Android Applications Security

Paul POCATILU
Bucharest University of Economic Studies
ppaul@ase.ro

The use of smartphones worldwide is growing very fast and also the malicious attacks have increased. The mobile security applications development keeps the pace with this trend. The paper presents the vulnerabilities of mobile applications. The Android applications and devices are analyzed through the security perspective. The usage of restricted API is also presented. The paper also focuses on how users can prevent these malicious attacks and propose some prevention measures, including the architecture of a mobile security system for Android devices.
Keywords: *Mobile Application, Security, Malware, Android, Reverse Engineering*

1 Introduction

Today, mobile applications security is very important taking in account the high number of mobile devices and users. Worldwide, about one million people were infected by mobile malware in the first semester of 2011 [1]. According to [2] only 4% of smartphones and tablets use antimalware and antivirus software and in the same period the number of mobile applications containing malware on marketplaces grew from 80 to 400 [3]. In [4] is presented a history of the most important mobile malware and several tools that can be used for malware analysis.
The market share of Android based devices is increasing very fast and due to its open source system many attackers are focusing on it.
The objective of this paper is to present the main aspects related to Android applications security. Also, it is proposed a mobile system architecture used to prevent mobile malware.
The paper is structured as follows.
The section *Mobile applications security issues* presents the main concerns related to mobile applications security and possible negative effects.
Android Applications Security Model section focuses on the main characteristics of Android security framework, focusing on the permission system.
Accessing restricted API in Android presents code examples used to access the SMS, location, telephony and file system services in Android applications.
The section *Prevention Measures* deals with best practices to avoid the use of malicious software on a mobile device. Also, there are presented Android cryptography classes and a practical example is given.
The paper ends with conclusions and future work.

2 Mobile Applications Security Issues

Mobile devices are vulnerable to attacks as any other computers even the number of attacks is still reduced compared with personal computers.
The following technologies that exist on many mobile devices can be used by attackers to send and/or receive confidential data, malicious applications and to make unauthorized actions:

- Bluetooth – applications use it to connect to other devices;
- Telephony – unauthorized phone calls are made, resulting in high costs or unauthorized recordings;
- Messaging (Short or Multimedia Messaging Services) – used to send confidential content to attacker or to access paid numbers;
- Wireless networks – used to connect to Internet to transfer data;
- NFC (Near Field Communication) – used for unauthorized payments.

Data services are used in order to transfer from and to malicious applications. The Internet access allows sending/receiving e-mail messages and unauthorized access to

Internet resources (Web sites, Web services, Internet servers etc.)
The behavior of malicious applications could vary from annoying messages to very unrecoverable damages. The possible actions are:

- annoying messages;
- unwanted web pages opened;
- advertising popup;
- higher costs resulting from messages sent (SMS, MMS), phone calls and payments;
- unauthorized use of personal information data;
- confidential data transferred on a remote location;
- altering data stored in file system, contacts, messages etc.

Depending on the platform and/or operating system, applications can:

- read and write contacts;
- read and write calendar entries;
- read short/multimedia messages;
- send e-mail messages, SMS and MMS;
- make phone calls;
- get location;
- access the internet;
- read and write on file system.

Not all operating systems and platform allow all of these actions that can be dangerous if they are used by a malicious application. Also the user is not always informed about these actions when install the application.

3 Android Applications Security Model

Android operating system is based on *Linux kernel* so that applications isolation, the file system and security rules are Linux specific.
As can be seen from Figure 1, users can interact with Android device using Linux shell provided through *adb* tool. Usually development devices have root access granted with full rights, but this access is not available on most of the Android devices.

```
c:\android-sdk-windows\platform-tools>adb -d shell
$ ls -l
ls -l
drwxr-xr-x root     system            2011-10-03 15:44 app-cache
dr-x------ root     root              2011-10-03 15:44 config
lrwxrwxrwx root     root              2011-10-03 15:44 sdcard -> /mnt/sdcard
drwxr-xr-x root     root              2011-10-03 15:44 acct
drwxrwxr-x root     system            2011-10-03 15:44 mnt
lrwxrwxrwx root     root              2011-10-03 15:44 etc -> /system/etc
drwxrwx--x system   system            2011-10-03 15:44 vendor
drwx------ root     root              2011-10-04 13:26 devlog
drwxrwx--- system   cache             2011-10-04 13:29 cache
-rw-r--r-- root     root          4429 1970-01-01 02:00 ueventd.rc
-rw-r--r-- root     root           758 1970-01-01 02:00 ueventd.pyramid.rc
-rw-r--r-- root     root             0 1970-01-01 02:00 ueventd.goldfish.rc
drwxr-xr-x root     root              2011-08-13 12:28 system
drwxr-xr-x root     root              2011-10-03 15:44 sys
drwxr-x--- root     root              1970-01-01 02:00 sbin
dr-xr-xr-x root     root              1970-01-01 02:00 proc
-rwxr-x--- root     root         20380 1970-01-01 02:00 init.rc
-rwxr-x--- root     root          9328 1970-01-01 02:00 init.pyramid.rc
-rwxr-x--- root     root          1677 1970-01-01 02:00 init.goldfish.rc
-rwxr-x--- root     root        107208 1970-01-01 02:00 init
-rw-r--r-- root     root           118 1970-01-01 02:00 default.prop
```

Fig. 1. Android command line shell

Android applications files are packed in files with *apk* extension. These files contain all the classes and resources required by the applications [5]. The binary classes are converted into a proprietary format for Dalvik Virtual Machine (*dex* files). Usually, when applications are launched for the first time, the *dex* files are optimized and the resulting files are stored in */data/dalvik-cache* directory. When the same application is launched again, the optimized code is loaded from that special directory.
Each application runs in a separate virtual machine, having its own unique identifier (UID) Figure 2. Due to this, application resources are protected from other applications and the communication and data transfer between applications has a high degree of confidence. Even so, applications could communicate each other using messages, this being another source of threat.

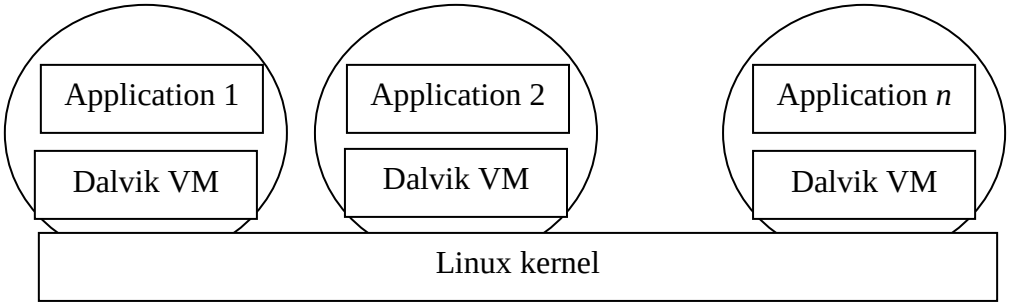

Fig. 2. Android applications

Android applications can be installed from various sources, including:

- Android Market
- Alternative online shops
- Own developed applications
- Other sources: third-party developers, unauthorized sites etc.

The applications published on Android Market by the developers do not need approval from Google like the application published on App Store (Apple) or Marketplace (Microsoft).

Android applications need to be signed before they will publish on the Android Market. The certificate can be generated using *keytool* and *jarsigner* Java tools manually or automatically. *zipalign* tool is used to optimize the *apk* file after it has been signed [6].

Applications need user permissions in order to access restricted API. According to [7], application permissions fall in one of these categories:

- *normal* - the features used by the applications don't presents any risk for applications or system and the user is not informed when the applications is installed;
- *dangerous* – the application has capabilities that, if used by malicious code, could produce negative effects and the user must be aware of these;
- *signature* – the applications need to be signed and the signature has to be the same as that used to define the permissions;
- *signature or system* – this is required for system applications (installed in *system* folder) and it is not available for normal applications.

All Android applications need users' permission in order to access potentially dangerous features. All permissions required need to be declared in application's *AndroidManifest.xml* file like this:

```
<uses-permission android:name=
      "android.permission.INTERNET">
</uses-permission>
```

Table 1 presents several Android permissions required to use restricted API.

Table 1 Android application permissions

Action	**Required Permission**
Read and write contacts	READ_CONTACTS WRITE_CONTACTS
Read and write Calendar items	READ_CALENDAR, WRITE_CALENDAR
Send SMS, read and write SMS	SEND_SMS, READ_SMS, WRITE_SMS
Access the Internet	INTERNET
Use the telephony	CALL_PHONE
Access the external storage	READ_EXTERNAL_STORAGE WRITE_EXTERNAL_STORAGE
Get location	ACCESS_FINE_LOCATION, ACCESS_COARSE_LOCATION

If the permissions are not included in the application's manifest file, that will lead to a run-time exception.

Android security model is based on these permissions that the users have to be aware in order to prevent malicious applications installation.

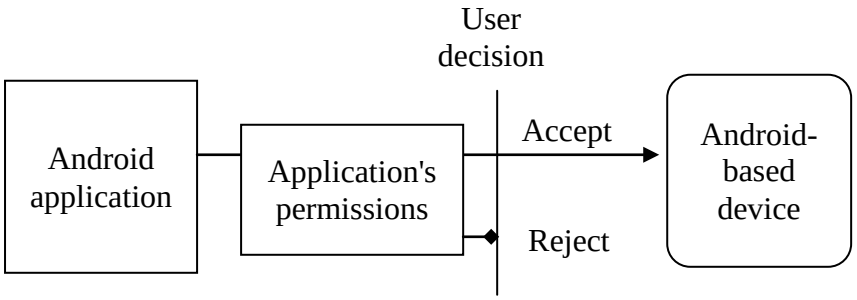

Fig. 3. Android applications installation process

Figure 3 depicts the process of application installation on an Android device. If the user doesn't accept the permissions required by the applications the installation process will end, otherwise the application will be installed on the device.

Also, applications can define custom permissions that can be use to access their resources. The permission access is similar and these have to be declared in application's manifest file.

Attackers write their own applications or modify existing ones by reverse engineering. For example, an existing application published on Android Market could be modified and malicious code inserted and published on the Market with a different name or on a public Web site. After an Android package is modified (*apk* file) it has to be signed again.

There are specialized tools for decompiling, modification and compiling Android binary files such as:

- *APKTool* [8] – used for compile and decompile *apk* files; resources and xml files can be modified easily;
- *smali* [9] – assembler and dissasembler for Dalvik binary files (*dex* files);
- *dex2jar* [10] – converts Dalvik binary files (*dex* files) into Java archives (*jar* files) that contain standard Java *class* files;
- *JD-GUI* [11] – decompile a *class* file into a Java source file.

These tools and similar ones are also used for malware analysis.

4 Accessing restricted API in Android

All restricted API used in Android applications requires permissions due to potentially dangerous consequences when used by malicious code. Attackers could use very easily in their malicious code the services provided by the:

- SMS and MMS;
- e-mail system;
- telephony;
- personal information system (calendar, contacts);
- file system;
- location providers.

In this respect, several examples will be provided.

In order to ***send SMS***, after the *SmsManager* is initialized, the message is immediately sent as follows:

```
SmsManager mesaj =
  SmsManager.getDefault();
mesaj.sendTextMessage("1234", null, "SMS
de test", null, null);
```

If the default settings are used, the not null parameters are the phone number and the message text. This code doesn't tell anything about the operation success. It has to be implemented with specific intents. The user is not aware about the success or failure of this action (sent or delivered).

Malicious application can install a *BroadcastReceiver* for incoming messages in order to get access to private information. As the messages arrive, they are received by the malicious application and processed. An example of this type of receiver is presented in the next section.

In order to ***make a phone call***, the specialized activity needs to be called: dialer or call. The code is similar to this:

```
Uri nrTel = Uri.parse("tel:1234");
Intent apelTel =
 new Intent(Intent.ACTION_DIAL, nrTel);
startActivity(apelTel);
```

or

```
Uri nrTel = Uri.parse("tel:1234");
Intent apelTel2 =
 new Intent(Intent.ACTION_CALL, nrTel);
startActivity(apelTel2);
```

In both cases user will be aware of the calls if he or she looks at the screen.
For accessing the ***external storage***, the application needs to have permission also (like *WRITE_EXTERNAL_STORAGE*). The following code is used to write a file on the external storage:

```
try
{
//get the external storage state
String stareSD =
Environment.getExternalStorageState();

//if it is mounted
if(stareSD.equals(Environment.MEDIA_MOUN
TED))
{
   //get the directory
   File rootSDDir =  Environment.
       getExternalStorageDirectory();

   if (rootSDDir != null)
   {
     //read and write files
   }
}
catch (IOException ex)
{ Log.e("PDM1", ex.getMessage()); }
```

It is important to check the status of external media to avoid exception.
To ***access the location*** the GPS and network can be used. The location providers will send to associated listener the device location:

```
//initialization
LocationManager locationManager = (Loca-
tionManag-
er)getSystemService(Context.LOCATION_SER
VICE);

//associate the listener
locationManager.requestLocationUpdates(
LocationManager.GPS_PROVIDER, 0, 0, lo-
cationListener);

//...
```

```
//location listener implementation
LocationListener locationListener = new
LocationListener()
{
@Override
public void onLocationChanged(Location
poz)
 {
       if (poz != null)
       {
        //use location:
        //poz.getLatitude(
        //poz.getLongitude()
        //poz.getAltitude()
        //etc.
       }
}
//...
}
```

The *ACCESS_FINE_LOCATION* permission is required to run the code properly. If the GPS is unavailable, the *LocationManager.NETWORK_PROVIDER* can be used, but with less accuracy.
In [12] is presented an example of an Android application that uses Web services to access remote data. The Web services could be hosted on the attacker's servers and to be used to upload users' private data collected using some of the presented examples.

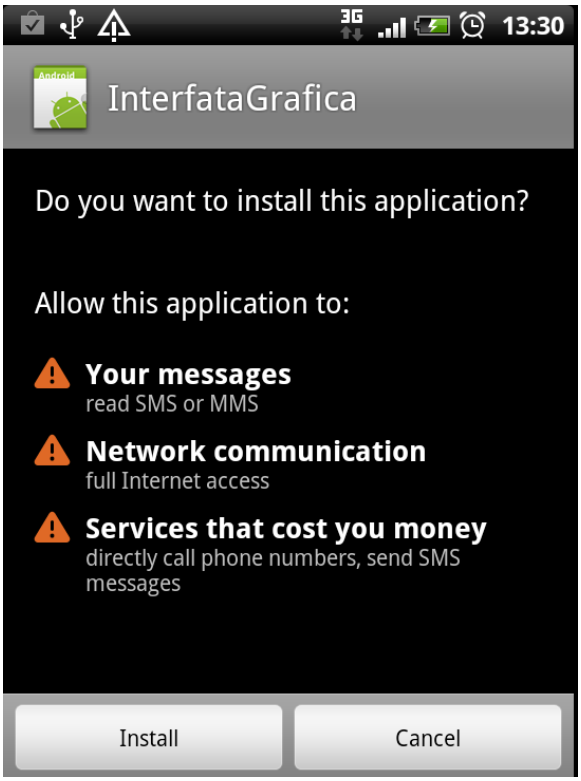

Fig. 4. Application permissions screen

Figure 4 shows an example of an installation process, when the user is informed about the permissions required by the application. These permissions are required when using some of code presented above.
If the user is not sure about the application he or she can cancel the installation process.
Another potential risk is the possibility to access the native code using NDK.

5 Prevention Measures

Installation from trusted sources and based on other users reviews and scores is very important when installing an applications. If there will be complaints from users regarding the spyware or malware built in the application the users will be informed.
Users have to read carefully the application permissions and if they have doubts regarding the application intentions they have to cancel the installation. It is very important for users to know the permissions with higher risks and to check them with the application's features.
Many free applications require full Internet access for advertising.
Another option is to install applications that monitor the system for malicious code preventing the infections. The number of these applications is growing due to increase number of malicious attack.
One solution will be to encrypt their users' sensitive data using specialized applications or using their own applications. Such kind of system can encrypt and decrypt data on request or they can monitor services and activities and make these actions on the fly, as data arrive (SMS, e-mails, SQLite databases interrogations, files etc.).
Figure 5 shows the components of the proposed system. Messages sent by the system or applications are intercepted by specialized broadcast receivers implemented as services (or applications). Received data is encrypted and stored in a database or on the external storage. Some of the intercepted messages are not delivered to default applications in order avoid to write plain data on the file system or databases. When data is required by some applications, it is extracted from the SQLite databases, decrypted, and sent to those applications.

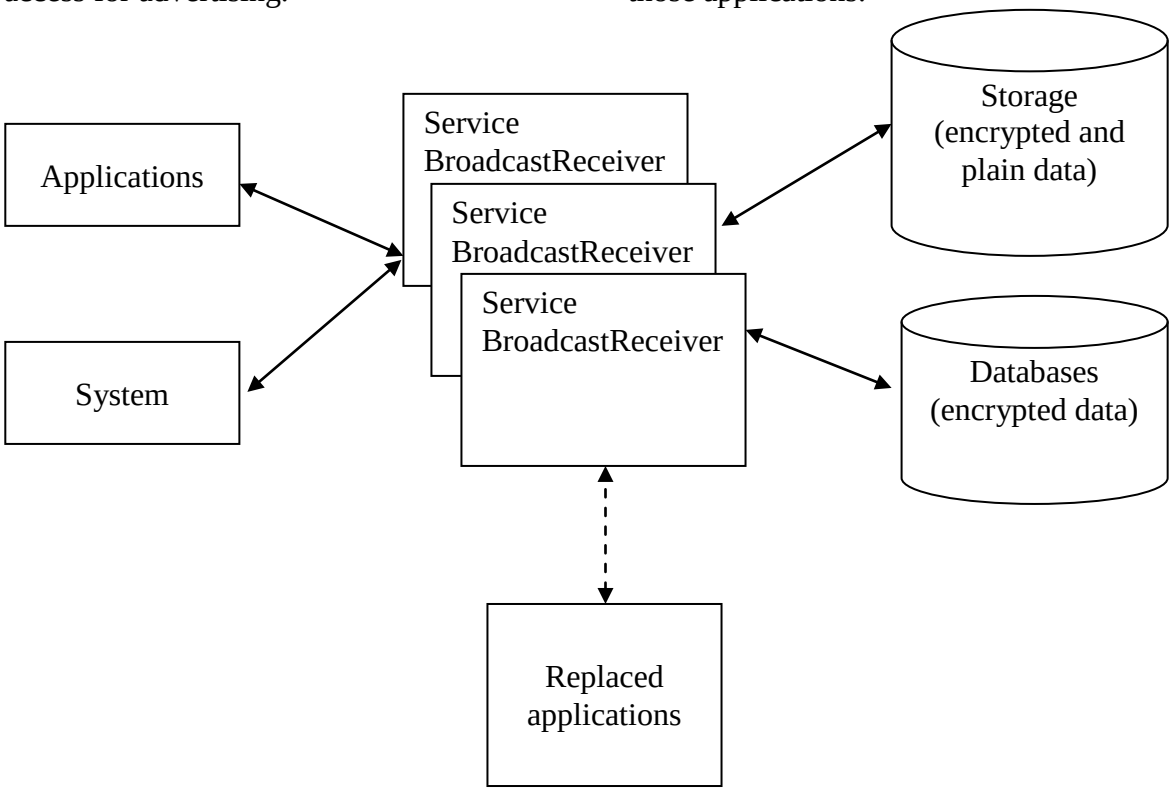

Fig. 5. Cryptographic system architecture

In order to encrypt and decrypt data, classes from `javax.crypto` package can be used.

The package includes classes for symmetric key cryptography (AES, DES), public keys encryption (RSA, DH) and message digests (MD5, SHA-1 etc.).
The following example uses DES algorithm (with ECB mode and PKCS#5 padding scheme) to encrypt and decrypt strings. Because it uses strings as parameters it is necessary to convert to ASCII characters only the encrypted text in order to maintain its integrity [13].

```
import javax.crypto.*;
import org.kobjects.base64.Base64;

class SecCD
{
 public static String decripteazaDES(String sir, SecretKey cheie)
 {
       String sirDecriptat = null;

       try
       {
        Cipher cifDecriptare = Cipher.getInstance("DES/ECB/PKCS5Padding");
        cifDecriptare.init(Cipher.DECRYPT_MODE, cheie);
        sirDecriptat = new String(cifDecriptare.doFinal(Base64.decode(sir)));
       }
       catch(Exception ex)
       { Log.e("PDM1", ex.getMessage());}
       return sirDecriptat;
}

 public static String cripteazaDES(String sir, SecretKey cheie)
 {
       String sirCriptat = null;

       try
       {
        Cipher cifCriptare = Cipher.getInstance("DES/ECB/PKCS5Padding");
        cifCriptare.init(Cipher.ENCRYPT_MODE, cheie);
        sirCriptat = Base64.encode(cifCriptare.doFinal(sir.getBytes("UTF8")));
       }
       catch(Exception ex)
       { Log.e("PDM1", ex.getMessage());}
       return sirCriptat;
 }
}
```

The class *SecCD* is used for encryption and decryption as follows:

```
try
{
  //secret key is generated here
  SecretKey cheie =
       KeyGenerator.getInstance("DES").
       generateKey();
  //plain text to be encrypted
  String mesajDeCriptat =
              "Criptare cu DES";
  //encrypted text
  String mesajCriptat =
       SecCD.cripteazaDES(
       mesajDeCriptat, cheie);
  //decrypted text
  String mesajDecriptat =
       SecCD.decripteazaDES(mesajCriptat,
       cheie);
}
```

```
catch(Exception ex)
{ Log.e("PDM1", ex.getMessage());}
```

Using a similar approach all confidential data can be encrypted on the device. The classes could be easily adapted to be used with file streams and arrays of bytes.
The encryption will be applied for example to text and e-mail messages. In the following example an SMS receiver is implemented in order to get the messages before any other application:

```
private String SMS_RECEIVED_ACTION =
"andro-
id.provider.Telephony.SMS_RECEIVED";

BroadcastReceiver br = new BroadcastRe-
```

```
ceiver()
{
public void onReceive(Context context,
Intent intent)
{

 String act = intent.getAction();
 if(act.equals(SMS_RECEIVED_ACTION))
 {
  Bundle contMesaj = intent.getExtras();

  if (contMesaj != null)
  {
     Object[] cont =
       (Object[])contMesaj.get("pdus");
     SmsMessage mesaj =
     SmsMessage.createFromPdu(
     ((byte[])cont[0]));

      //here we encrypt the message
      // mesaj.getMessageBody()
      //stop broadcasting the message
      abortBroadcast();
     }
  }
}
}
```

Before register the receiver, an *IntentFilter* is initialized with the required action (*SMS_RECEIVED*) and the priority is set to a higher value in order to receive the message before other applications:

```
IntentFilter smsFilt =
       new IntentFilter();
//higher priority
smsFilt.setPriority(1000);
smsFilt.addAction(SMS_RECEIVED_ACTION);
this.registerReceiver(br, smsFilt);
```

It is important to unregister the receiver when it is not needed anymore. This is done by calling:

```
unregisterReceiver(br);
```

In order to work, the *SMS_RECEIVE* permission is required.

It is very important to focus on quality especially when building this type of application. In [14] and [15] is analyzed this aspect and several quality metrics were developed. These metrics helps to obtain high quality mobile applications.

The disadvantage of this security system is that it could slow down the system and maybe, some applications have to be replaced with similar ones.

6 Conclusions and future work

Mobile applications security is very important today due to increasing number of users and the importance of personal and confidential data stored on mobile devices.

Users' role is very important in reducing security threats. They have to be aware about the risks they expose when installing application from unknown or unsafe sources. Also, users have to pay attention when they install application to permission required by the applications. If the applications don't suppose to need some kind of permission the installation process has to be cancelled.

It is very important to remember that an important source of infection with malware is through the Web browser.

The system security can be improved by using specialized applications. Another option is to write applications that will encrypt and decrypt all private data as they are accessed in order to assure its confidentiality.

Acknowledgements

This work was supported by CNCSIS – UEFISCSU, project number PNII – IDEI 2637/2008, project title: *Project management methodologies for the development of mobile applications in the educational system*.

References

[1] The Associated Press, Smartphone malware infections growing fast [Online]. Available at: http://bulawayo24.com/index-id-technology-sc-mobile+phone-byo-6321-article-Smartphone+malware+infections+growing+fast+.html (August 2011)

[2] J. Leyden, Mobile app malware menace grows [Online]. Available at: http://www.theregister.co.uk/2011/08/04/mobile_malware_trends/ (August 2011)

[3] J. Scott, Mobile devices in danger of attack, [Online]. Available at: http://www.itpro.co.uk/634240/mobile-devices-in-danger-of-attack (August 2011)

[4] H. Dwivedi, C. Clark and D. Thiel, *Mobile Application Security*. New York: McGraw-Hill, 2010, pp. 364-389

[5] E. Burnette, *Hello, Android: Introducing Google's Mobile Development Platform, 3rd Edition.* The Pragmatic Bookshelf, 2010, p. 33.
[6] The Developer's Guide | Android Developers [Online]. Available at: http://developer.android.com/guide/index.html (March 2010)
[7] R.styleable | Android Developers [Online]. Available at: http://developer.android.com/reference/android/R.styleable.html#AndroidManifestPermission_protectionLevel (July 2011)
[8] android-apktool project [Online]. Available at: http://code.google.com/p/android-apktool/ (July 2011)
[9] smali project [Online]. Available at: http://code.google.com/p/smali/ (July 2011)
[10] dex2jar project [Online]. Available at: http://code.google.com/p/dex2jar/ (July 2011)
[11] JD-GUI | Java Decompiler [Online]. Available at: http://java.decompiler.free.fr/?q=jdgui
[12] P. Pocatilu, "Developing Mobile Learning Applications for Android using Web Services," *Informatica Economică*, Vol. 14, No. 3, pp. 106-115, September 2010
[13] J. Andress, *The basics of information security: understanding the fundamentals of InfoSec in theory and practice*. Syngress, 2011
[14] C. Boja and L. Batagan, "Analysis of M-Learning Applications Quality," *WSEAS Transactions on Computers*, issue 4, vol. 8, May 2009, pp. 767-777
[15] C. Ciurea, "A Metrics Approach for Collaborative Systems," *Informatica Economica*, vol. 13, no. 2, pp. 41-49, June 2009.

Paul POCATILU graduated the Faculty of Cybernetics, Statistics and Economic Informatics in 1998. He achieved the PhD in Economics in 2003 with thesis on Software Testing Cost Assessment Models. He has published as author and co-author over 45 articles in journals and over 40 articles on national and international conferences. He is author and co-author of 10 books, (Software Testing Costs, and Object Oriented Software Testing are two of them). He is associate professor in the Department of Economic Informatics of the Academy of Economic Studies, Bucharest. He teaches courses, seminars and laboratories on Mobile Devices Programming, Economic Informatics, Computer Programming and Project Management to graduate and postgraduate students. His current research areas are software testing, software quality, project management, and mobile application development.