On the Construction of a False Digital Alibi on the Android OS

Pietro Albano*, Aniello Castiglione†, Giuseppe Cattaneo§, Giancarlo De Maio¶, Alfredo De Santis‡
Dipartimento di Informatica "R.M. Capocelli"
Università degli Studi di Salerno
I-84084 Fisciano (SA), Italy
pietro.albano@gmail.com*, castiglione@ieee.org†, cattaneo@dia.unisa.it§, demaio@dia.unisa.it¶, ads@dia.unisa.it‡

***Abstract*—Digital evidence can determine either the conviction or acquittal of a suspect. In the latter case, such information constitutes a digital alibi. It has been recently shown how it is possible to set up a common PC in order to produce digital evidence in an automatic and systematic manner. Such traces are indistinguishable post-mortem from those left by human activity, thus being exploitable to forge a digital alibi. Modern smartphones are becoming more and more similar to PCs, due both to their computational power as well as their capacity to produce digital evidence, local or remote, which can assume a probative value. However, smartphones are still substantially different from common PCs, with OS limitations, lack of tools and so on, thus making it difficult to adopt the same techniques proposed for PCs to forge a digital alibi on a mobile device.**

In this paper novel techniques to create a false digital alibi on a smartphone equipped with the Android OS are presented. In particular, it is possible to simulate human interaction with a mobile device using a software automation, with the produced traces being indistinguishable post-mortem from those left by a real user. Moreover, it will be shown that advanced computer skills are not required to forge a digital alibi on an Android device, since some of the presented techniques can be easily carried out by non-savvy users. This emphasizes how the probative value of digital evidence should always be evaluated together with traditional investigation techniques.

***Index Terms*—Digital Forensics; Mobile Forensics; Anti-Forensics; Mobile Anti-Forensics; Counter-Forensics; Android Forensics; Digital Evidence; Digital Investigation; False Digital Evidence; False Alibi; Digital Alibi.**

I. Introduction

In recent years smartphones have become widely used and are replacing traditional mobile phones. They support complex applications, such as web browsers, e-mail clients, navigation systems and so on. The computational and storage capacity of a smartphone can be compared to that of a common PC. Moreover, OSes for smartphones make it possible to execute most of the operations supported by the traditional OSes for PCs. With respect to portable devices such as notebooks and netbooks, the main advantage of smartphones is that they can be used everywhere, thanks to their size.

Smartphones are personal devices and contain sensitive user information, such as contacts, text messages, call lists, as well as e-mails, websites visited, geo-location data, etc.. From a digital forensics point of view, a smartphone is one of the most important sources of evidence. A mobile device also produces a number of traces on remote locations, for example, regarding its registration with a base station, which can provide a fairly accurate position of the user at a given time, as well as accesses to services via Internet and so on. All this information can provide a well-defined profile of the user and makes it possible to reconstruct his temporal activities. Usually remote traces have a more probative value than local ones, since they are, in a sense, validated by the trusted third-parties to which they belong. For example, an user cannot manipulate his cell phone transcripts at the mobile operator, nor modify the posting time of a message on Facebook. Such evidence can be considered reliable in the context of a digital alibi.

There has been a significant increase in the amount of digital evidence being brought into Courts. Digital devices can be subject to forensics investigations in order to collect useful traces about the behavior of a person involved in a proceeding, to be either cleared of an accuse or charged with an offense.

A recent work [1] highlighted how Courts should be careful when considering the admissibility and probative value of digital evidence, since an individual (not necessarily skilled) can setup a software automation which is able to generate digital traces on a PC being indistinguishable in a post-mortem analysis from those produced by human activity. Such an automation can be exploited in order to construct a *false digital alibi* or, in other words, a set of "reliable" digital evidence proving to the Court that the user was acting on his PC during a specific timeline - if in reality he was elsewhere.

The aim of this work is to show that a similar approach can be adopted in order to create a false digital alibi by means of a common smartphone equipped with the Android OS. While a lot of tricks can be employed to accomplish this task on a PC, the inherent limitations of mobile devices - limited OSes, inability to configure some parameters, lack of tools and so on - lead to the adoption of different strategies. The lack of exploitable tools on Android is mostly due to the strict permission constraints imposed by the OS.

The construction of a digital alibi includes the implementation of an automation strategy as well as the application of a method to avoid/erase any unwanted evidence left by the automation on the involved device(s). The automation strate-

Corresponding author: Aniello Castiglione, Member, *IEEE*, castiglione@ieee.org, Phone: +39089969594, FAX: +39089969821

gies presented in this work are divided into two classes: *local techniques*, which make use of procedures local to the Android device, and *remote techniques*, whose execution is controlled from a common PC. Both cases include novel strategies that do not require advanced computer skills, but also more complex techniques which require some programming knowledge.

In Section II the permission mechanism of Android is analyzed. In Section III techniques viable to creating an automation for Android are presented, while in Section IV an analysis of the possible evidence left by their execution is dealt with. In Section V the development and testing phases of the proposed techniques are discussed. In Section VI, a real case study is analyzed, ending with the authors conclusions in Section VII.

II. Permissions in the Android OS

Android is an open-source operating system for mobile devices developed by the Open Handset Alliance, a business alliance of 83 companies led by Google [2]. Android consists of a kernel derived from the Linux kernel, a middleware layer which includes a set of system libraries and a runtime environment, as well as an application layer which includes APIs and applications [3].

The Android system provides a signature-based permissions enforcement, so that applications can share functionality and data only if signed by the same developer. This inherent limitation of the OS is the main cause for the lack of fully-fledged tools to automate actions in Android. As a consequence, all the automation techniques for a PC presented in [1] are difficult to implement in such a context, since they are all based on the concept of a "controller process" (the automation) injecting events (clicks, keystrokes, etc.) in some "slave processes" (word processor, browser, etc.).

This section is followed by a discussion about the previously mentioned mechanism.

A. Security and Permissions

Android is a privilege-separated OS, where each application runs with a distinct system identity (the Linux *user ID* and *group ID*). A central design point of the Android security architecture is that no application, by default, has permission to perform any operations that could adversely impact other applications, the operating system, or the user. This includes reading or writing the user private data (such as contacts or e-mails), reading or writing another application files, performing network access, keeping the device awake, etc.

The kernel is solely responsible for "sandboxing" applications from each other, which implies that applications must declare the permissions they need for additional capabilities not provided by the basic sandbox functionalities. Such permissions can be statically declared in the application package, and are prompted by the OS to the user at installation time, which let the user choose whether to proceed or not.

An Android package (.apk file), containing an application, has to be signed by the developer and has to include the certificate that makes it possible to verify its signature. The certificate does not need to be signed by a Certificate Authority: usually Android applications include self-signed certificates, whose purpose is only to distinguish the different authors of applications.

At installation time, the Android OS gives each package a distinct Linux user ID, which remains the same for the entire lifetime of the application on the device. Since security enforcement happens at process level, two applications having different user IDs cannot interact (i.e., share functionalities and data) by default. It is possible to overcome this limitation by assigning the same user ID to the `sharedUserId` attribute in the file `AndroidManifest.xml` of the involved packages, which is a request to the OS to share the same user ID. It is important to note that the kernel accepts such requests only if the packages are signed by the same author (therefore have the same certificate) [4].

III. Automation Techniques

The techniques proposed in this work, for the creation of a false digital alibi by means of an Android device, is based on the creation and use of an "automation", which is a program that performs a predetermined set of operations on the smartphone within a certain timeline. The aim of such an automation is to produce evidence indistinguishable by a digital forensics analysis from those left by the real user activity. This can be exploited by the user in order to forge a digital alibi which could be valid in the context of a legal proceeding. Even though advanced techniques could be evaluated (e.g., implementing low-level automations compiled to native ARM language), the aim of this work is to show that the result can be also obtained by adopting approaches viable by non-skilled users.

The proposed automation techniques are divided into two categories: *local techniques*, which are executed only on the smartphone; *remote techniques*, which are generally simpler to implement but require a secondary control device - i.e., a common PC.

A. Local Techniques

Several techniques that make it possible to create an automation that only require an Android device are presented in this section. Some simple approaches which do not require particular computer skills are firstly addressed, then an analysis of slightly more complex techniques, which extend the set of reproducible scenarios, is dealt with.

1) Simple Automation: The aim of these basic techniques is to show that advanced skills are not required to forge a digital alibi by means of an Android device. The idea is to exploit some simple applications, freely available on the Android Market, which make it possible to trigger automatic actions at the occurrence of certain events. Applications such as AutomateIt [5] and Tasker [6] can be used to implement this technique.

The triggers are usually system events such as the reception of a text message, or an incoming call from a specific number.

The triggered actions can usually be chosen from a predetermined set, and usually make it possible to enable/disable mobile data connectivity, kill applications, send text messages, place or answer calls and so on. From the point of view of the digital alibi, evidence produced by sending a text message or placing a call should appear in the communication transcripts maintained by the mobile operator, and could be accepted in the context of a legal proceeding. However, the set of scenarios being reproducible using such an approach is generally limited. This is mostly due to the permission constraints (see II-A) imposed by the Android OS, which denies a full interaction among applications belonging to different packages. In other words, it is not possible to simulate activities that require interactions with other applications, including a browser to surf the web, the Facebook application to post a message, as well as a text editor in order to modify a document.

2) Event-Injection: The aim of the event-injection technique is to overcome the limits of the previous technique, i.e. create an automation able to simulate a full interaction of the user with the smartphone. The idea is to directly inject events (clicks, keystrokes and so on) into applications executing on the device. In order to accomplish this task, it is possible to exploit some professional testing tools, such as Robotium [7], which makes it possible to directly inject events into third party Android software. It usually requires the implementation of a "tester" program that should be installed on the device and realizes the interaction with the involved applications components. However, even though the APIs provided by such frameworks are generally easy-to-use, the need for programming skills makes such an approach more difficult to be realized with respect to the previous one. An automation created using this technique can simulate any scenario involving the use of applications: the posting on Facebook, the modification of a text document stored on the device and so on.

3) Event-Replication: The event-replication approach is based on the idea of capturing events received by the input devices of the Android smartphone in order to replicate them later. For each input device (touchscreen, keyboard, light sensor, etc.) the Android kernel allocates a virtual device (usually `/dev/input/eventX`) which can be used to both read occurring input events and generate new input events. Android also provides two system tools, namely `getevent` and `sendevent`, that make it possible to read and generate input events via shell commands. This technique can potentially replicate any user activities, previously recorded with `getevent`, by means of a shell script which performs the same actions through a series of `sendevent` commands. While a very basic knowledge of shell scripting is required in order to realize this strategy, the correct functioning of the automation is strictly related to the "current state" of the smartphone. For example, a slight change in the graphics, as well as an unexpected slowdown in loading an application could compromise the correct generation/reception of the events.

B. Remote Techniques

The mobile device can be remotely controlled from a secondary device connected to it, such as a common PC. As a consequence, an automation running on the controller machine can indirectly perform actions on the Android device in order to produce evidence for a digital alibi. Although an automation that is implemented following this strategy is simple and powerful, it needs more attention in the analysis of the possible unwanted evidence left on the devices.

This technique requires the combined use of three elements: a communication channel, a remote control software and an automation program.

1) Communication: A persistent connection between the controller device and the smartphone should be maintained for the entire execution of the automation procedure. While the standard in-box USB cable can be used to physically connect the devices, some expedients can be adopted in order to make it work over a wireless channel. Communication with the smartphone requires an Android Debug Bridge (ADB) daemon running on the Android system. ADB is already present on any default installation of Android and can be enabled by switching some simple system settings, thus not requiring any particular skills nor suspicious packages to be installed on the device.

2) Remote Control: A remote control software is required in order to issue commands from the controller device to the smartphone. Open-source tools, such as Android Screencast [8] and Droid@Screen [9], are particularly useful for this purpose, since they implement a graphical remote control interface, usually called "simulator", which is able to reproduce the current screen view of the Android device on the Windows/*NIX system connected to it. Mouse clicks on that interface are translated into touch events on the smartphone screen, as well as keystrokes on the controller system generate keyboard events on the mobile device.

3) Automation: As previously stated, the remote technique requires a simulator in order to reproduce the smartphone screen on a controller machine. Clearly, all the operations being reproducible through the simulator can be also performed by a software automation running on the controller system. Substantially, the same techniques viable to implementing an automation for a PC [1] can be reused in order to create an automation for a smartphone equipped with Android. A tool like Sikuli [10] is particularly useful when employing such an approach. It is a framework projected to automate the test of GUIs using screenshots, and that can be exploited to create powerful automations for mobile devices. In fact, an automation implemented with Sikuli can interact with anything seen on the screen of the mobile device, which is in turn seen on the screen of the controller machine. Any other tools based on screen coordinates, such as AutoIT [11] and Automator [12], can be also exploited for the same purpose.

IV. Digital Evidence of an Android Automation

Most of the techniques proposed in this paper can leave unwanted traces on the involved system(s). In this section

some guidelines to identify and avoid/remove such traces are presented. A comprehensive analysis of the evidence detectable by a digital forensics analysis on an Android system is addressed in [13].

A. Application Data

At installation time the Android package manager creates a distinct folder for each package, usually located in `/data/data`, which is inaccessible for reading and writing operations by any other applications. For example, for the Facebook application the OS creates the folder */data/data/com.facebook.katana*. This directory stores application data, including executable files and application logs, and will be "logically removed" (i.e., through the common `unlink` operation) by the OS when the package is uninstalled.

Application traces can be also left in other different filesystem locations: in fact, packages could require additional permissions at installation time, including read/write access on the SD card. Generally, data created on the SD card is not removed at uninstallation time. An automation constructed by following the simple or the event-injection techniques require the installation of third party applications on the Android system, which should be completely removed after the automation procedure. The set of unwanted traces could include files created on the SD card.

The simple uninstallation of a package is not sufficient to completely remove its presence from the device. The method analyzed in [14] can be adopted instead in order to delete applications and data from an Android system in a secure manner.

B. Non-Sandboxed Code Execution

Some software such as native executables and shell scripts can be executed outside of the sandbox mechanism described in II-A, thus bypassing the standard permission constraints. More generally, any software executed with root permissions (see IV-D) can potentially write everywhere on the filesystem (except on partitions mounted in read-only mode). In substance, any data produced by an automation procedure requiring root access to the system should be carefully identified.

It is important to note that usually the execution of shell commands is not logged in Android, while system logs are written by default only on the RAM [13]. Due to their volatile nature, this work does not considered them as a potential unwanted trace, since a simple system reboot should be sufficient to erase them.

The event-replication technique requires the execution of some shell commands with root permissions on the Android system. However, all the data written on the filesystem by the procedure can be easily isolated in a specific location, such as a directory on the SD card, which can be securely deleted.

The remote control tools employed to realize remote-control strategies usually leave unwanted evidence on the Android device. For example, Android Screencast copies a file named `InjectAgent.jar` (containing the program in charge of injecting keyboard/mouse events on the smartphone) in the directory `/data/local/tmp/`. This operation is transparently performed by sending shell commands to the mobile device through ADB, which are clearly executed outside the Android sandbox mechanism. Although this file is automatically removed by the application upon exiting, it would be advisable to apply the aforementioned sanitization strategy.

C. Certificate Incoherence

As discussed in II-A, any applications running in Android are signed by the author and include a digital certificate which makes it possible to verify the signature. Moreover two applications can fully interact only if signed by the same author. Considering the event-injection technique, the tester program and any other applications controlled by the automation should have the same certificate. It requires that any involved packages are re-signed by the same author and re-installed on the mobile device. Clearly, the presence of proprietary software having a certificate different from the original one can be considered at least suspicious in the context of a digital forensics analysis. In order to minimize the unwanted traces, the tester should be securely deleted. A paranoid approach could include the secure removal of any re-signed applications by using the same technique. Moreover, the original version of any re-signed applications should be re-installed in order to avoid the certificate incoherence.

D. Superuser Access

Most of the techniques discussed in III, except the simple approach and the event-injection strategy, require superuser access to the Android OS. Unfortunately, the logon as the `root` user is usually disabled by default on Android. There are some procedures that could be followed in order to unlock this feature (many tutorials are available on [15]), which do not require advanced computer skills.
In the authors opinion, such evidence cannot be considered suspicious in the context of a digital forensics analysis, since it is very common among the Android users.

E. Digital Evidence on Secondary Devices

When adopting a remote control strategy, most of the supporting tools are installed on the controller device. In particular, the Android SDK is required to run the ADB client on the PC, as well as some tools in order to execute the automation. Clearly, the automation program is itself an unwanted trace that has to be considered.

All the precautions discussed in [1] can be adopted in order to avoid as much unwanted evidence as possible on the controller machine (as it is a common PC), while the secure deletion methodology presented in [16] can be implemented in order to remove the remaining ones.

V. Development and Testing

In order to minimize the amount of data that can potentially lead back to the use of an automation, the development and testing phases should be accomplished in an isolated environment. Possible unwanted traces being produced during such phases are analyzed in this section.

Local Techniques: No particular precautions should be taken for the implementation of the simple technique. In fact, the previously mentioned applications also provide the development environment for the automation. The traces left during this phase are generally contained in the same filesystem directory containing the private data of the package, therefore it can be removed during the sanitization phase. The implementation of the event-injection technique requires a more sophisticated environment. In particular, the Android SDK as well as the specific framework libraries should be installed on the system. Clearly, this implies that the development of the automation should be done on a PC. The unwanted traces left during this phase can thus be avoided/removed/obfuscated by adopting the methods exposed in [1]. The resulting automation can be properly tested on the same development environment making use of the standard emulator provided by the Android SDK.

The implementation of the event-replication technique can be accomplished on the same Android system as well as a secondary PC connected to the smartphone through ADB. In the former case, all the produced evidence can be isolated in a specific filesystem directory, so that it can be removed during the sanitization phase. While in the latter, the same precautions mentioned for the development of the event-injection technique can be adopted.

Remote Techniques: The implementation of the remote technique requires a controller system, with the Android device being connected to it in order to properly construct an automation based on the smartphone screen view. Clearly, all the elements on which the technique is based - a communication channel, a remote control interface and an automation software - should be set-up on the development environment.

The precautions advised in [1] can be taken in order to minimize any unwanted traces on the controller machine, while the same methods proposed for the sanitization can be adopted in order to remove those left on the Android device by the remote control tools.

VI. Case Study

In this section a real case study focusing on the production of a false digital alibi by means of an Android smartphone is analyzed.

Several experiments have been conducted by following the simple technique with different automation tools, such as AutomateIt and Tasker, and different smartphone models such as HTC Desire Z and HTC Nexus One. In particular, an automatic SMS exchange between these devices was implemented, driven by both time events and SMS contents. The unwanted traces produced by the involved tools on the devices were removed by adopting the sanitization method presented in [14]. A digital forensics analysis on the smartphones, conducted by following the techniques presented in [13], revealed no significant unwanted evidence.

Since the simple technique is very easy to implement and does not require detailed explanations, the case based on a technique of medium difficulty, namely the event-replication, is discussed.

In order to minimize the risks of any unwanted evidence being left on the involved devices, any unnecessary writing operations on persistent memories (e.g., hard disks and smartphone storage) were avoided. A digital forensics analysis on the devices involved in the experiment revealed no significant traces about both the preparatory phase (i.e., the development and testing) and the definitive execution of the automation.

A. Operational Environment

The machines used for the experiments consisted of an Android device, namely an HTC Desire Z, together with a common PC. A standard in-box USB cable was used to maintain a connection between the devices for the entire preparatory phase of the automation. Any communications took place by means of ADB. The smartphone came equipped with a modified version of Android 2.3.4 - namely Cyanogen Mod 7.1.0-RC1 - providing the superuser access that was required to implement the event-injection technique. Any operations involving the PC were executed from a live Linux distribution - Ubuntu 11.04.

The only software tool installed on the Android device during the experiments was Script Manager [17], an application which makes it possible to run a shell scripts at a specific time. The presence of this tool should not raise any suspicion in a digital forensics analysis, since it is commonly used as well as freely available on the Android Market. Even though the Android SDK is required to use ADB, it produces no permanent traces on the PC since it is installed on the live Linux distribution.

B. Construction of the Automation

The event-replication technique consists of two steps: the recording of the events to be cloned and their replication. In this case study, only the events generated by the interaction with the touchscreen were considered in order to realize the automation. The events were captured and saved in a text file on the live system, named `events.txt`. Precisely, it was accomplished by executing the following command:

```
$ adb shell getevent -t /dev/input/event1 > events.txt
```

where `-t` indicates to output the timestamp of each event and `/dev/input/event1` represents the touchscreen device (it may vary depending on the smartphone).
The following dump is an example of the output generated by the previous command for a single pressure on the screen.

```
6559-132416: 0003 0030 0000003f
6559-132447: 0003 0032 00000005
6559-132477: 0003 0035 000000f6
6559-132477: 0003 0036 000002a7
6559-132508: 0000 0002 00000000
6559-132508: 0000 0000 00000000
6559-171845: 0003 0030 00000000
6559-172333: 0000 0000 00000000
```

Each outputted line is composed of a timestamp and three hexadecimal values indicating the event occurred.
The result of the recording phase was processed in order to implement a simple shell script which replicates the same actions by means of the `sendevent` command, which takes

in input the same - decimal-coded - values given in output by the `getevent` command. The timestamps were useful in preserving the timing of the events, which was implemented by interleaving the instructions with some `sleep` commands.

An excerpt of the shell script obtained by processing the `events.txt` file - corresponding to the same pressure event examined before - follows.

```
sendevent /dev/input/event1 3 48 63
sendevent /dev/input/event1 3 50 5
sendevent /dev/input/event1 3 53 246
sendevent /dev/input/event1 3 54 679
sendevent /dev/input/event1 0 2 0
sendevent /dev/input/event1 0 0 0
sendevent /dev/input/event1 3 48 0
sendevent /dev/input/event1 0 0 0
```

According to this approach, some different shell scripts were implemented in order to reproduce a plausible activity timeline, with it being exploitable by a person in order to forge a digital alibi. In particular, an automation performing the following activities was implemented in this case study.

1) Posting to Facebook: The first action performed by the automation is the posting of a status message on Facebook. It reproduces a touch on a link of the Facebook application on the home screen. After some seconds, the scripts sends a pressure to the appropriate text section, thus simulating the use of the virtual keyboard to write a message. Finally, the automation reproduces the pressure on the "Send" button and restores the home screen view by touching the "Home" button.

2) Sending a SMS: Starting from the home screen, the automation simulates a touch on the messaging application link in the application bar. Similarly to the previous activity, it generates the appropriate input events in order to write an SMS, and finally sends it to a predetermined telephone number.

C. Development and Testing

The recording phase of the the proposed technique was remotely conducted from the same PC. In particular, the `getevent` command was sent through a remote connection over ADB, and the output was recorded on the PC. The result of this operation was processed in order to obtain the corresponding sequence of `sendevent` commands. A resulting shell script of this sequence was executed several times from the PC on the smartphone in order to verify that all the proper actions were correctly executed. The development and testing of the automation on a live Linux distribution made it possible to avoid the creation of any meaningful traces on both the hard disk of the PC as well as the smartphone.

Prior to its execution finalized to the construction of a false digital alibi, only the final version of the automation script was loaded onto the external memory of the smartphone. The device was sanitized adopting the method presented in [14]. A digital forensics analysis conducted by the authors on the involved device, following the directives presented in [13], revealed no meaningful traces about the automation procedure.

VII. Conclusions

This work highlights how it could be possible to artificially create a false digital alibi, plausible in the context of a legal proceeding, by exploiting some features of an Android device. The proposed techniques make use of a software automation which is able to fully simulate a real series of human activities. It is worth noting that advanced skills are not required to implement such techniques.

A real case study was also analyzed in order to validate the authors thesis. The results stress the need of a constant effort by digital forensics experts to upgrade and keep update their knowledge about new technologies. In addition, the work underlines that Courts should consider digital evidence as a part of a larger pattern of behaviour reconstructed by means of traditional forensics techniques.

Acknowledgements

The authors are grateful to the President of the IISFA Italian Chapter, Gerardo Costabile, who remarked the relevance of the subject of this paper in Courts. A special thank goes to Mario Ianulardo, lawyer in computer crime, for the endless and interesting discussions on the probative value of a false digital alibi.

References

[1] A. De Santis, A. Castiglione, G. Cattaneo, G. De Maio, and M. Ianulardo, "*Automated Construction of a False Digital Alibi*," in *MURPBES*, ser. Lecture Notes in Computer Science, A. M. Tjoa, G. Quirchmayr, I. You, and L. Xu, Eds., vol. 6908. Springer, 2011, pp. 359–373.

[2] Open Handset Alliance, http://www.openhandsetalliance.com/, 2011 (accessed July 2011).

[3] Android Developers, http://developer.android.com, 2011 (accessed July 2011).

[4] Android Developers, "Security and Permission Android Developers," http://developer.android.com/guide/topics/security/security.html, 2011 (accessed July 2011).

[5] Android Muzikant, "Android Muzikant: AutomateIt Pro," http://muzikant-android.blogspot.com/2011/05/automateit-goes-pro.html, 2011 (accessed July 2011).

[6] Crafty Apps, "Tasker for Android," http://tasker.dinglisch.net/, 2011.

[7] Robotium Developers, "Robotium it is like Selenium, but for Android," http://code.google.com/p/robotium/, 2011 (accessed July 2011).

[8] A. Thiel, "androidscreencast - Desktop app to control an android device remotely," http://code.google.com/p/androidscreencast/, 2011 (accessed July 2011).

[9] J. Riboe, "Droid@Screen," https://github.com/ribomation/DroidAtScreen1, (accessed July 2011).

[10] Sikuli.org, "Project Sikuli," http://sikuli.org/, 2011 (accessed July 2011).

[11] J. Bennet, "AutoIt v3.3.6.1," http://www.autoitscript.com/autoit3/, April 2010 (accessed July 2011).

[12] Apple Inc., "Apple Automator," http://www.macosxautomation.com/automator/, 2010 (accessed July 2011).

[13] A. Hoog, *Android Forensics: Investigation, Analysis and Mobile Security for Google Android*, J. McCash, Ed. Syngress, 2011.

[14] P. Albano, A. Castiglione, G. Cattaneo, and A. De Santis, "A Novel Anti-Forensics Technique for the Android OS," in *BWCCA*. IEEE Computer Society, 2011.

[15] xda-developers, "xda-developers forum," http://forum.xda-developers.com/, 2011 (accessed July 2011).

[16] A. Castiglione, G. Cattaneo, G. De Maio, and A. De Santis, "Automatic, Selective and Secure Deletion of Digital Evidence," in *BWCCA*. IEEE Computer Society, 2011.

[17] devwom, "Script Manager - devwom," https://sites.google.com/site/devwom/script-manager, 2011 (accessed July 2011).