DETERMINATION OF THE PERFORMANCE OF ANDROID ANTI-MALWARE SCANNERS

AV-TEST GmbH
Klewitzstr. 7
39112 Magdeburg
Germany
www.av-test.org

CONTENT

ABSTRACT

Today's mobile devices are small computers whose functionality is similar to desktop computers. With the high number of different sensors they can even be used for tasks, that a desktop computer is not capable to achieve. But due to their mobility without a permanent power line their resources are rather limited to save battery power and to gain longer battery life. With the rise of malware for Android based mobile devices there is also the need for an Anti-Malware scanner on a device to protect the device itself and the personal data on it. There are many products on the market, but do they all have proper malware detection and do they use the limited resources economically? This paper shows a way, how the malware detection rate and resource usage of Anti-Malware products can be measured. With the data achieved by our method we are able to determine which Anti-Malware product provides good protection and whether it may lead to a shorter battery life.

Determination of the Malware Detection Rate

Prevalent Malware & Real-Time Testing

For a malware detection test, it's very important, how the sample set is composed. The malware detection rate is determined in two categories. In the prevalent malware category we use only recent malware, which is not older than 4 weeks. The samples cover several relevant malware families. Families with less than 10 samples are classified as 'Other'. The total sample set consists of several thousand samples.

In the real-time test category each sample is tested on the day of its discovery.

On-Demand Test

The first approach to determine the malware detection rate is an on-demand scan, which is provided by most Anti-Malware products. To test a high number of malicious samples, we copy them to the SD card. If the tested Anti-Malware product provides several scan options, we choose the most comprehensive one (e.g. Full System Scan), which usually includes a scan of the SD card. All detected samples will be removed afterwards. All undetected samples will be tested on-access.

On-Access Test

An on-access test uses the real-time protection feature of the Anti-Malware product to scan the malware samples. This is used to determine the complete malware detection rate in case a product doesn't handle all samples with the on-demand scan or doesn't even offer this feature. In the end only the combined score of on-demand and on-access is provided by AV-TEST. The samples are installed over the Android Debug Bridge. The Anti-Malware product has to provide a visible feedback to the user, when a sample was detected.

Test Execution

The prevalent malware samples are tested in two parts. The first part is an on-demand test and the second part is an on-access test. In the second part we only test the missed samples from the first part.

The real-time samples are tested on-access only.

False Positive Test

A malware detection rate is not meaningful without an associated false positive rate. For false positive tests we use the on-access test methodology described above. The samples are known clean apps obtained from Google Play and 3rd party app stores.

MEASURING THE PERFORMANCE IMPACT

CONCEPT

Usually an Anti-Malware testing lab is an ordinary office with desks, PCs, network hardware and mobile devices. The environment doesn't meet laboratory conditions required to reliably test the influence on the battery life for different products in a comparable manner. Such laboratory conditions are e.g. a constant room temperature, an equal number of load cycles of the battery of the test device for each product and an equal age of the battery in case of several test devices. Today's batteries are very optimized and may survive thousands of load cycles before an impact on the battery life can be measured. Nevertheless we can't eliminate such impact by constant laboratory conditions and therefore we use another approach.

To exclude external influences on the test results, we measure the CPU utilization and the network traffic during simulated user activity. See Appendix A for details.

IMPLEMENTATION OF TEST CASES

Basics

To test the CPU utilization of the scanner, we install several known clean apps on the test device using ADB. We measure the used CPU time for each installed app.

In addition to the app installation we've implemented several real world use cases, to measure the performance impact during more common activities.

The basic concept of the implementation of performance test cases is always the same. The controller program[1] starts an activity using the `am` command (Activity Manager), then it waits for a suitable time for the started activity to finish loading of the given data. The activity will be stopped and a custom broadcast will be sent to the performance monitoring app on the device. The monitoring app saves CPU and memory usage as well as the amount of network traffic for each process.

The `am` command for starting an activity usually has the following scheme:

```
$: am start -a android.intent.action.VIEW -d <data> --activity-clear-task <package>
```

<data> contains an URL, e.g. http://www.google.com/ or file:///mnt/sdcard/Download/document.pdf.

<package> is optional, if omitted, the system will use the default app which is associated with the data type, e.g. the browser will display http://-data. Sometimes multiple apps can handle the data, which leads to a popup where the user has to select his preferred app. Therefore for automatic tests it's better to declare a specific package, which is used to display the data.

The `am` command for closing an activity looks like this:

```
$: am force-stop <package>

$: am kill <package>
```

The second line is optional as force-stop should "stop everything associated with <package>".

[1] The test device is controlled over the Android Debug Bridge.

Browsing Websites

To browse websites, we use the Android default browser, which is in the package `com.android.browser` (for Android before 4.1.X) or com.android.chrome
The wait time is set to 10 seconds.

Watching Video

For video playback we use videos hosted on YouTube and the preinstalled YouTube app with the package `com.google.android.youtube`. The wait time is set to the playback time of the video rounded up to the next minute plus one extra minute. E.g. when the video playback time is 2:48, the wait time is 4 minutes.

Reading PDF Documents

To open PDF documents, we use the Adobe Reader app. This app is part of our installation test cases, which usually run before the real world cases. The package is `com.adobe.reader`. The wait time is set to 10 seconds.

Idling

We've also included three test cases for idling. No specific commands are required here. The test cases cover idle times from 10 over 20 to 30 minutes.

CERTIFICATION

To receive an AV-TEST certification, a mobile security app has to achieve more than 8.0 points in our scoring scheme. The scoring is composed out of the 'Protection' category, which includes the malware detection results (combined on-demand and on-access), the 'Usability' category, which includes performance and false positives, and an extra point for additional features.

PROTECTION SCORE

The protection score is calculated from the average detection rate for the prevalent malware and the real-time test. Points achieved are awarded by an allocation key, which is predefined by AV-Test every year. The allocation key describes which threshold in percent is needed for the corresponding points. Range for detection score is from 0 points to 6, by an interval of 0.5 steps.

USABILITY SCORE

The usability score is composed of the performance score and the false positive score. The performance score is 6.0 points minus 2.0 points for each performance impact. Performance impacts may be 'impact on the battery life', 'high traffic usage' and 'slow down'. The false positive score is 6.0 points minus 1.0 point for each false positive detection. The usability score is the sum of the performance und false positive score divided by two.

ADDITIONAL FEATURE POINT

If the mobile security app provides at least two additional security features beside the malware protection, it receives an extra point. Additional security features may include Anti-Theft, Safe Browsing, Privacy Advisor, Call Blocker, Message Filtering, Parental Control, Backup and Data Encryption.

FREQUENTLY ASKED QUESTIONS (FAQ)

Q: There are two products with 100% detection rate, but one got a protection score of 6.0 points and the other one got just 5.5 points.

A: The reason is the rounding of the detection rates. The product with 6.0 points has detected 100.00% and the product with 5.5 points only detected 99.95%, which is rounded to 100%.

APPENDIX A

PERFORMANCE TEST PROOF OF CONCEPT

Our approach is to measure the CPU time used by the Anti-Malware while known clean apps from Google Play are installed on an Android device. The Anti-Malwares real-time protection feature will check the newly installed apps for malware and it will therefore consume CPU time. To compare different Anti-Malware products we calculate the CPU usage in %. We also monitor memory usage and the amount of network traffic for each running process.

In parallel to the process monitoring the battery load is monitored and the battery state is logged every time the battery load changes. With this data we can verify our results from the process monitoring because higher CPU usage should lead to faster battery discharge. This may of course vary due to the external influence factors described above. Therefore battery discharge is not given as the measured value, but merely serves as verification that our approach of measuring the consumed CPU time is actually consistent and useful.

To check whether our concept is feasible, we've implemented an Android app, which does the process monitoring and a Java program, which automatically installs applications from the Google Play website on a specific device using the Google account from the device. Read about details in the implementation part. As test device we used a Samsung Galaxy Nexus with Android 4.0.4. One test cycle includes the installation of 35 apps from Google Play. We ran five test cycles per product with the same 35 apps per each cycle. After one test cycle the apps were uninstalled. The products under test include several free and evaluation versions of popular Anti-Malware products.

The following figures show the results from our test. Figure 1 compares the average CPU usage of the Anti-Malware products while 35 apps were installed from Google Play.

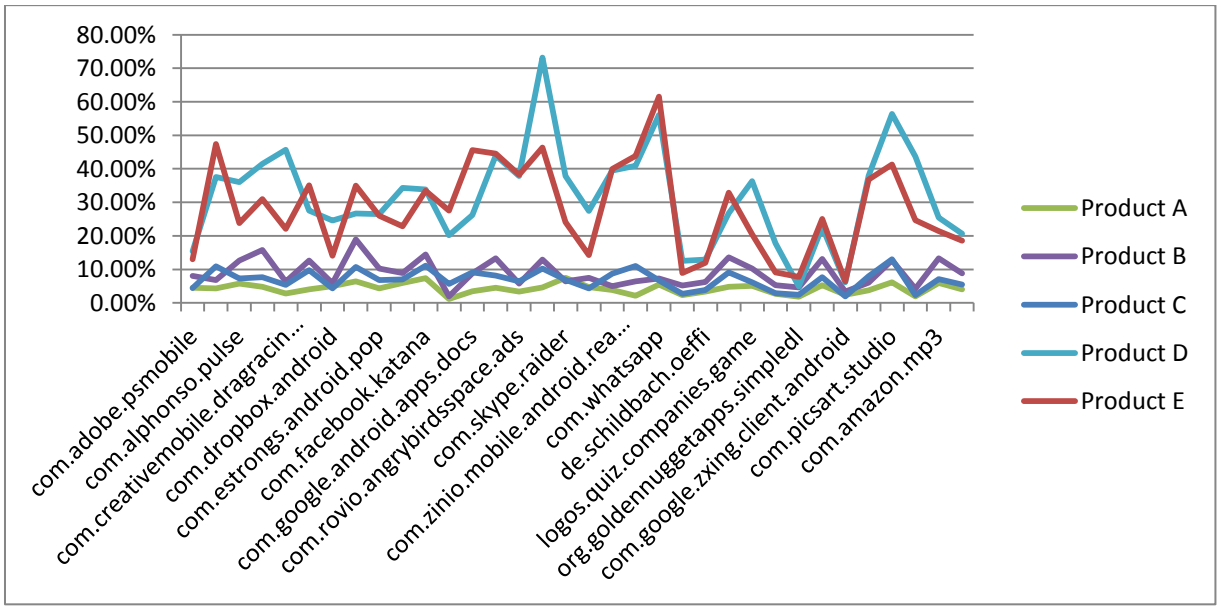

Figure 1: Average CPU usage chart during the installation of 35 apps from Google Play

Figure 2 shows the discharge rate of the battery in percent per minute.

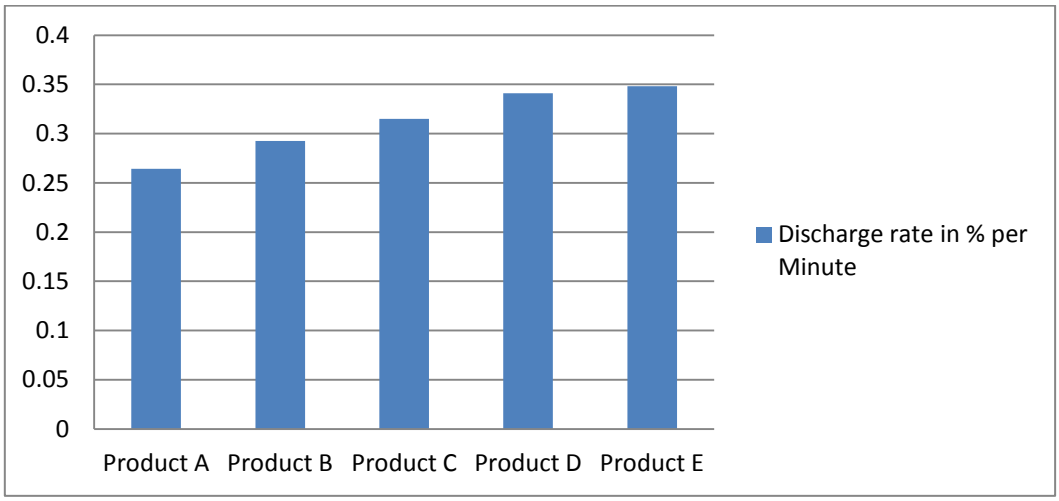

Figure 2: How fast was the battery discharged in average?

Figure 3 projects the values of figure 2 to an estimated battery life when the test would run indefinitely and the battery would discharge from 100% to 0%.

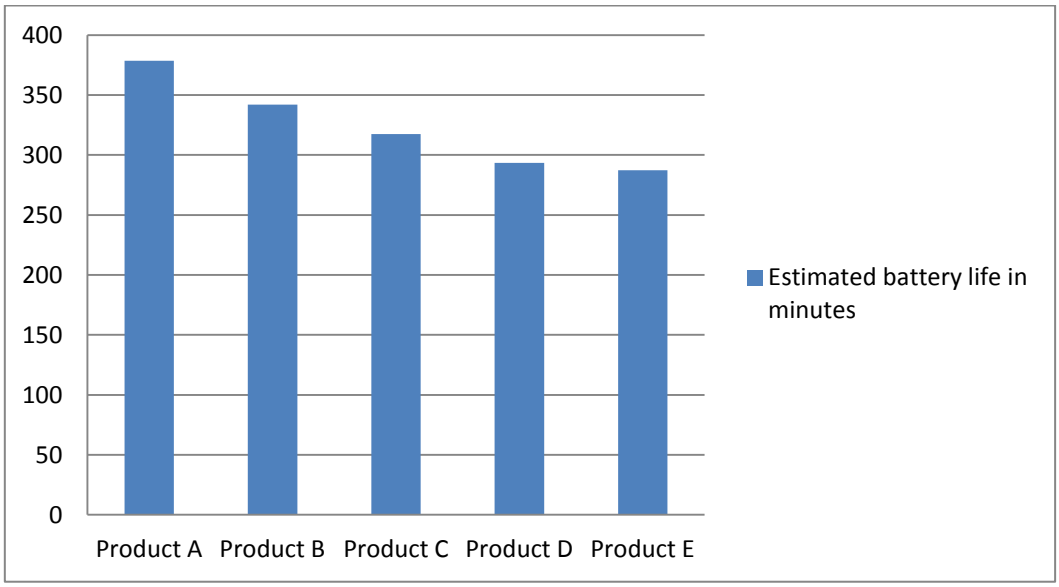

Figure 3: How long would the battery live when the test runs indefinitely?

We can see a clear difference in the figures 2 and 3 between product A with the lowest CPU usage and the products D and E with the highest CPU usage (figure 1). There may be other influences on the battery life like I/O operations or network access, which were not part of this analysis, but it became clear that an CPU efficient programming can increase the battery life.

Implementation

The test environment is set up as client-server-infrastructure where the mobile device is the client with a monitoring Android app and a Java program is the server. Communication is performed via HTTP over Wi-Fi.

The client has a list of Android apps to be installed from Google Play. It sends a request to the server including the next app identified by its Android package name and the Google account the device is registered with.

Using the Selenium browser automation framework the server will log into the Google Play website and start the installation of the requested app.

FIGURE 4: TEST ENVIRONMENT SCHEME

The device installs the app automatically via Wi-Fi. A broadcast receiver listens for the PACKAGE_ADDED action and saves the current state of all running processes to a report file. Then the main activity is notified to send the next request to the server. The broadcast receiver also listens for the BATTERY_CHANGED action and saves the battery state to another report file every time the battery discharges by one percent.