**Operating System Security**

Report #3 Android Rooting Attack

|  |  |
| --- | --- |
| **학번 / 이름** | 32131766 황준일  3215 전상민 |
| **담당교수** | 조성제 교수님 |
| **제출일** | 2019. 12. 9 |

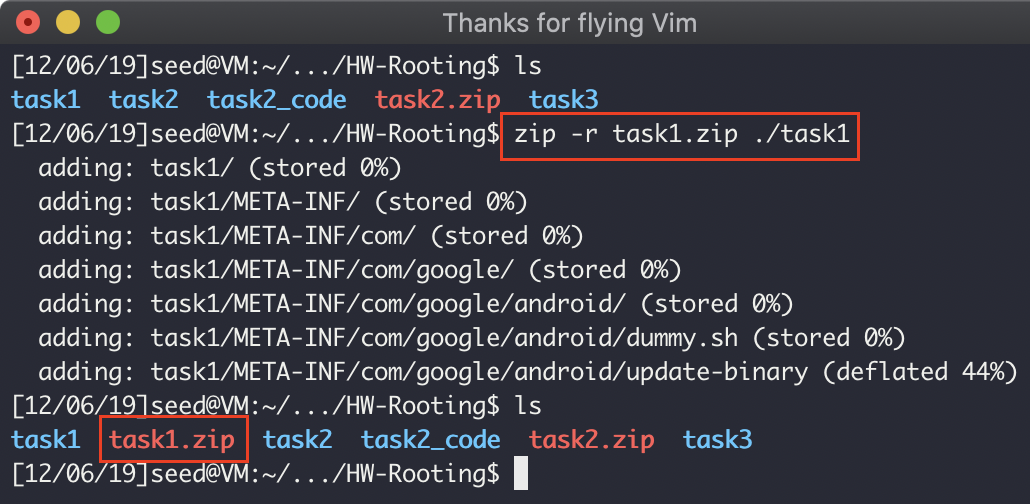
1. **Task 01: Build a simple OTA package**
   1. **진행 과정**  
      dummy.sh, update-binary 작성 → 파일 압축 → Android VM에서 Recovery OS 접근  
      → scp 명령을 통해 Android VM의 /tmp directory로 압축 파일 전송 후 압축 해제 (/tmp directory에는 누구나 파일을 작성할 수 있다)  
      → update-binary 실행 후 rebooting → Reboot 후 /system/dummy 파일 확인
   2. **상세 설명**  
      [seed@SEEDUbuntuVM]  
      ~/task1/META-INF/com/google/android/**dummy.sh**

|  |
| --- |
| echo hello > /system/dummy # 실행시 system 폴더에 dummy 파일 생성 |

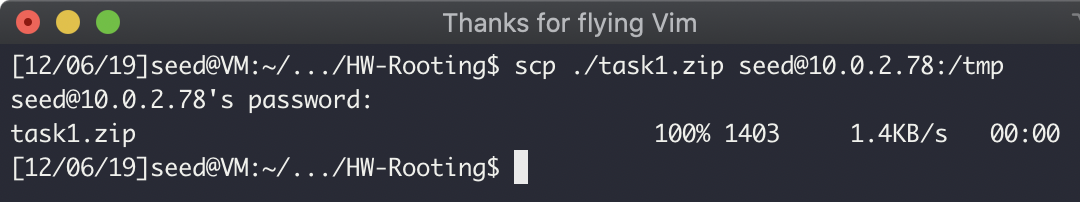
[seed@SEEDUbuntuVM]  
~/task1/META-INF/com/google/android/**update-binary**

|  |
| --- |
| cp dummy.sh /android/system/xbin # 현재 폴더에 있는 dummy.sh를 /android/system/xbin에 복사한다.  chmod a-x /anroid/system/xbin/dummy.sh # dummy.sh에 실행권한을 부여한다.  # init.sh에서 return 0 탐색 후, 해당 코드 뒤에 /system/xbin/dummy.sh를 추가한다 # init.sh: 부팅 과정에서 실행되는 Shell Script  # 즉, 부팅시 init.sh에서 dummy.sh를 실행하도록 함 sed -i "/return 0/i /system/xbin/dummy.sh" /android/system/etc/init.sh |

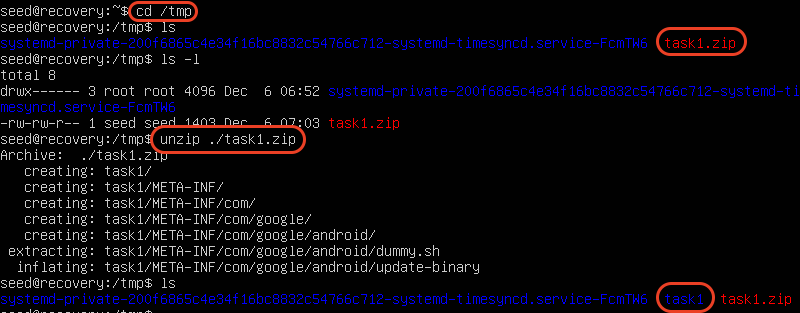
**chmod a+x** ~/task1/META-INF/com/google/android/update-binary # 실행 권한이 필요하다.

[seed@SEEDUbuntuVM]  
zip -r task1.zip ./task1  
# ./task1 폴더를 ./task1.zip 으로 압축. -r은 모든 리소스를 포함한다는 의미  


- Android VM에서 Recovery OS(Ubuntu)로 로그인 후 IP주소 확인  
  
# id: seed pw: dees ip주소: 10.0.2.78

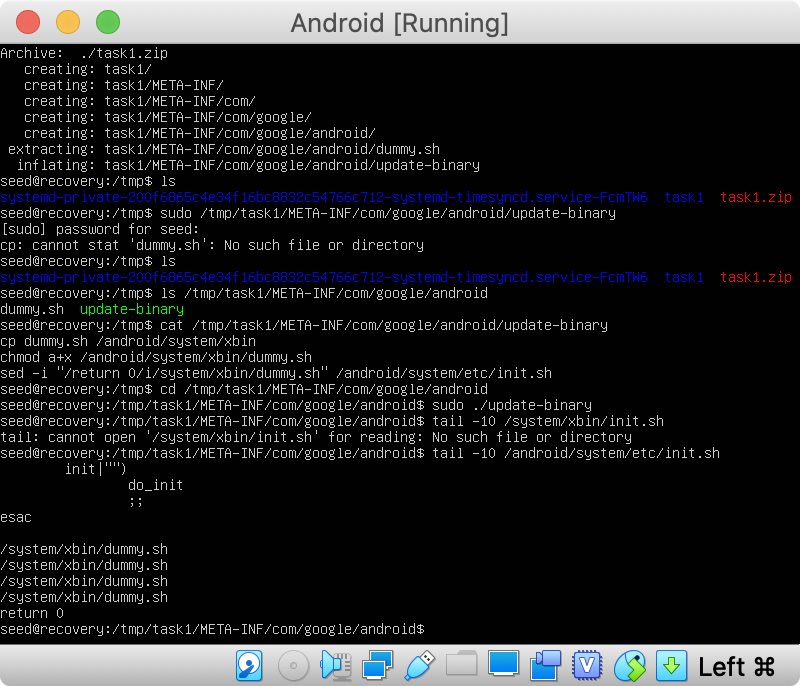
[seed@SEEDUbuntuVM]  
scp ./task1.zip seed@10.0.2.78:/tmp  
# ./task1.zip을 10.0.2.78에 접근 후 seed 계정으로 로그인 후 /tmp 폴더에다 복사한다.  


[seed@RecoveryOS]  
unzip /tmp/task1.zip

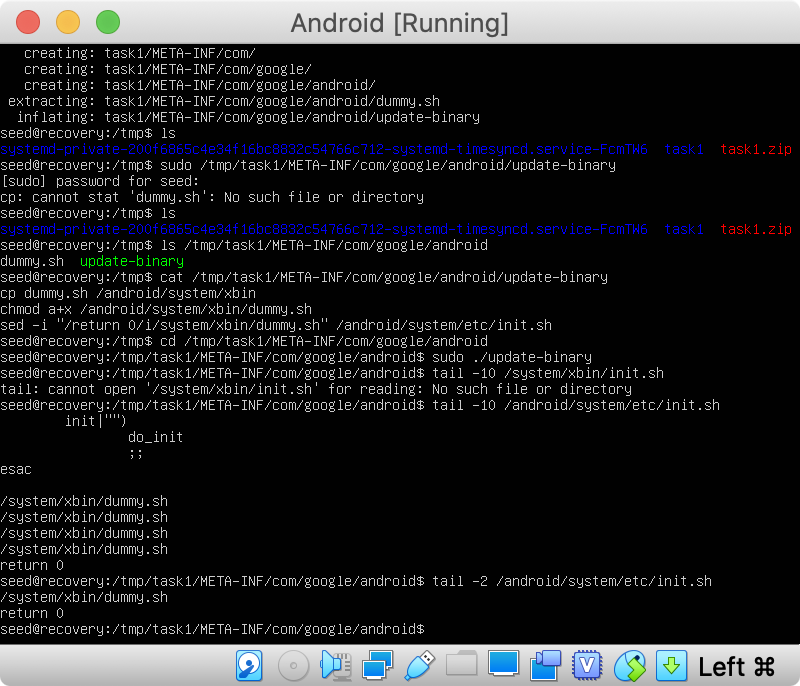


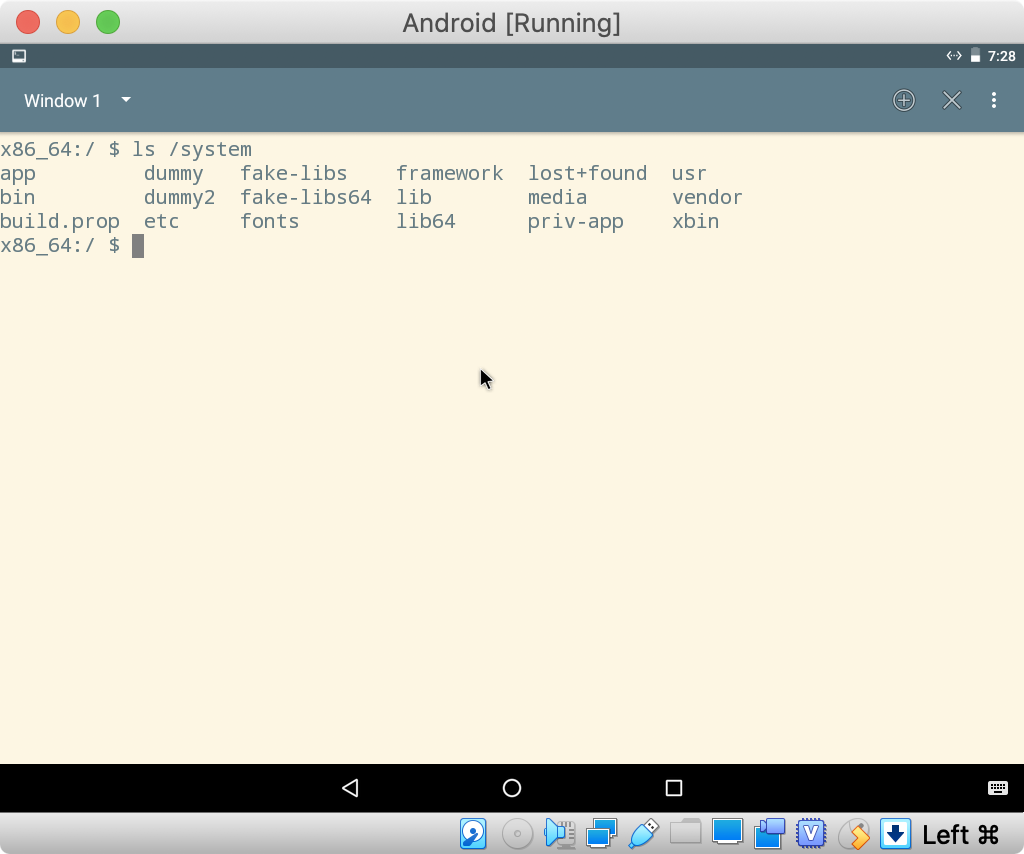
cd /tmp/task1/META-INF/com/google/android

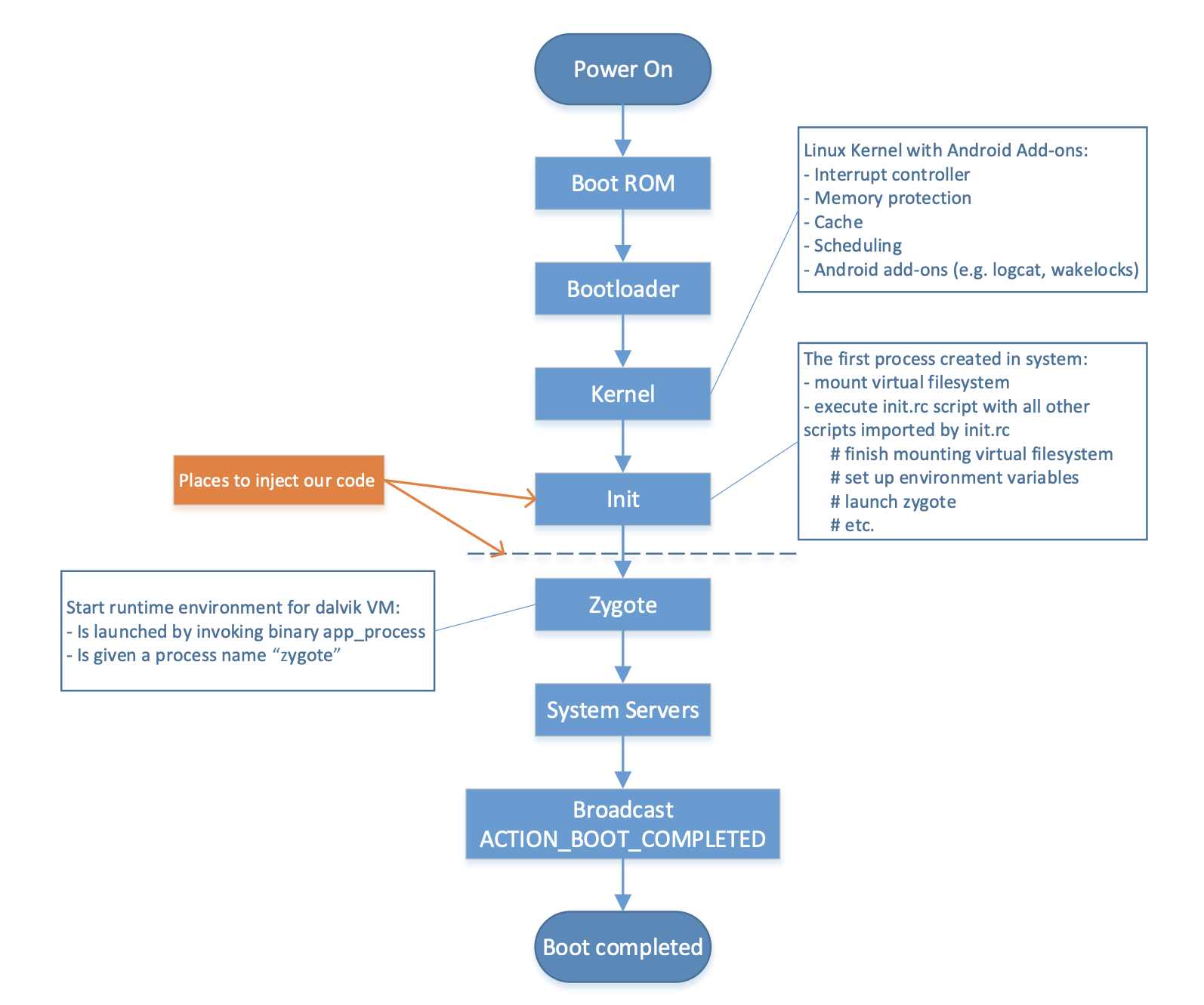
sudo ./update-binary 실행



/anroid/system/etc/init.sh가 수정되었는지 확인  
이 때 tail 명령어를 사용하면 뒤에있는 내용부터 탐색가능



AndroidVM 리부팅 후 /system/dummy의 존재 확인  
  


1. **Task 02: Inject code via app process**
   1. **핵심 개념**  
        
        
      위의 그림은 Android의 Boot 과정이다.  
      여기서 핵심내용은 다음과 같다.

|  |
| --- |
| init.sh → 시스템에서 생성 된 첫 번째 프로세스. 파일 시스템 마운트. init.rc 실행  init.rc → 환경변수 설정. zygote 런칭  Zygote → app\_process 호출(process name "zygote") → Dalvik VM(Android App의 Runtime Machine) 실행 |

Android Runtime Bootstrap이 Root 권한으로 항상 app process라는 프로그램을 실행한다.  
이 때 Zygote 데몬이 실행되고, Zygote를 통해 다른 process가 fork/excute 된다.  
즉, Zygoete는 모든 process의 parent가 되는 것이다.

[우리의 목표]  
app\_process 수정 → 원하는 process 실행 → /system/dummy2 생성 → **“루트 권한으로 프로그램을 실행할 수 있음”**

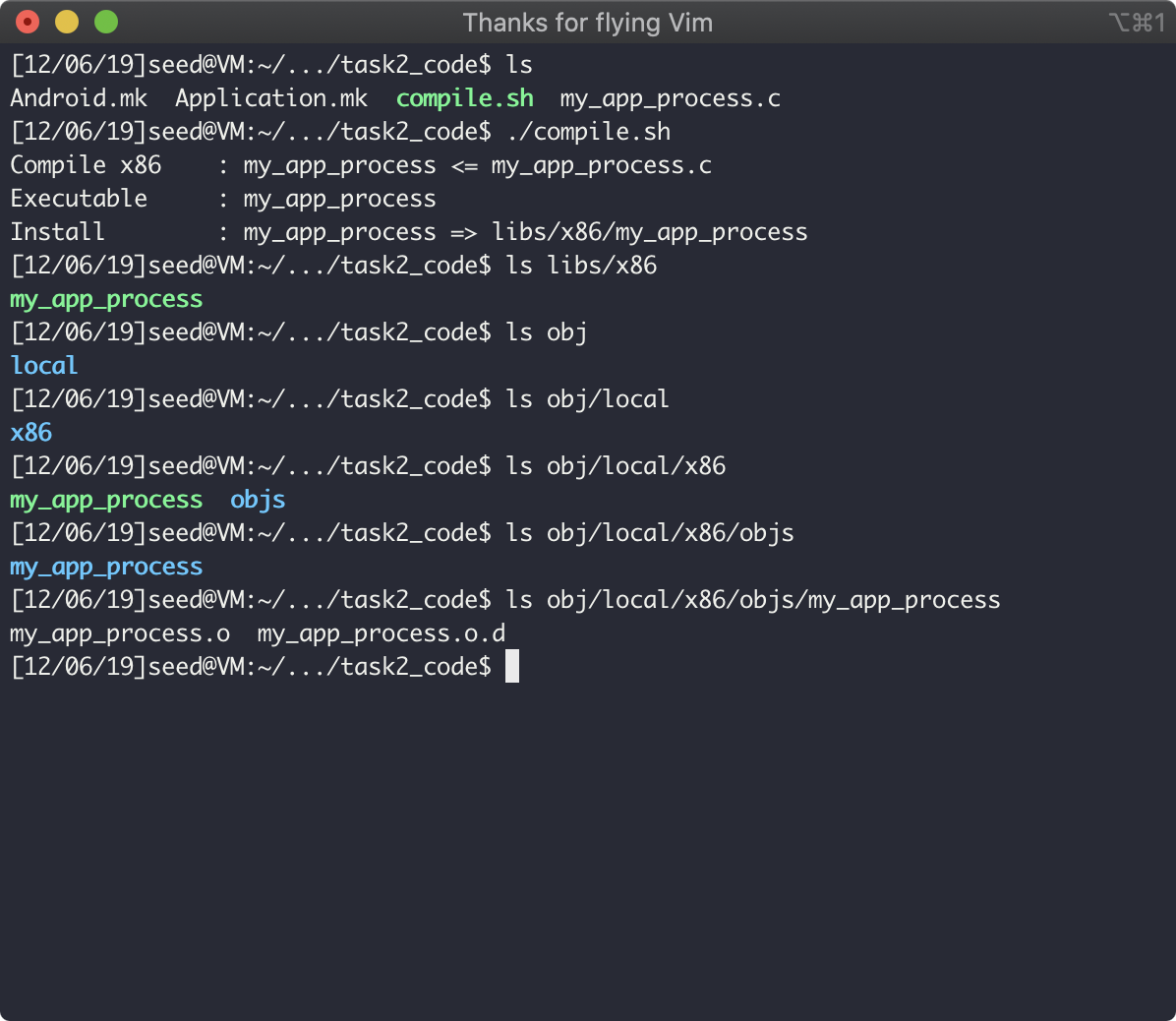
* 1. **진행과정**my\_app\_process.c / Android.mk / Application.mk / compile.sh / update-binary 작성  
     → my\_app\_process 컴파일 → 파일 압축  
     → Android VM에서 Recovery OS 접근  
     → scp 명령을 통해 Android VM의 /tmp directory로 압축 파일 전송 후 압축 해제  
     → update-binary 실행 후 rebooting → Reboot 후 /system/dummy2 파일 확인
  2. **실행코드 상세설명**

|  |
| --- |
| **my\_app\_process.c** |
| #include <stdio.h>  #include <stdlib.h>  #include <unistd.h>  extern char\*\* environ;  int main(int argc, char\*\* argv) {  // dummy2 생성  FILE\* f = fopen("/system/dummy2", "w");  if (f == NULL) {  printf("Permission Denied.\n");  exit(EXIT\_FAILURE);  }  fclose(f);   // 원본 app\_process 실행  char\* cmd = "/system/bin/app\_process\_original";  execve(cmd, argv, environ);  return EXIT\_FAILURE;  } |

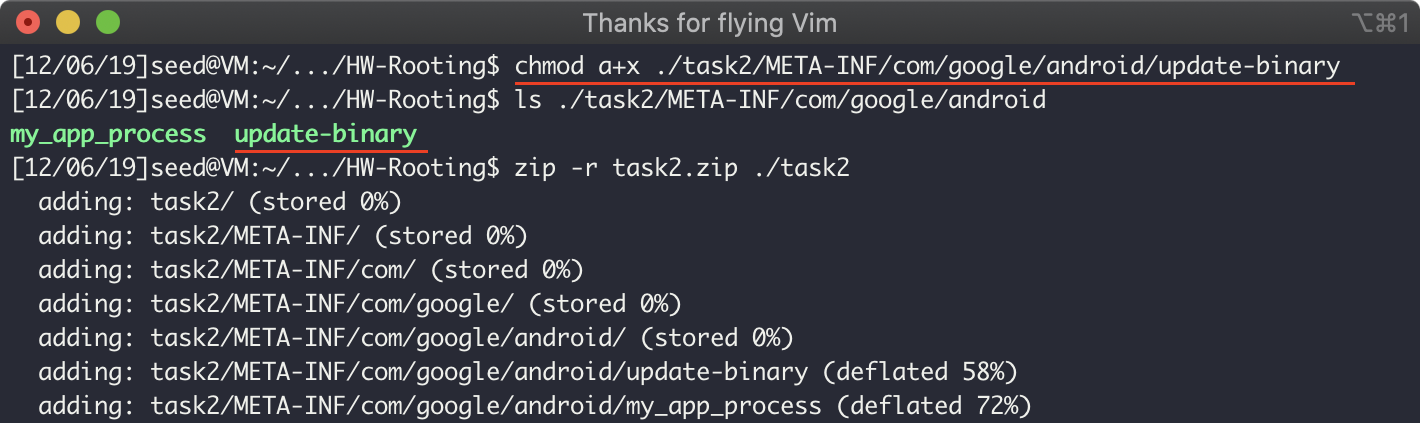
|  |
| --- |
| **update-binary** |
| # app\_process는 64bit 운영체제에서 app\_process64가 되고, 32bit에선 app\_process32가 된다.  # 원본 app\_process32를 app\_process\_original로 이름 변경 mv /android/system/bin/app\_process32 /android/system/bin/app\_process\_original cp my\_app\_process /android/system/bin/app\_process32 # my\_app\_process를 app\_process32로 이름 변경  chmod a+x /android/system/bin/app\_process32 # 실행 권한 부여 |

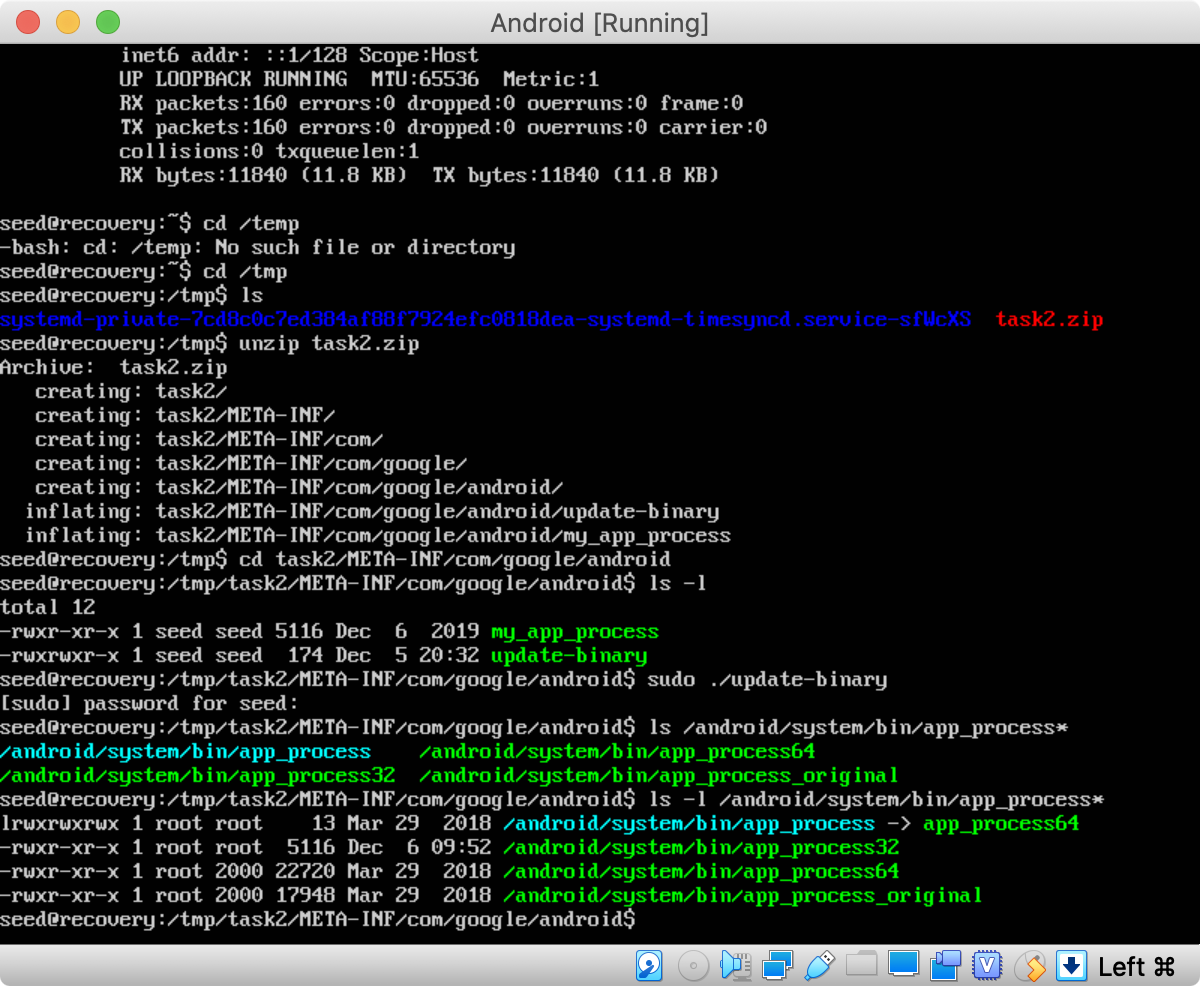
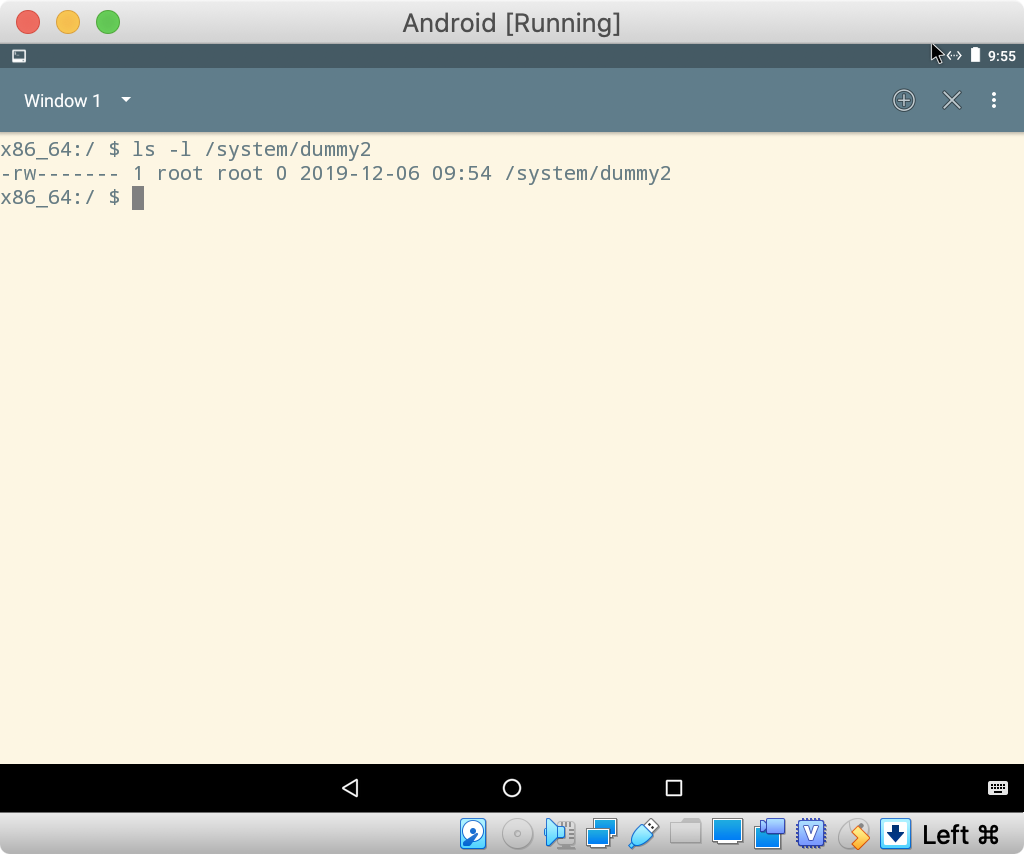
|  |  |
| --- | --- |
| **compile.sh ( my\_app\_process.c → my\_app\_process )** | |
| # Java는 JNI라는 것으로 Native Method를 사용할 수 있다.  # 그리고 Android에서는 Native Method를 App내에서 사용할 수 있도록 developer kit를 제공하는데 이를 JDK라고 한다.  # JNI: Java Native Interface  # Native Method: C or C++로 만들어진 Library Method  # NDK: Native Development Kit. Android OS에서 C 및 C ++ 코드를 컴파일 할 수있는 도구 세트 # NDK를 사용하려면 Application.mk와 Android.mk라는 두 파일을 만들어 소스 코드와 같은 폴더에 배치해야합니다.  # ndk-build: mk파일을 NDK를 이용하여 build # NDK\_APPLICATION\_MK = <file> 은 특정 file을 사용하여 빌드한다는 옵션 export NDK\_PROJECT\_PATH=. ndk-build NDK\_APPLICATION\_MK=./Application.mk | |
| **Application.mk (ndk-build의 전체 설정을 지정)** | |
| APP\_ABI := x86 # App Binary Interface. Binary 호환범위. x86 = 32bit 운영체제 호환  APP\_PLATFORM := android-21 # Android 최소 지원 범위  APP\_STR := stlport\_static # C library linking  APP\_BUILD\_SCRIPT := Android.mk | |
| **Android.mk 작성** | |
| LOCAL\_PATH := $(call my-dir) # Build 할 대상(Source Tree Root)의 위치. my-dir: 현재 폴더  include $(CLEAR\_VARS) # 모든 LOCAL 변수 초기화(LOCAL\_PATH제외)  LOCAL\_MODULE := my\_app\_process # 빌드하려는 모듈 이름  LOCAL\_SRC\_FILES := my\_app\_process.c # 빌드 대상  include $(BUILD\_EXECUTABLE) # 전체 Linking 후 빌드 실행 | |

* 1. **실행결과**



compile.sh 실행 → my\_app\_process 생성 완료  
  


my\_app\_process를 /META-INF/com/google/android 폴더로 옮긴다.  
  
  
update-binary에 실행 권한이 없는 상태이기 때문에 실행권한을 부여해야한다. 그리고 압축파일로 만든다.  
  


scp를 이용하여 RecoveryOS/tmp 폴더에 task2.zip을 전송한다.  
  
  
task2.zip의 압축을 해제한 후, update-binary를 실행한다.  
정상적으로 실행되었다면, /android/system/bin 폴더에 기존 app\_process32가 app\_process\_original으로 변경되었고,  
/META-INF/com/google/android/my\_app\_process가 복사되어 app\_process32로 변경된 것을 확인할 수 있다.  
  
  
Android로 부팅 후 /system/dummy2 파일이 생성된 것을 확인할 수 있다.

