Lesson 8 – Reverse Engineering Dangerous Environment Detection

Introduction

This lesson focuses on protections against running the application in a dangerous environment – on rooted device or an emulator. Secure application should detect running an application on a rooted device, because it may be a target to malware which could use the root access steal the data from the application. Also running on the rooted device or an emulator could be used to Reverse Engineer the application.
Users may not know that their devices are rooted, so some application producers decide just to put a warning and send the information to the server when a user runs an application on rooted device, while others make the application stop working. Root detection is crucial for high security applications – like MDM (Mobile Device Managment) which often make encrypted containers for confidential data and wipe them in the event of root detection.

Getting started

You will need:

- a rooted device and emulator.

Finding vulnerabilities

To verify if the application can be run on a rooted phone and an emulator start it, try to use its functionalities and check for any messages about root access to be displayed. Sometimes applications are allowed to run but they offer limited functionality on rooted devices.
Also look for information about root detection in the system log.

If nothing was found the code of the application can be inspected in order to search for root or emulator detection code.

Fixing the vulnerability

There are many ways to detect rooted device or emulator, simple check if the device is rooted can be performed using only one of them. But high security applications combine many of these checks together with obfuscating the code heavily to make reverse engineering more difficult. The best solutions also use root and emulator detection in native code to make the decompilation process more costly in time and resources. Some of the root detection also detect emulators and vice versa since emulators usually provide root access.
Detection can be performed by

- Running su command and checking uid with id command (root has uid 0). Most root access managing applications make user explicitly allow or deny root access to an application, so the denial may result in false negative for this check, but this should work for emulators.
- Checking for the existence of su binary in binary directories like /system/, /system/bin/, /system/xbin/, /sbin/.
- Checking for the existence of packages of rooting applications and root managing applications like eu.chainfire.supersu, com.kingroot.kinguser.
- Checking build.prop file with build properties for “test-keys” string that indicates unofficial ROM or “sdk”, “Genymotion”, “generic”, “Emulator” which indicate emulator.

- Existence of some binaries that are not a standard part of Android release build, but are often installed on rooted devices. This includes busybox and binaries it provides – telnet, ftp, wget, vi, etc.

Example environment check function:

```
public boolean checkEnvironment() {
//returns true if device is rooted or is an emulator

//root and emulator check
String[] files = {"/system/su", "/system/bin/su", "/system/xbin/su","/sbin/su"};
for (String file : files) {
      if (new File(file).exists()) return true;
}

//unofficial ROM check
if (Build.TAGS.contains("test-keys")
      return true;

//emulator check
if (Build.FINGERPRINT.contains("generic")
|| Build.PRODUCT.contains("sdk")
|| Build.MODEL.contains("Emulator")
|| Build.MANUFACTURER.contains("Genymotion"))
      return true;
}
```