Inject code via app process

In the previous task, we modify the init.sh file to get our injected program to run automatically, and with the root privilege. This initialization script file is used by the underlying Linux operating system. Once the Linux part is initialized, Android OS will bootstrap its runtime that is built on top of Linux. We would like to execute our injected program during this bootstrapping process. The objective of this task is not only to find a different way to do what we have done in the previous task, but also to learn how Android gets bootstrapped.

We can see that when the Android runtime bootstraps, it always run a program called app process, using the root privilege. This starts the Zygote daemon, whose mission is to launch applications. This means that Zygote is the parent of all app processes. Our goal is to modify app process, so in addition to launch the Zygote daemon, it also runs something of our choice. Similar to the previous task, we want to put a dummy file (dummy2) in the/system folder to demonstrate that we can run our program with the root privilege.

The following sample code is a wrapper for the original app process. We will rename the original app process binary to app process original, and call our wrapper program app process. In our wrapper, we first write something to the dummy file, and then invoke the original app process program.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

extern char** environ;

int main(int argc, char** argv) {
    //Write the dummy file
    FILE* f = fopen("/system/dummy2", "w");
    if (f == NULL) {
        printf("Permission Denied.\n");
        exit(EXIT_FAILURE);
    }
    fclose(f);

    //Launch the original binary
    char* cmd = "/system/bin/app_process_original";
    execve(cmd, argv, environ);

    //execve() returns only if it fails
    return EXIT_FAILURE;
}
```

It should be noted that when launching the original app process binary using execve(), we should pass all the original arguments (the argv array) and environment variables (environ) to it.

Step 1. Compile the code.

We need to compile the above code in our Ubuntu VM, not inside the recovery OS or Android OS, as neither of them has the native code development environment installed; we have installed the Native Development Kit (NDK) in our Ubuntu VM. NDK is a set of tools that allow us to compile C and C++ code for Android OS. This type of code, called native code, can either be a stand-alone native program, or invoked by Java code in Android apps via JNI (Java Native Interface). Our wrapper app process program is a standalone native program, which needs to be compiled using NDK. For more detailed instructions about NDK, please refer to the instructional manual linked in the web page.

To use NDK, we need to create two files, Application.mk and Android.mk, and place them in the same folder as your source code. The contents of these two files are described in the following:

```
The Application.mk file

APP_ABI := x86
APP_PLATFORM := android-21
APP_STL := stlport_static
APP_BUILD_SCRIPT := Android.mk
```

```
The Android.mk file

LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE := <compiled binary name>
LOCAL_SRC_FILES := <all source files>
include $(BUILD_EXECUTABLE)
```

We run the following commands inside the source folder to compile our code. If the compilation succeeds, we can find the binary file in the./libs/x86folder.

```
export NDK_PROJECT_PATH=.
ndk-build NDK_APPLICATION_MK=./Application.mk
```

Step 2. Write the update script and build OTA package.

Just like the previous task, we need to write update-binary to tell the recovery OS what to do. Students need to write the shell script code in this task. Here are some guidelines:

- We need to copy our compiled binary code to the corresponding location inside Android.
- We need to rename the original app process binary to something else, and then use our code as app process. The actual name of app process can be either appprocess32 or appprocess64, depending on the architecture of the device. Our Android VM is a 64-bit device, so the name should be app process64.