Assignment 2

Name: Elad Sezanayev
ID: 211909940

Weak Spots and how i found it:

As I explained in the previous assignment, the classifier I choose extracts its classification data from the /proc file system of an android emulator at runtime of the application (the apk file).
In detail it takes the data for the features from the following files:

1) **/proc/stat -** holds various pieces of information about the kernel activity.
2) **/proc/net/dev -** contains statistics about network interfaces.
3) **/proc/[pid]/statm -** Provides information about memory usage of the process with this pid.
4) **/proc/[pid]/stat -** Status information about the process with pid.

(Of curse that pid indicate the process id of the running apk)

Now we can see that the last two concrete files (3, 4) are related to the application process and only to him.
Therefore, there is such a weak spot: if we manage to run malicious code from another context (another process) that is not at all related to the process that runs the application - (not related as a child process), and also does not use too many general things that are checked in the first 2 files
(1 - kernel/system, 2 - network), so the classifier won't even see this suspicious activity.

How can we do it? For example if we manage to enable a malicious service to always run during the emulator's boot process. This malicious service will run only after the classifier is done with our app and rebooted. so of course it won't get the same pid as our app process.
Furthermore, if we succeed to enable such a malicious service to always run during the emulator's boot process, then the classifier's emulator will be infected by this service and any app it classifies from now on will be analyzed while the malicious service is running in the background, which could break the classifier's accuracy (at least until the classifier change his emulator or reset it).
So the main goal right now is to find an app that the classifier classifies as non malicious, add to it our malicious logic and keep that the classifier classifies as non malicious.

How am i intend to exploit it:

We have reached to the interesting part,
How am I going to exploit the weakness?
To answer this question we will have to explain about services managers like init.rc, systemd or sysVinit.

From systemd website:

“Linux services managers provides aggressive parallelization capabilities, uses socket and D-Bus activation for starting services, offers on-demand starting of daemons, keeps track of processes using Linux control groups, maintains mount and automount points, and implements an elaborate transactional dependency-based service control logic.”

And in simplicity:

Linux services managers provide us an easy to use interface to starting and stopping daemon processes at any time, at boot process, depending on other process status, depending on runlevels/groups etc.

So all we need to do is understand how to write a service which will run a Malicious code for us.
Most android distro uses init.rc services manager so we will cover it up.

init.rc uses runlevels. A runlevel defines the state of the machine after boot. Different runlevels are typically assigned (not necessarily in any particular order) to the single user mode, multi user mode without network services started, multi-user mode with network services started, system shutdown, and system reboot system states.

Every runlevel has a directory or file under **/etc/rc.d** directory and every runlevel directory (or file) contains a set of scripts that the system runs whenever it wants to enter this run level.
Those scripts usually start a bigger program.
At **/etc/rc.d** directory there will be either a set of files named **rc.0, rc.1, rc.2, rc.3, rc.4, rc.5** and **rc.6,** or a set of directories named **rc0.d, rc1.d, rc2.d, rc3.d, rc4.d, rc5.d** and **rc6.d**.
For example, run level 1 will have its start script either in file /etc/rc.d/rc.1 or any files in the directory /etc/rc.d/rc1.d.

Now all we need to do is to write the malicious program (can be in any binary format that can run on android) then write the script that will run the program compress it inside a non malicious apk and add logic to the app that when the app start running extract the script and move it to one of the runlevels directories (to be run automatically when entered to this runlevel).

And there is a problem! in most cases we can’t just copy/move scripts to those directories without permissions and it might be detected by **/proc/stat**.

So how can we do it?

Almost all of the scripts that are already in the runlevel directories are not real files but symbolic links.

So we can find the real location of those scripts and there we will have permissions

So we just gonna overwrite one of them with our script and it will be called instead.