https://www.browserstack.com/android-emulators

여기 보면 안드로이드 에뮬레이터 상에서 파일 시스템이랑 실제 기기 파일시스템이랑 일치하지 않을 수 있다고 해서 일단 기기로 직접 해보면 좋을 것 같아요

그리고이논문내용개어려워서읽었는데도뭔말인지잘모르겠어요

물리적 차이 활용: 메모리 암호화에 대한 사이드 채널 공격

<요약>

dm-crypt, 물리적 공격 측면에서 ext4 분석

차등 전력 분석(DPA), 차등 결함 분석(DFA)가 최신 메모리 및 디스크 암호화 방식을 무력화함

Zynq-7010 시스템 온 칩의 EM 사이드 채널은 ext4 디스크 내용 노출 가능

<실제 메모리 암호화>

Definition 1. *A memory encryption scheme is an encryption scheme* $Enc : \mathcal{K} \times \mathcal{A} \times \{0,1\}^n \to \{0,1\}^n$, *which*

- *uses a key* K *from key space* $\mathcal{K}$, *and*
- *splits the memory into* $s = \lceil \frac{size_{memory}}{n} \rceil$ n*-bit memory blocks,*
- *identifies each of the memory blocks by their address in address space* $\mathcal{A}$, *and*
- *provides address-dependent en-/decryption for each of these memory blocks.*

1. 블록 장치 또는 전체 디스크 암호화 2. 파일 수준 디스크 암호화

전체 디스크 암호화가 전체 디스크, 블록 장치 또는 파티션의 원시 메모리 공간에서 직접 암호화를 수행하는 반면 파일 수준 디스크 암호화는 파일 시스템 위 또는 파일 시스템 내에서 파일 수준으로 암호화를 수행

➔ 동일한 메모리 암호화 체계를 사용하나 적용하는 곳이 다름

키 파생 함수(KDF)를 사용하여 메모리 암호화 방식에서 사용할 데이터 암호화 키(DEK)를 사용자 비밀번호와 공개 논스 등에서 도출함

주요 파생 함수:

사용자 비밀번호 또는 PIN에서 키를 유도하기 위해 일반적으로 PBKDF2또는 scrypt와 같은 비밀번호 해싱 함수가 사용.

이 비밀번호 유도 키는 주로 일반 블록 암호를 사용하여 메모리의 실제 마스터 키 MK를 복호화하는 키 암호화 키(KEK)로 사용됨

마스터 키 MK는 메모리 암호화 체계의 DEK로 직접 사용되거나, 단일 파일의 파일 수준 디스크 암호화를 위한 DEK와 같은 키를 추가로 유도하거나 복호화하는 데 사용

암호화 체계:

일반적으로 외부 메모리의 암호화만 다룸. 예를 들어, dm-crypt에서 주로 XEX, XTS, ES SIV가 적용

XEX와 XTS-> 조정 가능한 블록 암호

두 가지 암호화 모드는 암호화된 섹터 번호와 메모리 블록 주소의 이진 필드 곱셈에서 비롯된 암호 E에 조정 T를 적용

XEX는 단 하나의 키만 사용, XTS는 암호의 두 인스턴스에 서로 다른 두 개의 키를 사용

ESSIV는 비밀 IV를 보장하여 워터마킹 공격을 방지. 섹터 번호를 솔트로 암호화해 IV를 계산

<최신 구현> 여기부터 중요한듯

안드로이드:

전체 디스크 암호화 제공. 안드로이드 5.0에서 블록 장치의 암호화는 AES-128 및 CBC-ESSIV를 사용하도록 구성된 dm-crypt를 기반. DEK는 128비트 크기로 설정되며 블록 장치에 암호화되어 저장

각각의 KEK는 사용자 비밀번호와 하드웨어에 바인딩된 키를 스크립트와 신뢰 실행 환경(TEE) 내의 서명 절차를 사용하여 파생됨

Ext4:

파일 수준 디스크 암호화 제공. 사용자의 passphrase와 솔트를 사용하여 PBKDF2로 만들어진 마스터 키가 할당된 특정 폴더에 대한 암호화를 설정 가능. ext4는 콘텐츠와 이름을 암호화하지만, 메타데이터와 파일 시스템 구조는 일반 텍스트로 제공

각 파일은 ECB 모드에서 AES-128 사용하면서 마스터 키(MK)와 파일 논스에서 파생된 개별 DEK 사용 (각각의 논스는 파일의 메타데이터 섹션에 저장됨)

파일 DEK은 XTS 및 AES-128을 사용하여 파일 내용을 암호화하는 데 사용

3 Physical Attacks on Memory Encryption 부분은 도저히 이해가 안 돼서 패스.. ㅠ

<Ext4 암호화에 대한 EM공격>

<ext4 디스크 암호화 분석>

EM Trace는 암호화 장치가 연산을 수행할 때 발생하는 **전자기파 신호**를 시간 경과에 따라 기록한 데이터

EM 공격은 AES 실행의 첫 번째 라운드에서 누출된 정보를 이용하여 사용된 마스터 키와 따라서 모든 콘텐츠를 드러내는 공격

마스터 키 MK는 개별 데이터 암호화 키 DEK를 유도하여 해당 파일의 콘텐츠와 이름을 암호화하는 데 사용

키 파생은 공개 파일 논스를 키로 사용하여 AES-128 ECB 모드로 MK를 암호화함으로써 수행. DEK가 필요하며 이미 메모리에 존재하지 않을 때 수행됨

MK, DEK 크기는 모두 512비트(AES-256 에서도 작동하도록). 그러나 현재 AES-128을 사용하므로 각각 마지막 256비트는 사용되지 않음

파일 논스인 Nf는 파일 inode의 확장된 속성에 저장됨

분명히 마스터키와 공개된 논스가 주어지면 각 DEK가 나옴. 그러나 ext4에서 선택된 주요 파생 방식은 DEKf와 해당 넌스 Nf가 주어졌을 때 마스터 키 MK를 계산할 수 있도록 함

그러므로 전력 분석으로 MK를 얻으려는 공격자는 1. 파일 내용의 데이터 암호화 2. DEK의 파생을 노릴 수 있음

디스크 암호화 공격의 실현 가능성을 실제로 검증하기 위해 논문에서는 2번 선택

<공격 흐름>

(리눅스 환경) Ext4를 사용하는 SD카드의 암호화된 폴더를 가정함. 공격자가 실행중인 웹

서버를 통해 데이터를 업로드하는 등 암호화된 폴더 내에 새 파일을 생성할 수 있다고 가정함.

1. SD카드의 암호화된 내용을 덤프 -> 메타데이터는 평문이므로 구조를 알 수 있음
2. SD카드에 충분한 수의 파일을 생성하고, EM 사이드 채널 관찰, 각각 EM 추적 정보를 저장
3. 다시 SD카드 덤프 -> 초기 내용과 비교해 어떤 파일이 생성되었는지 알아낼 수 있음. 새로 생성된 파일의 메타데이터를 통해 사용된 논스, 생성 날짜를 알아낼 수 있음. 이는 새로 생성된 파일을 SD카드에서 EM흔적을 매핑할 수 있게 함
4. 공격자는 키 파생을 위한 전력 모델을 생성 (DEKf = ENf (MK))
5. 전력 모델이 EM 추적과 일치하여 마스터 키를 밝힘

암호화된 디렉터리 접근 -> 디버깅, 포렌식 도구 사용 (도구 debugfs를 사용하여 파일 시스템에서 새로운 파일을 찾고 그 생성 날짜와 해당 논스를 확인)