Brute Force 을 보완한 기법이다.

별도의 컴퓨터에서 재현이 어려운 RSA Sign 과정을 포트로 통신하는 Client-Server 형 애플리케이션으로 안드로이드 기기 내에서 수행한다. 애플리케이션의 동작 방식은 그림 3과 같다.

RSA Sign 이 외의 과정은 Offline Brute Force 와 동일하게 /data와 Crypto Footer를 이미징하여 별도의 컴퓨터에서 복호화를 시도한다.

(그림 3) Client-Server형 애플리케이션을 이용한 RSA Sign 동작 과정

2.2. Cold Boot

Cold Boot는 Müller와 Spreitzenbarth[10]이 제안한 방법으로 RAM의 잔류자기 효과를 기반으로 실행 중인 메모리의 데이터에서 디스크 암 복호화 시 로드 되어 있는 DEK를 물리적으로 획득하여 FDE를 복호화 한다.

잔류자기 효과[11]는 휘발성 메모리에서 전류 공급이 차단되면 데이터가 즉시 소멸되지 않고 시간이 지남에 따라 점진적으로 소멸하는 것을 말하며, 온도가 낮을수록 소멸 속도가 늦다[12]. 일반 컴퓨터 RAM을 대상으로 하는 기존 Cold Boot와 달리 안드로이드 기기는 USB 부팅이 어려워 RAM의 데이터 손실이 불가피하며, 손실을 최소화하기 위해 Recovery 모드로 진입하여 데이터 복사해야 한다. 이때 Recovery 모드에서 데이터 복사가 가능하도록 Recovery Image를 수정한다.

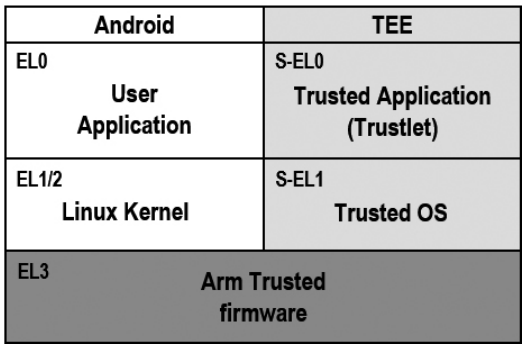
Müller 가 제시한 Cold Boot를 이용한 DEK 획득 방안은 다음과 같다.

① 안드로이드 기기 냉각
② 비정상적인 전원 공급 차단(배터리 분리)
③ 디바이스 부팅
④ Fastmode 진입
⑤ 수정된 Recovery Image 플래싱(주입)
⑥ 메모리 복사 및 DEK 획득

Recovery 모드는 펌웨어 업데이트 시 사용되는 부트 모드로 Recovery Partition(/Recovery)에 Recovery Image가 저장되어 있다. 수정된 Recovery Image를 플래싱하기 위해서는 Bootloader가 Unlocked 상태여야 한다. 또한 TEE가 도입된 기기부터는 DEK를 TEE 영역에 로드하므로 최신 기기에서 수행이 어렵다.

2.3. Qualcomm Arm Processor TEE Exploit

KEK 유도 중 포함되는 RSA sign(그림 1)은 TEE에서 동작하는 Keymaster에서 수행된다. Keymaster는 TEE OS 상에서 동작하는 애플리케이션(Trustlet) 중 하나로 안드로이드 내 키 생성, 서명, 인증을 담당하며 모든 작업에 하드웨어 기반의 Keymaster Key를 사용한다. TEE를 적용한 ARM 계열 Service On Chip(SoC)의 프로세서 아키텍처는 그림 4와 같다[13].

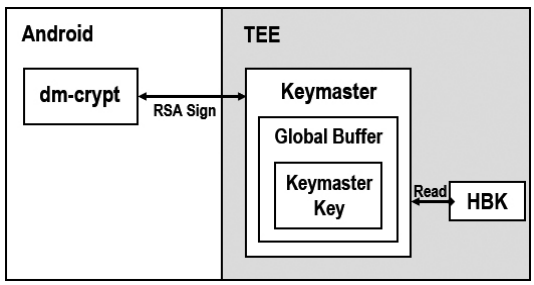


(그림 4) TEE가 적용된 ARM 계열 SoC 프로세서 아키텍처

Keymaster Key는 TEE Processor로만 접근할 수 있는 하드웨어 영역에 저장되어 있는 Hardware Bound Key(HBK)로부터 유도되고 그림 5와 같이 Keymaster 모듈 내 Global Buffer에 로드된다. 그림 5의 dm-crypt는 FDE 수행을 담당하는 모듈이다.

즉, RSA Sign 과정을 재현하려면 Keymaster 모듈 내 Global Buffer에 로드된 Keymaster Key를 획득해야 한다.

이번 장에서는 Qualcomm Arm Processor의 TEE에서 취약점을 분석하고 이를 이용한 Keymaster Key 획득 및 FDE 복호화에 관한 연구를 살펴본다.



(그림 5) RSA Sign 요청 시 Keymaster 내 Global Buffer에 로드되는 Keymaster Key

2.3.1. Mediaserver 및 Widevine 취약점(CVE-2016-2431)

Mediaserver는 리눅스 커널 모듈로 미디어 재생, 오디오 볼륨 조절, 녹음, 카메라 실행 등 미디어 관련 작업을 담당한다. Mediaserver는 영상 및 음성 데이터를 출력하기 전 데이터 검증을 위해 Output Device를 핸들링하는 AudioOuputDescriptors를 패치한다. 이 과정 중

```
5972  void AudioPolicyManager::AudioOutputDescriptor::changeRefCount(audio_stream_type_t stream,
5973                                                                  int delta)
5974  {
5975      // forward usage count change to attached outputs
5976      if (isDuplicated()) {
5977          mOutput1->changeRefCount(stream, delta);
5978          mOutput2->changeRefCount(stream, delta);
5979      }
5980      if ((delta + (int)mRefCount[stream]) < 0) {
5981          ALOGW("changeRefCount() invalid delta %d for stream %d, refCount %d",
5982                delta, stream, mRefCount[stream]);
5983          mRefCount[stream] = 0;
5984          return;
5985      }
5986      mRefCount[stream] += delta;
5987      ALOGV("changeRefCount() stream %d, count %d", stream, mRefCount[stream]);
5988  }
```

(그림 6) Mediaserver의 changeRefCount 함수

Output 데이터 타입의 사용 횟수를 카운트하는 ChangeRefCount 함수가 호출 시 파라미터인 stream의 값이 0x7FFFFFFF 이상일 때 그림 6의 5986번 라인에서 stream 값을 시스템 로그에 쓰는 버그가 존재한다[14]. 이 버그를 이용하여 시스템 로그를 쓸 때 획득하는 쓰기 권한을 탈취하면 원하는 코드를 시스템 내에 작성할 수 있다.

Widevine은 Google이 인수한 DRM 시스템으로 TEE에서 동작하는 Trustlet이다. 이 애플리케이션은 Android의 서비스 요청을 처리하는 과정 중 요청 메시지의 일부 값을 검증하는 함수에서 오버플로우가 발생할 수 있다. 그림 7은 검증을 담당하는 함수인 PRDiagVerifyProvisioning의 내용으로 (1)에서 fourth dword 값이 0이면 (2)를 실행하게 되면서 third dword 만큼 오버플로우가 발생한다[15].

Mediaserver 및 Widevine 취약점을 이용한 Qualcomm의 Keymaster Key 획득 과정은 다음과 같다.

① Mediaserver 취약점을 이용한 TEE 접근 권한 획득

② Widevine을 이용한 쓰기 권한 획득 및 Keymaster

```
(1) else if ( fourth_dword )
    {
      memzero(global_data + 5628, 0x80u);
      pStr = strncpy(global_data + 5628, global_data + 4604, 128);
      len1 = strlen(global_data + 4348);
      len2 = strlen(global_data + 4604);
      v10 = global_data + 5628;
      memcpy((void *)(len2 + global_data + 5628), (void *)(global_data + 4348), len1);
      v11 = pStr + strlen(global_data + 4348);
      *(_BYTE *)(global_data + 5628 + v11) = 47;
      v12 = strlen(global_data + 4348);
      memcpy((void *)(global_data + 5628 + v11 + 1), (void *)(global_data + 4348), v12);

                    ⋮

  else
  {
    strlen(global_data + 0x11FC);
    memzero(global_data + 0x10FC, 0x80u);          (Fully Controlled Overflow)
(2) memcpy((void *)(global_data + 0x10FC), pTzMsg + 4, third_dword);
  }
```

(그림 7) Widevine의 PRDiagVerifyProvisioning 함수 일부

의 Global buffer를 일반 애플리케이션이 접근 할 수 있도록 Shared buffer에 복사하는 Shellcode 삽입

③ System call table의 qsee_hmac 명령의 mapping 주소를 shellcode의 주소로 변경

④ 사용자 애플리케이션을 통해 Keymaster 모듈로 Encryption key 및 HMAC key 요청 (qsee_hmac 명령 실행)

⑤ Shared buffer에 접근하여 복사된 Keymaster Key 획득

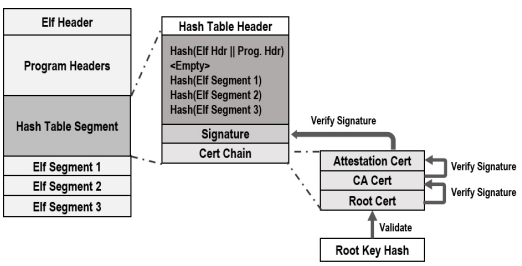
Mediaserver 및 Widevine 취약점은 2016년 5월 1일 보완되어 이후 안드로이드 기기에 대한 적용이 어렵다.

2.3.2. Trustlet 업데이트 취약점

Qualcomm Trustlet은 TEE에 적재되기 전 무결성을 보장하기 위해 해시값 및 서명 검증을 거치며, 이때 인증 체인과 하드웨어 기반의 Root Key가 사용됨으로써 안전성을 높인다. 만약 파일을 변조할 시, 검증에 필요한 모든 인증서뿐 아니라 하드웨어 수정이 필요하다.

Qualcomm Trustlet은 파일 시스템에 4개의 파일로 분할되어 있다가 메모리로 로드되기 전 버퍼에서 재조립된 후 검증과정을 거친다[16]. ELF 파일 형식을 따르는 Trustlet의 구조 및 검증과정은 그림 8과 같다.

먼저 프로그래밍 가능한 비휘발성 메모리인 eFuse에 쓰인 Root Key의 해시값으로 Root Certification이 검증된 후 인증 체인이 검증된다. 이러한 인증 체인의 마지막 인증서인 Attestation Certification은 Trustlet의 버전 관리를 위한 SW_ID 값을 포함한다. SW_ID의 상위 32bits는 버전, 나머지 32bits는 식별 값으로 구성되고 이는 취약점 및 버그가 존재하는 이전 버전으로의 롤백을 방지하기 위한 용도로 업데이트 시 갱신되어야 한다.



(그림 8) Client-Server 형 애플리케이션을 이용한 RSA Sign 동작 과정

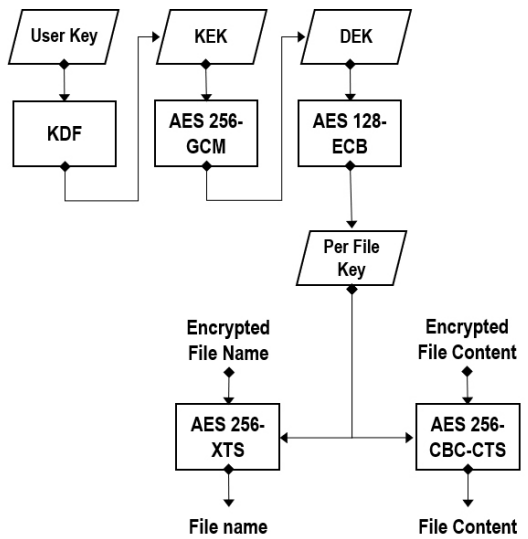
SW_ID를 갱신하려면 모든 인증서뿐 아니라 eFuse에 쓰인 Root Key를 수정해야 하므로 절차가 까다로워 Trustlet을 업데이트 시켜도 해당 필드를 수정하는 벤더가 적다. Gal Beniamini[13]의 연구에 따르면, 안드로이드 펌웨어 이미지 45개 중 44개에서 Widevine의 Version 필드 값이 0으로 확인되었다.

즉 Mediaserver 및 Widevine 다운그레이드 시 인증 및 동작에 문제가 없어 이전 취약점을 활용하여 Keymaster key를 획득할 수 있다.

Ⅲ. File Based Encryption (FBE)

FBE는 /data를 파일 시스템 수준에서 암호화로 여러 키를 사용하여 여러 파일을 암호화할 수 있어 2 명 이상의 사용자를 동시에 보호한다[4]. 또한 /data를 사용자 인증 완료 전과 후에 접근할 수 있는 Device Encrypted(DE) Storage 와 사용자 인증 완료 후에만 접근 가능한 Credential Encrypted(CE) Storage로 분할하여 다이렉트 부트 시 일부 기능을 이용할 수 있다.

FBE는 FDE와 동일하게 /dev/urandom에서 읽은 512bits의 DEK에서 상위 256 bits는 파일 이름, 하위 256bits는 파일내용을 암호화하는 데 사용한다. 또한 파일 내용은 AES256-XTS로 암호화되고 파일 이름은



(그림 9) File Based Encryption(FBE) 복호화 과정

AES256-CBC-CTS로 암호화 된 후, Base62 로 인코딩 되어 저장한다. KEK는 화면 잠금 키와 동일한 유저 키에서 유도되고 이 과정에 Keymaster가 관여하여 RSA Sign을 수행한다. FBE의 복호 과정은 그림 9와 같다.

3.1. 메타 데이터를 활용한 사용자 행위 유추

안드로이드 파일 시스템인 Ext4의 중요한 기본 요소 중 하나인 inode는 파일 및 폴더의 메타 데이터를 저장하는 구조체이다. inode에는 수정 시간, 접근 시간, 생성 시간, 크기, 권한 정보, Generation ID, 파일의 확장된 속성 등이 저장된다. FBE에서 파일 암호화 시, 파일 이름 및 내용은 암호화되나 메타 데이터를 포함하는 inode는 평문으로 저장된다. 이러한 파일 및 폴더의 inode 내 메타 데이터를 이용하면 사용자의 행위를 유추할 수 있다.

사용자의 행위에 따라 생성 또는 수정되는 파일의 개수가 다르고 대부분 동일 디렉토리 내에 존재하므로, 특정 행위에 따른 메타 데이터 변경에 패턴이 존재할 수 있다. Groß 외 2인[17]은 메타 데이터를 이용하여 설치된 애플리케이션 리스트를 추출하였고, WhatsApp에서 9가지 사용자 행위에 대한 패턴을 확인하였다.

하지만 Android 9.0 이상부터 Metadata 암호화를 지원하면서 최신 안드로이드 기기에 대한 적용은 어렵다[18].

Ⅳ. 결 론

본 고는 안드로이드에서 FDE와 FBE의 동작 과정 및 특징을 정리하고 이를 복호화하기 위한 기존 연구들을 소개했다. FDE 복호화 연구는 크게 Userkey Brute force, Cold boot, Qualcomm TEE Exploit 3가지로 분류할 수 있었고 FBE는 파일들의 메타 데이터 패턴을 활용한 사용자 행위를 유추만 가능했다. 또한, 구글은 FDE 및 FBE의 안전성 연구 결과들을 지속해서 반영함에 따라 최근 출시되는 안드로이드 디바이스에 대한 적용이 어려움을 확인했다.

하지만 디지털 포렌식 수사 과정에서 FDE와 FBE는 수사관이 증거 분석은 물론 증거 데이터에 대한 접근 자체를 어렵게 하므로 복호 방안 마련은 중요하다. 특히 최근 도입되기 시작한 FBE는 암호화 프로세스 및 복호 방안에 대한 연구 자료가 미흡하다.

구글은 계속해서 FDE와 FBE를 보완하고 있는 만큼 증거 데이터 획득을 위한 연구도 지속적인 노력이 요구된다.

참 고 문 헌

[1] Statcounter, “Mobile Operating System MarketShare Worldwide”, https://gs.statcounter.com/os-market-share/mobile/worldwide/#monthly-201301-201911

[2] Oliver Kunz, “Android full-disk encryption: asecurity assessment”, Royal holloway University of London, pp24, 2016

[3] AOSP, “Direct Boot”,https://developer.android.com/training/articles/direct-boot

[4] AOSP, “File-based Encryption”, https://source.android.com/security/encryption/file-based

[5] AOSP, “Full-Disk Encryption”, https://source.android.com/security/encryption/full-disk

[6] Ronan Loftus, Marwin Baumann, “Android 7 File Based Encryption and the Attacks Against It”, University of Amsterdam, pp.33, 2017

[7] J. Engler and P. Vines. “Electromechanical PIN Cracking Implementation and Practicality”, Defcon 21, 2017

[8] Google Git, “Gatekeeper.cpp”, https://android.googlesource.com/platform/system/gatekeeper/+/master/gatekeeper.cpp

[9] The Linux Kernel, "Ext4 Filesystem-Data Structures and Algorithms", https://www.kernel.org/doc/html/latest/filesystems/ext4/

[10] T. Müller, M. Spreitzenbarth and F. C. Freiling. “FROST: Forensic Recovery of ScrambledTelephones”. Tech. rep. Department of Computer Science, Friedrich-Alexander Universityof Erlangen-Nuernberg, p. 19, Oct. 2012

[11] Peter Gutmann, “Data remanence in semiconductor devices”, In Proceedings of the 10th conference on USENIX Security Symposium-Volume 10, USENIX Association, pp.4. 2001

[12] J. Alex Halderman, Seth D. Schoen† , Nadia Heninger, William Clarkson, William Paul, “Lest We Remember: Cold Boot Attacks on Encryption Keys”, 2008 USENIX Security Symposium, pp.49, 2018

[13] Gal Beniamini, “Trust Issues: Exploiting TrustZone TEEs”, https://googleprojectzero.blogspot.com/2017/07/trust-issues-exploiting-trustzone-tees.html

[14] Gal Beniamini, “Android privilege escalation to mediaserver from zero permissions”, http://bits-please.blogspot.com/2016/01/android-privilege-escalation-to.html

[15] Gal Beniamini, “QSEE privilege escalation vulnerability and exploit (CVE-2015-6639)”, http://bits-please.blogspot.com/2016/05/qsee-privilege-escalation-vulnerability.html

[16] Qualcomm ,“Secure Boot and Image Authentication”, https://www.qualcomm.com/media/documents/files/secure-boot-and-image-authentication-technical-overview.pdf

[17] Groß, Tobias, Matanat Ahmadova, and Tilo Müller. "Analyzing Android's File-Based Encryption: Information Leakage through Unencrypted Metadata." Proceedings of the 14th International Conference on Availability, Reliability and Security. ACM, 2019

[18] AOSP, “Encryption-Metadata encryption”, https://source.android.com/security/encryption

〈저자소개〉

서 승 희 (Seunghee Seo)

학생회원

2017년 2월 : 서울과학기술대학교 컴퓨터공학과 학사

2019년 2월 : 서울과학기술대학교 컴퓨터공학과 석사

<관심분야> 정보보호, 디지털 포렌식

이 창 훈 (Changhoon Lee)

종신회원

2001년 : 한양대학교 자연과학부 수학전공 학사

2003년 : 고려대학교 정보보호대학원 석사

2008년 : 고려대학교 정보경영전문대학원 정보보호전공 박사

2008년 4월~2008년 12월 : 고려대학교 정보보호연구원 연구교수

2009년 3월~2012년 2월 : 한신대학교 컴퓨터공학부 조교수

2012년 3월~2015년 3월 : 서울과학기술대학교 컴퓨터공학과 조교수

2015년 4월~현재 : 서울과학기술대학교 컴퓨터공학과 부교수

<관심분야> 정보보호(Personal Information), 암호학(Cryptography), IoT(Inter-net of Things), 디지털포렌식(Digital Forensics) 등