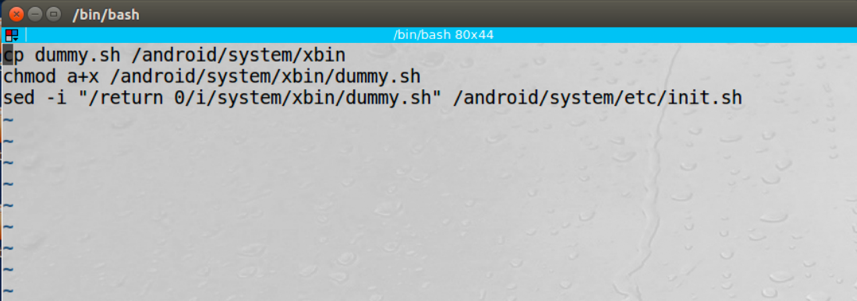
이번 실습은 팀과제로 진행되어, 전반적인 실습 내용을 먼저 설명하고 자세한 분석 내용은 뒷부분에서 다루었다.

**Task 1**

1번 과제는 recoveryOS를 통해 update-binary를 실행하여 androidOS의 /system/ 디렉토리에 우리가 원하는 코드(exploit)를 작성하여 전송해보는 실습이다. Dummy.sh에는 /system/dummy을 만들고 그 파일에 hello를 출력하는 쉘코드이며, update-binary는 이 dummy.sh 파일을 안드로이드OS에서 우리가 원하는 디렉토리로 dummy.sh를 복사한 뒤, 실행 권한을 부여한다. 마지막으로 안드로이드OS가 시작할 때 실행된는 init.sh에 해당 dummy.sh 안의 코드를 복사하여 자동으로 실행되게끔 수정하는 파일이며, 2,3번 실습에서도 쓰이는 이번 과제의 핵심 파일이다. 1번 실습에서는 안드로이드OS의 rooting을 위해 필요한 exploit의 전반적인 구조와 flow를 이해하기 위한 의도로 해석했다.



**Update-binary**

코드 작성은 기존 SEEDOS에서 진행하여 scp를 통해 RecoveryOS로 전달하도록 하였다. 아래와 같은 구조로 된 디렉토리를 zip으로 압축시켜 scp를 통해 /tmp/ 디렉토리로 전송하여 unzip을 한다. 마지막으로 sudo 권한으로 update-binary를 실행한 후, 재부팅을 하여, 터미널앱을 통해 /system/ 디렉토리를 확인하면 dummy 파일에 hello가 적혀있는 걸 확인할 수 있다.

스크린샷이(가) 표시된 사진

자동 생성된 설명스크린샷이(가) 표시된 사진

자동 생성된 설명

**Zip otaP1 & Unzip otaP1.zip**

스크린샷이(가) 표시된 사진

자동 생성된 설명스크린샷이(가) 표시된 사진

자동 생성된 설명

**Unzip otap1 and run update\_binary & Result**

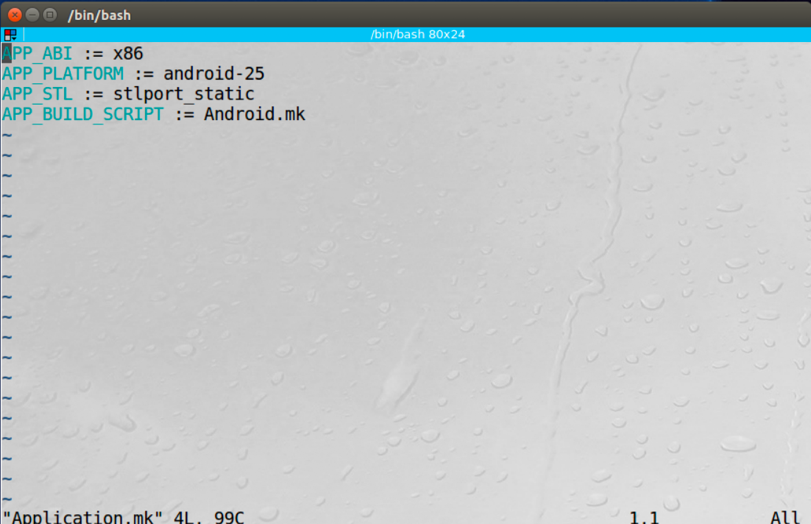
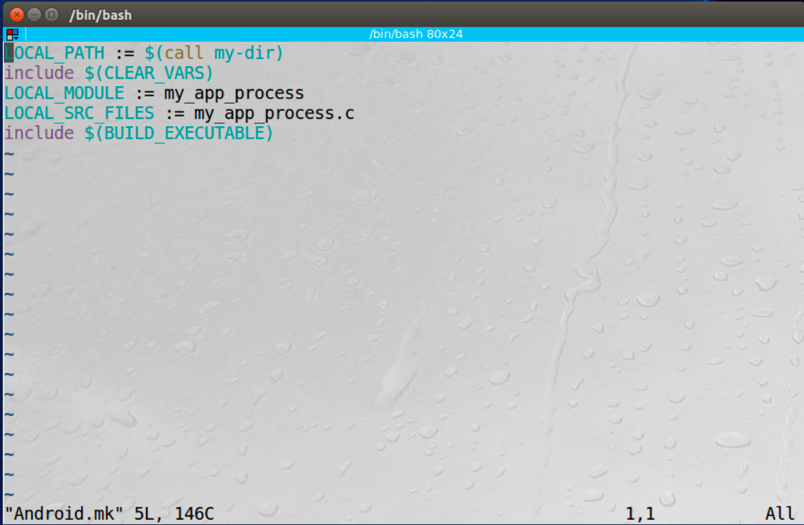
**Task 2**

1번 실습은 안드로이드OS가 시작할 때 실행하는 init.sh 파일에 우리가 원하는 실행 코드를 입력하여 실행하도록 의도했다. 2번 실습은 안드로이드OS가 항상 실행하는 프로그램인 my\_app\_process 라는 프로그램을 root 권한으로 실행한다. 이 프로그램은 모든 앱 프로세스의 부모 프로세스이며 어플리케이션을 실행하는 zygote daemon을 실행한다. 2번 실습은 이 my\_app\_process를 수정하여 zygote 프로세스를 실행함과 동시에 우리가 원하는 코드를 실행하게 만드는 것이다. 2번 실습 역시 1번 실습과 마찬가지로 정해진 폴더 구조로 OTA(Over-The-Air) 패키지를 만든다. Update-binary 파일은 우리가 컴파일하여 생성한 파일을 새로운 my\_app\_process를 새로 만든 app\_process\_original에 옮긴 후, root 권한으로 dummy2 파일을 생성한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

**my\_app\_process.c**



**Android.mk & Application.mk**

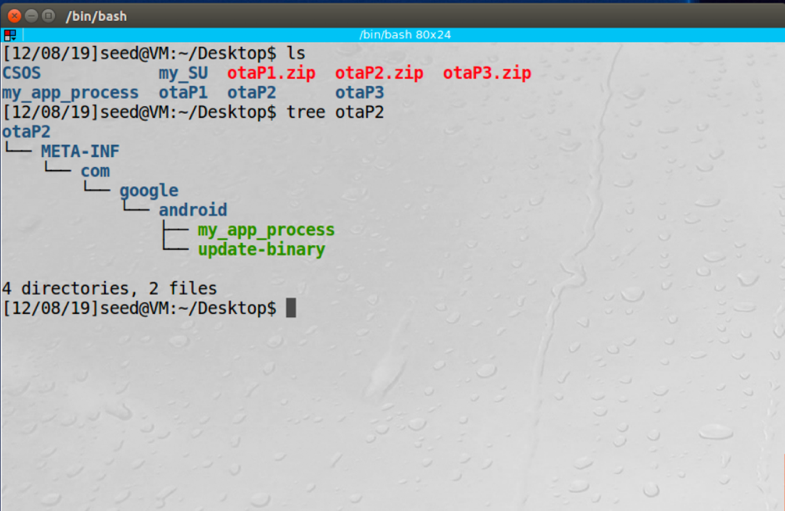
2번 실습은 아래와 같이 컴파일한 my\_app\_process와 실습1에서 사용한 update-binary를 아래에 출력된 구조와 같이 넣은 뒤 압축하여 scp로 동일하게 전송을 한 뒤, sudo 권한으로 update-binary를 실행하면 root 권한으로 생성된 dummy2가 생성된다.

텍스트, 신문이(가) 표시된 사진

자동 생성된 설명텍스트, 신문이(가) 표시된 사진

자동 생성된 설명

**Compile my\_app\_process.zip & my\_app\_process in compiled result**

 스크린샷이(가) 표시된 사진

자동 생성된 설명

**otaP2 Dir Tree & Run Otap2 update-binary on Recovery OS**

스크린샷이(가) 표시된 사진

자동 생성된 설명

**Otap2 Result**

**Task 3**

실습 3 역시, update-binary는 실습1,2와 동일한 형태이며, 다운받은 소스코드를 컴파일하여 나온 mysu와

mydaemon을 복사 해준 뒤, mydaemon 내용을 init.sh에 넣어 실행시킨다. 유저가 root 권한을 가진 쉘을 가지기 위해는 root daemon에 요청을 보내야한다. 이 때, root daemon이 쉘을 실행하고 그걸 클라이언트(유저)에게 돌려준다. 이를 통해 유저가 root 권한을 가질 수 있다. 결국 쉘 프로세스를 제어하기 위해서는 쉘의 표준 입력, 출력 장치를 제어할 수 있어야 하지만, 쉘 프로세스는 생성될 때 표준 입력, 출력 장치를 상속한다.

루트 권한의 프로세스는 일반 유저 프로그램으로 제어가 불가능하기 때문에, 유저 프로그램의 입력, 출력을 쉘 프로세스에서 사용하게 설정하면 사용자는 쉘 프로세스를 제어할 수 있다.

스크린샷, 모니터이(가) 표시된 사진

자동 생성된 설명

**Update-binary**

텍스트이(가) 표시된 사진

자동 생성된 설명

**Compile file**

스크린샷이(가) 표시된 사진

자동 생성된 설명

**otaP3 Dir Tree**

텍스트, 신문이(가) 표시된 사진

자동 생성된 설명스크린샷이(가) 표시된 사진

자동 생성된 설명

**Zip otaP3 & send, unzip otaP3 on recoveryOS**

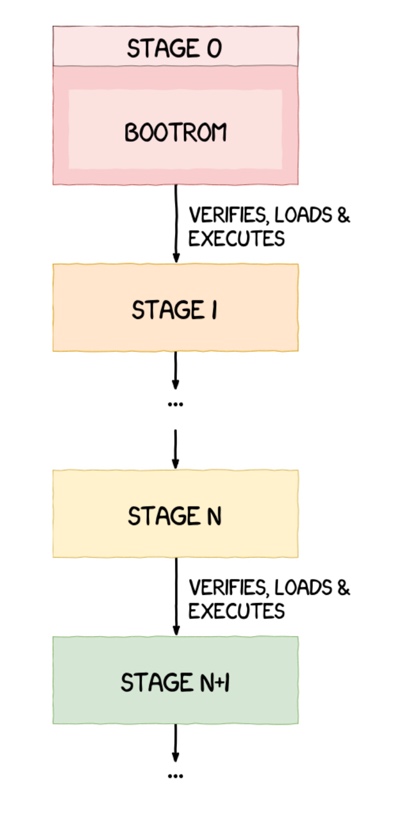
스크린샷이(가) 표시된 사진

자동 생성된 설명

**otaP3 result**

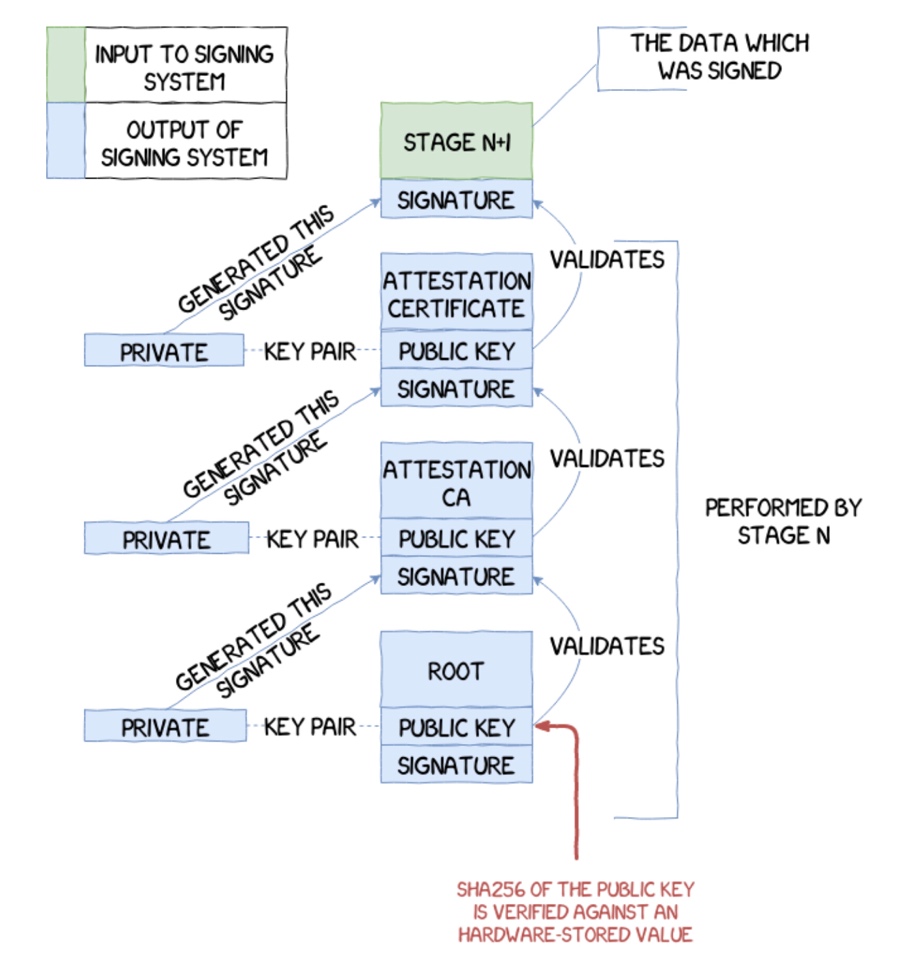
**과제 분석**

전반적인 Low level의 Booting Process는 다음과 같다.

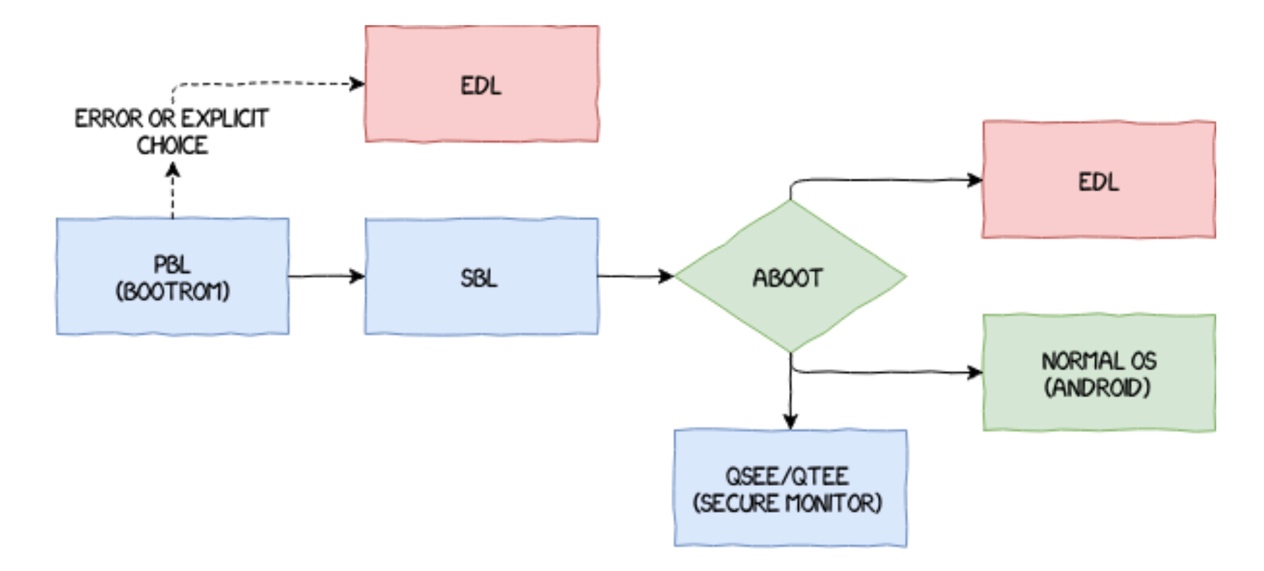


세부적인 과정은 기기 내부의 Chipset 제조사에 따라 달라진다.

Qualcomm의 경우엔 대략적으로 다음의 과정을 거치게된다.



우선, 처음에는 Vendor에서는 root certificate를 가지고 있으며, bootROM 내부에는 public key가 저장된다. 그리고 이 public key는 특정 hardware 내부에 hash 값으로 저장되며, 쉽게 tampering 할 수 없다. 이러한 certificate는 여러 개가 존재하며, 각각 다음 단계의 certificate를 sign하게 된다.



Qualcomm secure boot chain은 다음과 같다.

* bootROM은 Primary Boot Loader (PBL) 이라고도 불리며, CPU에 의해 제일 먼저 실행되는 코드를 가지고 있다.
* 그 다음 과정에서는 Secondary Boot Loader (SBL)이 실행되는데, 보통은 SBL1 ~ 3까지 나누어진다. (지금은 eXetended Boot Loader라고 불리는 XBL로 바뀌었다)
* SBL은 QSEE나 QTEE로 불리는 Secure Monitor를 올리게 되는데, 이게 현재 Trusted OS라고 불리는 TrustZone 내부에서 동작하는 것이다. 그리고, Android Boot Loader (ABL)이라고 불리는 ABOOT을 올리며, 실행흐름을 ABOOT으로 넘겨주고, 실행 레벨을 Exception Level 1 (Ring 0)으로 조정한다.

EDL Mode에서는 해당 코드가 bootROM에서 실행되기 때문에, 이 상태에서 사용자가 특정 command를 보낼 때까지 기다리는 상태가 된다.

이 커맨드들 중 하나가 USB를 통해 signed image를 보내어, 부팅하는 것이다.

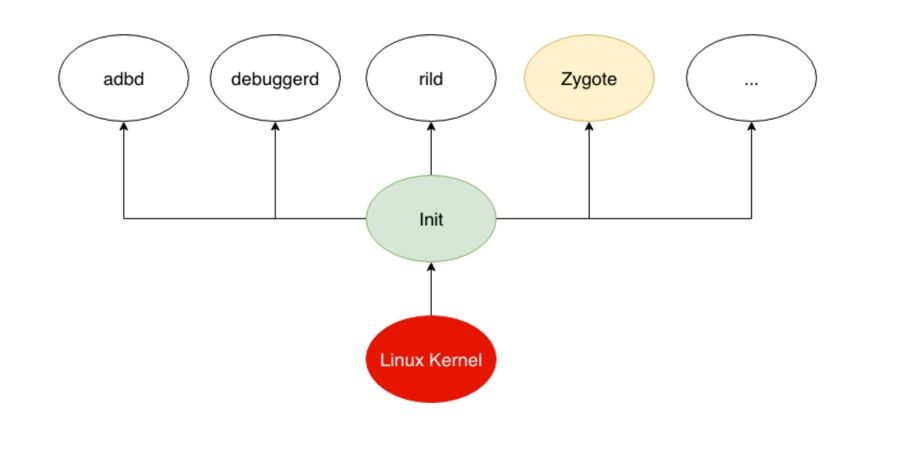
일련의 Boot chain이 진행된 후에는, 다음과 같은 것들이 실행된다.

1. Core Kernel initialization (Memory, I/O, Interrupts, etc...)
2. Kernel Driver initialization
3. /root file system mount
4. "init" process start

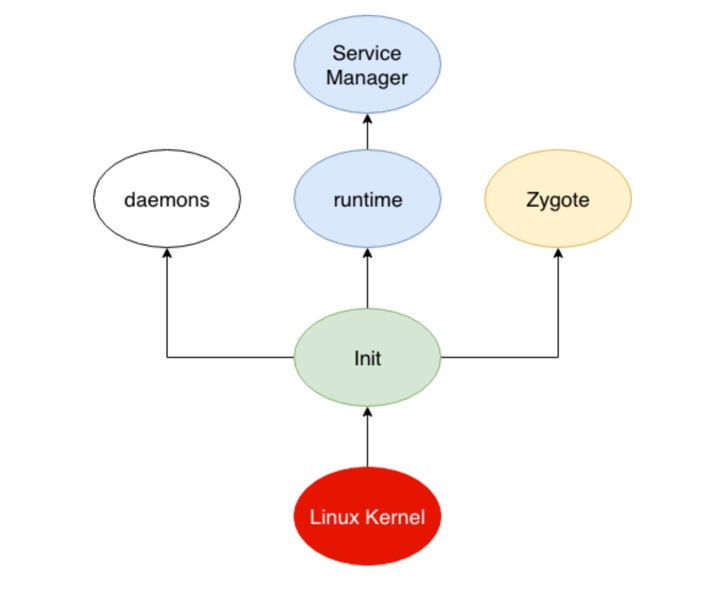
3번의 과정까지 진행되면, userspace에서 볼 수 있는 pid 1의 init process를 실행한다.

"init" process는 Android system의 핵심적인 요소들을 초기화하는 역할을 한다. 처음에는 모든 Android는 공통적으로 init.rc라는 스크립트 파일을 읽어서, 해석 후, 실행하게 된다. 그리고 그 다음에는 init.<machine\_name>.rc를 실행하는데, 이건 device specific하다. init.rc에서는 처음에 여러 daemon들을 실행시키는데 다음과 같은 것들이 있다.

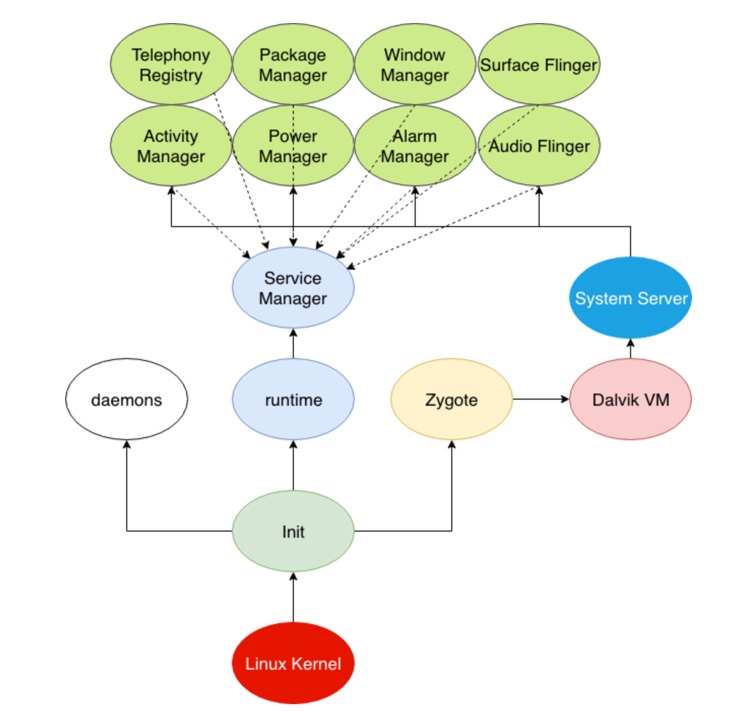
* Android Debug Bridge Daemon
* Debugger Daemon
* Radio Interface Layer Daemon
* ServiceManager



여러 데몬들을 초기화하고난 이후에는, Zygote라는 process를 실행하게된다. 본 과제에서 사용하는 app\_process는 사실 Zygote 그 자체라고 할 수 있다. 왜냐하면, Zygote의 바이너리 이름이 app\_process 이기 때문이다. 이 Zygote는 Android system에서 사실상 심장과 같은 역할을 한다고 볼 수 있다. 이 프로세스는 Android Application이 실행될 수 있도록, Android RunTime (ART)이나 Dalvik VirtualMachine (DVM) 환경을 세팅해주는 역할을 한다. 물론, 앱이 실행될 때, 이러한 요청은 내부 IPC 메커니즘에 따르지만, 이는 Booting과는 상관 없기 때문에, 여기서 다루지는 않겠다. "init"이 여러 데몬을 초기화한 이후, Zygote를 올린다고 했는데, 그 외에도 다른 runtime service들을 실행하는 역할을 한다. 전반적인 개요는 다음과 같다.



runtime은 Service Manager를 의미하는데, 이는 DNS와 비슷하게 Service Register / lookup을 도와주는 매개체 역할을 하게 된다. 그리고 이 Service Manager는 Android IPC 핵심인 Binder의 코어 역할을 한다. Service Manager를 실행하는 역할이 끝나면, Zygote에서 ART/Dalvik VM instance를 fork 해주며, System Server라는 것을 실행하게 된다. System Server가 관리하는 것은 다음과 같은 것들이 존재한다.

* Entropy Service
* Power Manager
* Alarm Manager
* ...

이 부분들을 정리하면 우측과 같이 요약할 수 있다.

**Rooting**

1차적으로 Rooting 이라는 것은 root privilege를 얻는 것을 의미하며, Samsung Galaxy나 Google Pixel 등에서는 KNOX나 SELinux와 같은 security component를 전부 무력화하는 과정까지 포함된다. 왜냐하면, root 권한을 가진다라고하여도, 또 다른 보안 요소에 의해, 기능이 제한될 수 있기 때문이다. 즉, 제한된 기능을 전부 해제하여, 모든 리소스에 접근가능한 상태가 되는 것이 Android에서는 Rooting이라고 부르며, iOS에서는 Jailbreak라고 부른다. 본 과제와 같이, 임의로 OTA를 통해 root 권한을 얻는 경우도 있지만, 연구적으로나 세계적으로 주목받는 것은 Kernel 컴포넌트나 Boot chain 취약점을 통해, rooting 하는 것이다.



**IOS jailbreak & Android rooting**

**Recovery OS**

위에서 설명한 PBL (BootROM)은 Recovery Mode로 Execution mode를 변경할 수 있는데, 이를 Emergency Download Mode (EDL) 이라고 한다. 해당 모드로는 다음과 같은 상황에서 접근가능하다.

* SBL이 손상되거나, 버전이 안맞을 경우
* 보드 상의 특정 테스트 지점에 결함이 있을 경우
* SoC에 명시된 특정한 키를 booting 시에 눌럿을 때

Recovery mode에서는 다음과 같은 역할들을 할 수 있다.

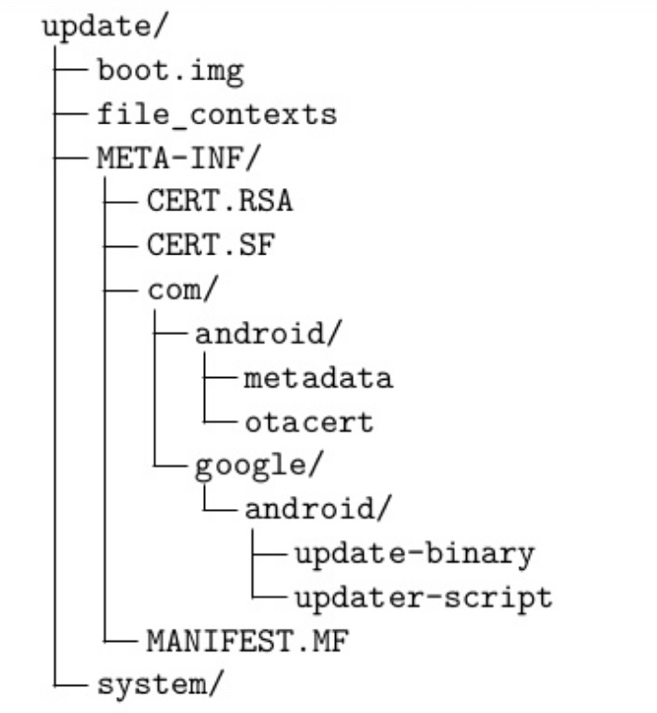
* Install from internal storage
  + 디바이스 내부에 저장된 zip파일을 Android system에 설치할 수 있다.
* Install from ADB
  + ADB를 통해 파일을 추가/삭제/수정 등을 할 수 있다.
* Wipe data and cache
  + 시스템 세팅 재설정
  + 캐시 비우기
  + 파일 전체 지우기

**OTA (Over The Air)**

Over The Air update는 무선 방식으로 업데이트 할 소프트웨어나 펌웨어를 가져와서 업데이트 하는 방식을 일컫는다. Android Recovery OS에서 업데이트 할 때도 이와 같은 방식을 취하는데, 순서는 다음과 같다.

1. update package를 다운로드 받는다. (Wireless)
2. Package가 유효한 지, 간단한 signature check를 하게 된다.
3. Recovery Console로 Reboot한다.
4. Package의 signature를 다시 체크한다.
5. update를 적용한다.
6. Android를 재부팅한다.
7. 마지막으로 Recovery update script를 실행한다.

Update package는 다음과 같은 구조를 가지고 있다.

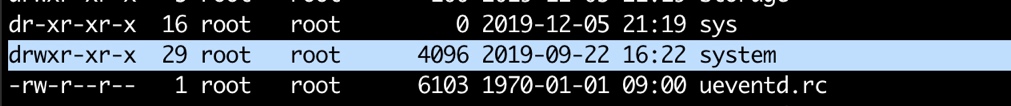


이게 full update archieve 인데, system, boot 그리고 recovery partition들에 대한 정보를 다 담고 있는 것이다.

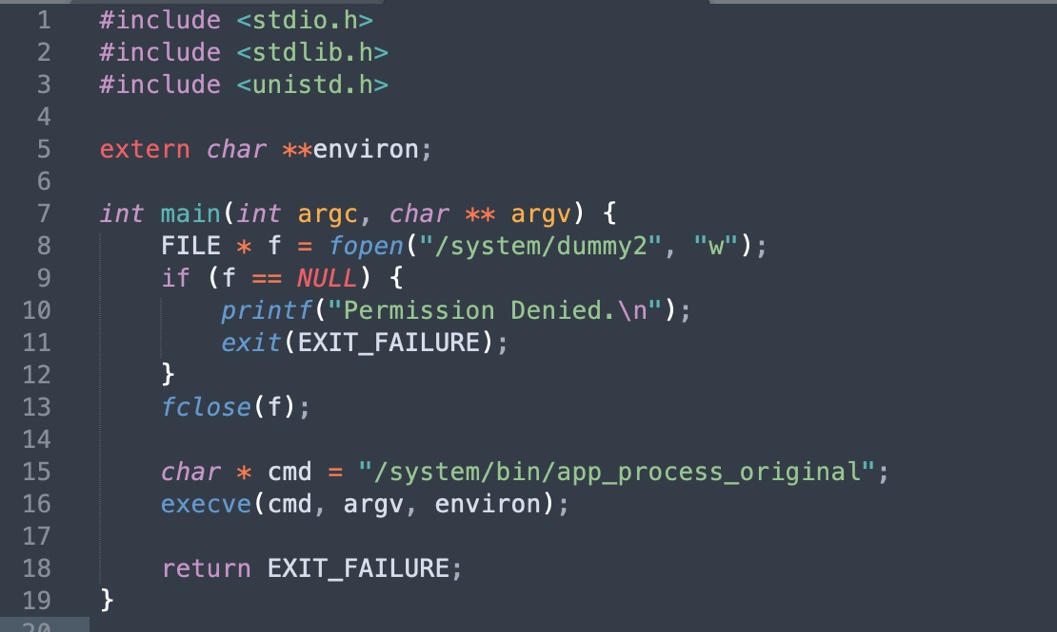
핵심적인 부분들만 살펴보면 다음과 같다.

* file\_context
  + SELinux label 정보들을 가지고 있는데, 이건 업데이트 이후에 적용된다.
* META-INF
  + APK 파일의 META-INF와 비슷한 역할을 하는 것이며,
* update-binary
* updater-script
  + OTA update의 메인 루틴이며, update-binary가 script를 올려서, 해석하고 실행하는 역할을 하게 된다.

**my\_app\_process**



Android root filesystem에서 system 디렉터리는 root 유저가 아니면, 임의로 쓸 수 없다.



my\_app\_process는 간단하게 /system/ 디렉터리에 임의 파일을 써넣는 것을 테스트하는 코드이다.

이 코드 자체는 OTA 과정에서 root permission을 가지고 진행되기때문에, 제대로 되었다면 가능하다.

**my\_SU (mydaemon)**



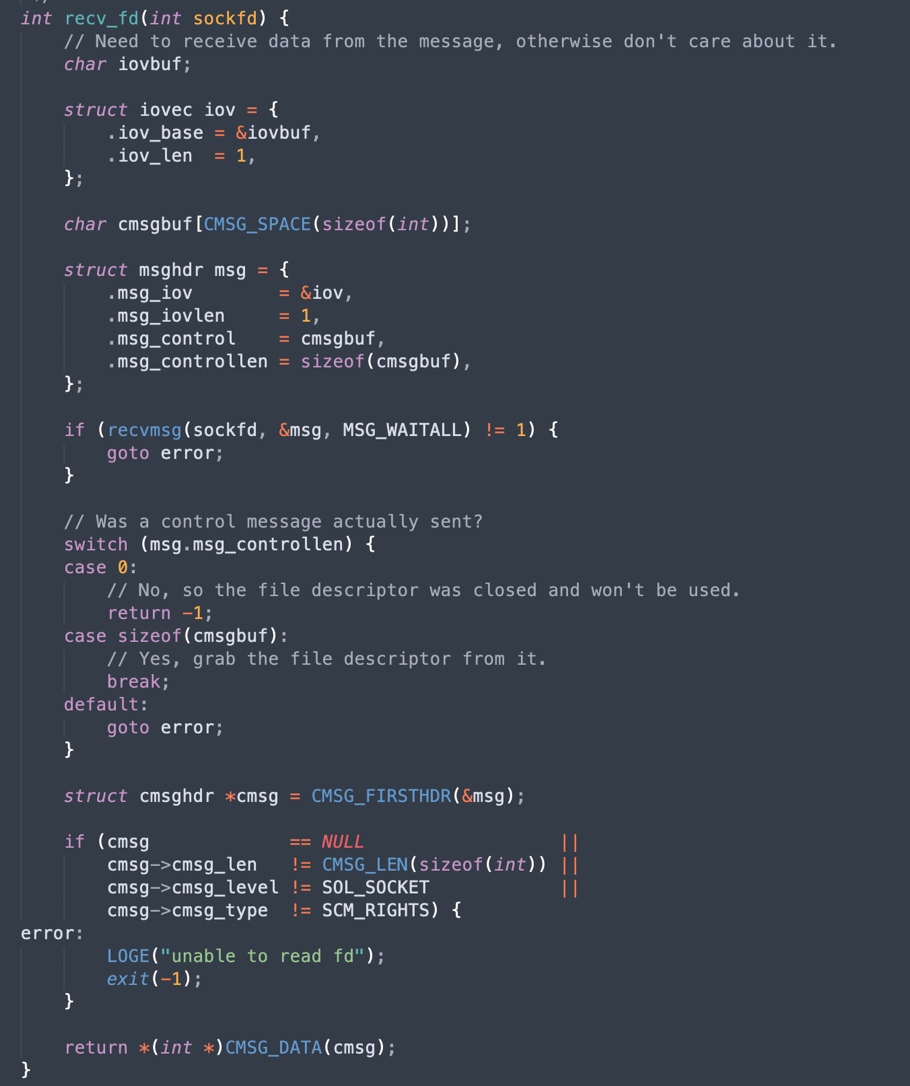
my\_SU의 daemon 파일도 실행은 간단하다. Parent process는 기존의 app\_process를 실행시키고, Child process가 실제 daemon의 역할을 하게 된다. detect\_daemon의 경우엔, /data/mydaemon/server에 socket connection이 이루어지면, daemon이 존재하며, 아닐 경우엔, run\_daemon 루틴으로 들어가게된다.



이 바이너리 또한, OTA 과정에서 root permission으로 진행되므로, uid check 루틴이 상단에 있는 것을 확인할 수 있다. /data/mydaemon/server에 Unix domain socket을 형성하여, 여기에 클라이언트 접속을 기다린다. client가 여기에 붙을 경우, child\_process 루틴이 실행된다.



daemon의 기본 특성상 표준 입출력과 관련된 것이 존재하면 안되는데, 이를 dup2 syscall을 통해, client와 socket 통신할 수 있도록, socket descriptor를 기존 표준 입출력에 덮어준다.



recv\_fd는 iovec를 통해 client-server 형태로 communication 하게 되는데, 통신에 사용할 file descriptor 1개만을 input으로 받게된다. 사실 이 경우에, 굳이 iovec을 사용할 이유가 없긴 하지만, 아무튼 client와 통신할 파일 디스크립터를 반환하는 역할을 한다. 그리고 "/system/bin/sh"을 실행시켜주는데, 기본 입출력이 client의 파일 디스크립터로 변경된 상태이므로, 클라이언트에 interactive root shell이 떨어지게 된다.

**my\_SU (mysu)**



client는 동일하게 unix domain socket에 연결한 후에, handshake 과정을 거치는데, handshake는 그냥 랜덤 값을 보내놓고, 동일한 값이 리턴되는지 확인하는 것이다.

정상적으로 동작했다면, daemon에서 I/O redirection과 dup가 진행되었으므로, root 권한을 가지게되는데, main에서 마지막에 execve를 통해 쉘을 띄워줌으로써, 완전히 유저에게 interactive root shell을 쓸 수 있게 해준다.

**한계점**

이번 과제를 진행하며 안드로이드의 전반적인 OS 부팅 구조와 rooting 방법에 대해 공부해볼 수 있었다. 하지만, 이번 과제에서 한계점도 존재했다.

OTA의 정의는 wireless 방식으로 파일을 받아와서, 펌웨어 업데이트를 하는 과정을 노리는 것인데, 리모트에서 받아오는 파일이 아니라는 점이 있다. 또한, secure boot chain이 구현되어 있는 상태에서는 boot/recovery 파티션을 건드리면, 정상 부팅이 불가능하다. 이런 점에 따르면, 이번 과제는 현재 널리 쓰이는 Google의 Pixel 시리즈와 Samsung의 Galaxy 시리즈와 같은 실 기기에는 적용할 수 없다는 점이다.

무엇보다도, recovery OS에서 init.sh에 일종의 명령어들은 모두 root 권한으로 넣게 되는데, 이런 OS가 있더라도, root 권한이 없으면 애초에 불가능한 예제이기 때문에 이번 실습은 매우 특수한 예제임을 알 수 있었다.