IEEE Access
practical innovations : open solutions

Received April 17, 2013, accepted April 24, 2013, published May 10, 2013.

Digital Object Identifier 10.1109/ACCESS.2013.2260817

Automated Production of Predetermined Digital Evidence

ANIELLO CASTIGLIONE (Member, IEEE), GIUSEPPE CATTANEO, GIANCARLO DE MAIO, AND ALFREDO DE SANTIS (Member, IEEE)

Department of Computer Science, University of Salerno, Via Ponte don Melillo, Fisciano I-84084, Italy

Corresponding author: A. Castiglione (castiglione@ieee.org)

ABSTRACT Digital evidence is increasingly used in juridical proceedings. In some recent legal cases, the verdict has been strongly influenced by the digital evidence proffered by the defense. Digital traces can be left on computers, phones, digital cameras, and also on remote machines belonging to ISPs, telephone providers, companies that provide services via Internet such as YouTube, Facebook, Gmail, and so on. This paper presents a methodology for the automated production of predetermined digital evidence, which can be leveraged to forge a digital alibi. It is based on the use of an automation, a program meant to simulate any common user activity. In addition to wanted traces, the automation may produce a number of unwanted traces, which may be disclosed upon a digital forensic analysis. These include data remanence of suspicious files, as well as any kind of logs generated by the operating system modules and services. The proposed methodology describes a process to design, implement, and execute the automation on a target system, and to properly handle both wanted and unwanted evidence. Many experiments with different combinations of automation tools and operating systems are conducted. This paper presents an implementation of the methodology through VBScript on Windows 7. A forensic analysis on the target system is not sufficient to reveal that the alibi is forged by automation. These considerations emphasize the difference between digital and traditional evidence. Digital evidence is always circumstantial, and therefore it should be considered relevant only if supported by stronger evidence collected through traditional investigation techniques. Thus, a Court verdict should not be based solely on digital evidence.

INDEX TERMS Digital evidence, digital investigation, digital forensics, antiforensics, counter-forensics, automated alibi, false alibi, digital alibi, false digital alibi.

I. INTRODUCTION

People's lives are depending more and more on technology. In addition to the workplace, digital devices are currently used by most people to manage personal business and to always be available. The number of Internet users in the world is more than 2.4 billion, with a penetration of 34.3% of the population [1]. In this context, crime also takes advantage from technology. Computer crime, or cybercrime, refers to any crime that involve a computer and a network, which may be used in the commission of a crime or may be the target [2]. According to the Norton Cybercrime Report 2012 [3], 2/3 of online adults in the world have been victim of cybercrime in their lifetime, with an estimated global cost of $110 billion annually. Also in the case in which a digital device has not been directly used for the commission of a crime, it may be subject to a forensic investigation aimed to collect useful information about the accused. Such information may be extracted from digital documents, photos, messages, emails and so on.

Any probative information stored or transmitted digitally, which can be used by a party in a judicial dispute in Court, is referred to as *digital evidence* [4]. In recent years the use of digital evidence has increased exponentially. Consequently, Digital Forensics are becoming more and more concerned about the admissibility and the probative value of digital evidence. Digital evidence should be carefully examined by the Court before being considered reliable, since it is *ubiquitous* and *immaterial*. Moreover, it is fundamental to make a distinction between *local* and *remote* digital evidence.

Ubiquitous: Digital evidence is ubiquitous, since it can be located anywhere in the world. Digital data can be easily moved from one device to another on the other side of the

world. It may reside on mobile equipment (phones, PDAs, laptops, GPSes, etc.) and especially on servers that provide services via Internet. The device under investigation may be located in a Country different from that where the crime has been committed, with it being an obstacle for the acquisition of digital evidence.

Immaterial: Digital evidence is also immaterial, since it is just a sequence of ones and zeros. It can be easily tampered with by the owner of the device, since the owner has full access to any software and hardware component. In the case in which the evidence is stored on a remote location, it might be modified or lost over time without any control over it.

Local: A digital evidence is referred to as local in case the information is stored on a device owned by the accused. In most cases, the Court can order the seizure of the inquired device. Such evidence can be extracted from digital documents, browser history and so on. Local evidence tampering is simple for an accused having basic technical skills.

Remote: A remote evidence is related to an information stored on a remote machine. Remote evidence is difficult to be tampered with by the accused, since it would require unauthorized access to the remote system or the intervention of an accomplice. Remote digital evidence can be extracted from online social networks, emails stored on a server and so on. With respect to local digital evidence, they may be considered more reliable since validated, in a sense, by the company providing the service (Google, Facebook, etc.). In practice, the company may act as an implicit trusted-third-party in the trial.

A. DIGITAL ALIBI

Digital devices, and in particular personal computers, can be involved in a forensic investigation for several reasons. They may have been used to commit the crime or may have been the target of a cybercrime. In general, a computer may be analyzed to track the activity of an individual involved in a legal case. Digital devices may also contain evidence that can be used to clear the charged, for example, proving that he was working at his personal computer while the crime was committed. In such a case the digital evidence constitutes an alibi. *Alibi* is a term deriving from the Latin expression *alius ibi*, which means *in another place*. According to the USLegal online legal dictionary [5], *"It is an excuse supplied by a person suspected of or charged with a crime, supposedly explaining why they could not be guilty. [...] An alibi generally involves a claim that the accused was involved in another activity at the time of the crime."*. In this work an alibi based on digital evidence is referred to as *digital alibi*. There are several examples of trials in which digital evidence played a fundamental role to argue the acquittal of the accused.

Rodney Bradford: An interesting case is that involving Rodney Bradford, a 19 years old resident of New York, arrested in October 2009 on suspicion of armed robbery [6]–[8]. His defense lawyer claimed the innocence of Mr. Bradford asserting that he was at his father's house at the time of the crime. The evidence offered in support of this thesis was a message posted by the suspected on Facebook with the timestamp ''October 17 - 11:49 AM'', exactly one minute before the robbery. The status update would take place from the personal computer of his father. The subsequent investigation confirmed that the connection was established from an apartment located at more than thirteen miles from the scene of the crime. Rodney Bradford was released 12 days after his arrest. This is the first case in which a status update on Facebook has been used as an alibi. Although an accomplice could have acted on behalf of Rodney Bradford to construct his alibi, the Court rejected such a possibility, since it would have implied a level of criminal genius unusual in such a young individual.

Alberto Stasi: Another example, extremely interesting in terms of assessing a digital alibi, is the Italian case of Garlasco [9], [10]. The trial of first instance ended with the acquittal of Alberto Stasi, the main suspect in the murder of his girlfriend Chiara Poggi. The defendant proclaimed his innocence by claiming that he was working at his computer at the time of the crime. This trial has been characterized by a close comparison between the results of the analysis performed on each type of specimen, such as DNA traces and digital evidence on the PCs of both the victim and the accused. These findings were complemented by traditional forensic techniques such as interrogations. However, the attention of the investigators mainly focused on verifying if the digital alibi claimed by Stasi was true or false. Since no overwhelming evidence proving the contrary was found, the Court accepted the alibi of Stasi and directed his acquittal. The appeal, concluded in December 2011, confirmed the innocence of Alberto Stasi [11]. The appeal to the Cassation Court, opened in April 2012, is currently in progress [12].

Other Cases: Not all claims of digital alibi end up being accepted by Courts. It is worth mentioning the case involving Everett Eugene Russell [13], a Texan man accused by his ex-wife of the violation of a protective order requiring the ex-husband to stay away from her residence. Part of the evidence supporting the alibi claimed by the defendant consisted of a disk containing a set of cookies, which would have proven that Mr. Russell was surfing the Web at the time of the crime. It was not sufficient, since the Prosecutors argued that the cookie files could have been tampered, and the Jury agreed. Mr. Russell was also convicted by the Texas appeals Court.

Another example is the case involving Douglas Plude [14], accused of the murder of his wife Genell. Mr. Plude claimed that she committed suicide, and the proof would have been found on her personal computer. The digital forensic analysis revealed that some online searches were performed from the PC about Fioricet, the substance ingested by Genell which hypothetically conducted her to death. However, the majority concluded that ''Whoever performed the search on Genell's computer only examined the first page of various results—no page with dosing information was ever displayed on the computer.'' Also in this case, digital evidence did not provide the expected alibi to the accused.

B. FALSE DIGITAL ALIBI

In light of what has been discussed so far, it is worth questioning whether it is possible to artificially produce a predetermined set of digital evidence in order to forge a digital alibi. While it may be quite easy to identify the owner of a device, it is not so easy to determine ''who'' (human) or ''what'' (software) was acting on the system at a specific time. It is due to the ubiquitous and immaterial nature of digital information. Consequently, digital evidence should always be considered as circumstantial evidence, and should be always integrated with evidence obtained by means of other forensic techniques (fingerprints, DNA analysis, interrogations and so on).

Any attempt of tampering or producing digital information in order to construct an alibi has been referred to as *false digital alibi* [15]. This work shows that an individual may predispose a common personal computer to perform a predefined sequence of automatic actions at a specific time, which generate, in turn, a set of digital evidence that can be exploited in a trial to claim a digital alibi. The traces produced by means of this technique resulted to be indistinguishable, upon a post-mortem digital forensic analysis, from those produced by a real user performing the same activities. Besides, no particular skills are needed to implement the presented methodology, since it makes use of existing and easy-to-use tools. This result is an advise for Judges, Juries and Attorneys involved in legal proceedings where reliability of digital evidence could have an high impact on the verdict. This work also provides some insights to be followed in order to consider possible fake evidence.

The remainder of this paper is organized as follows. In Section II different methods to artificially produce digital evidence on a computer are examined. In Section III the automation methodology is presented, while in Section IV the process aimed to develop and deploy an automation is discussed. In Section V a number of automation tools are analyzed and in Section VI a real case study on Windows 7 is presented. Finally, in Section VII the conclusions are drawn.

II. CREATION OF PREDETERMINED DIGITAL EVIDENCE

In this work the following scenario is supposed. There is an individual interested in constructing a false digital alibi, called Alibi Maker (AM), who can leverage his personal computer, called Target System (TS), in order to accomplish this task. After the digital alibi is forged, the TS is subject to a post-mortem analysis performed by a group of Digital Forensic Analysts (DFA), in charge of extracting any useful information supporting, or invalidating, the digital alibi. This paper focuses on the specific case where the TS is a common personal computer. One of the main objective of this work is to prove that also a non-skilled individual would be able to construct a false digital alibi.

Keeping in mind the considerations made in Section I, it is clear that an AM should aim to produce remote digital evidence, or even a mix of local and remote evidence. The main difficulty is that, while timestamps of data stored on a local device can be easily tampered with, remote information cannot be modified once stored on the server (excluding the case of unauthorized access). The only possibility is that the actions aimed to produce remote digital evidence are ''really'' performed during the alibi timeline. There are some different strategies to accomplish this task.

The trivial solution is to engage an accomplice in charge of using the TS to produce local and remote digital traces on behalf of the AM. However, the involvement of another person could be risky. First, it requires physical contact of the accomplice with the device and the environment where it is located (e.g., AM's home), which may produce biological traces (fingerprints, DNA, etc.). Second, the accomplice could be spotted by other persons, who may witness the fact during the trial. Third, the accomplice itself could yield to the pressure and confess his involvement.

Actually, there are two further approaches that allow an AM to forge reliable digital evidence without any accomplice: Remotization and Automation.

The *remotization* approach is based on the remote control of the TS from a different machine. It can be accomplished, for example, by means of a remote control software or a KVM device. On the contrary, the *automation* approach is based on the use of a software able to perform a series of automated actions on behalf of the AM. Pros and cons of both approaches are discussed below.

A. REMOTIZATION

There are various ways to remotely control a computer. In any case, master and slave should be connected by means of a communication channel in order to send and receive commands. In this section two possible techniques are considered, followed by some considerations about risks and applicability.

A simple solution consists of the use of a *remote control software* to pilot the TS from another (presumably far) computer. Since the presence of a server application allowing remote administration may be considered suspicious by the DFA, the AM should avoid installation of this kind of application on the TS. To accomplish it, he might use a portable application such as TeamViewer Portable [16], Portable RealVNC Server [17], GoTo VNC Server Java Applet [18], or even a backdoor trojan-horse such as ProRat [19], Bandook [20], etc.. While all the mentioned remote control software is freely available and easy-to-use, the success of these techniques strongly depends on the ability of obfuscating the server process on the TS. As discussed in the remainder of this article, it might require a deeper knowledge of the underlying operating system.

An alternative solution is the use of a keyboard, video, mouse (KVM) switch over IP (IP-KVM). An IP-KVM is a device that allows the remote connection to the keyboard, video and mouse ports of the TS. It captures, digitizes and compresses the video signal of the TS before transmitting it to a remote controller by means of an IP connection. Conversely,

a console application on the controlling computer captures mouse and keyboard inputs and sends them to the IP-KVM device, which, in turn, generates respective mouse and keyboard inputs for the controlled system. The amount of suspicious traces that can be found upon a digital forensic analysis of the TS is limited, mostly considering that modern IP-KVM devices [21] do not require any software to be installed both on the controller and on the target systems. Obviously, the KVM device is itself evidence that should be destroyed or obfuscated by the AM in order to divert the investigation.

In both cases, it is worth highlighting that lot of suspicious traces, such as information about the controller machine, MAC or IP addresses of the KVM, may be recorded by other components of the network, such as DHCP, NAT and DNS servers. An analysis of caches and logs of these systems may reveal with no doubt an anomalous connection lasting for the entire alibi timeline. Although the AM has the possibility of being very far away from the potential alibi location, the main limitation of the Remotization approach is the necessity of human intervention.

B. AUTOMATION

A typical anti-forensic activity aimed at constructing a false digital alibi is the tampering of timing information of files and resources. It can be accomplished, for example, by modifying the BIOS clock. The main limitation of this approach is that it can only produce local forged digital evidence, since it is assumed that the AM cannot modify resources stored by trusted-third-parties such as Facebook or Google. The methodology presented in this work shows that an AM can generate a false digital alibi based on reliable digital evidence without any human intervention. All the local and remote evidence may be ''really'' generated at the required time by a software running on the TS, which is referred to as *automation*. The implementation of an automation does not require advanced skills, since it can be accomplished by means of freely available and easy-to-use tools.

Potentially, an automation may be able to simulate any common user activity, such as Web navigation, authentication to protected websites, posting of messages, sending of emails, as well as creation and modification of local documents, and even playing with online videogames. The evidence generated by such actions may provide a person with an airtight alibi. The presented methodology also includes techniques to avoid or remove traces which may reveal the execution of the automation. If all the needed precautions are taken, there is no way to determine if the evidence found on a computer has been generated by either a common user activity or an automated procedure.

III. THE AUTOMATION METHODOLOGY

The remainder of this work focuses on the automation approach to produce fake digital evidence. The presented methodology is based on the use of common software tools and its implementation does not require advanced computer skills. Some real case studies are subsequently discussed, showing that a digital forensic analysis on the computer used to run the automation procedure cannot distinguish between digital evidence produced by a real user and digital evidence produced by an automation.

A framework providing methods to reproduce mouse and keyboard events is referred to as an *automation tool*. Such frameworks are typically used to perform systematic and repetitive actions, like application testing or system configuration. Other possible uses include data analysis, data munging, data extraction, data transformation and data integration [22]. Currently, automation tools are available for any platform.

This paper warns about a possible malicious use of automation tools. Basic operations provided by these frameworks, like mouse clicks, movements and keystrokes, could be exploited to build up more complex actions such as interaction with user interfaces and controls (text boxes, buttons, etc.). Finally, such actions could be combined in order to reproduce full user activities such as Web browsing, email sending, document writing and so on. The constructed automation may be launched at a given time.

Most automation tools provide powerful implementation frameworks, easy-to-use graphical interfaces, high-level scripting environments, and often does not require any programming competences nor specific computer skills to construct an effective automation. Because of their growing usefulness, many resources like tutorials, online communities, utilities, downloads and books on automation tools are currently available on the Web. Table 1 presents a non-exhaustive list of activities that can be implemented by means of an automation tool, which may turn out to be useful to construct a false digital alibi.

TABLE 1. Example of Activities That May Be Produced by an Automation.

Activity	Actions
Web surfing	Execution of a Web browser, interaction with windows and tabs, surfing and switching among different webpages. Use of authentication data such as username and password to access protected Web sites, filling of forms, upload and download of files. Such actions can be combined to access online social networks and popular Web sites like Picasa, Dropbox, Gmail, Facebook, Twitter, and so on.
Document writing	Creation and modification of textual document, electronic sheets and presentations by means of an office application suite. Customization of file metadata such as access time, modification time, and filename. Deletion of files and directories.
Multimedia processing	Download and visualization of pictures from the Web. Modification of images with an image editing application. Adjustment of audio controls, reproduction, and/or record of audio and video files.
Videogame playing	Execution of, and interaction with, standalone videogames and browser-based videogames. Access to multiplayer games and use of the game character to perform basic actions (acting like a videogame bot [23]).
Control of other devices	Sending of AT commands to a modem connected to the PC in order to make a call over the PSTN network. Activation of a Bluetooth connection with a mobile phone, access and modification of data like contacts, SMSs, photos, etc.. Sending of messages, initialization of voice calls over the telecommunication network.

A. DIGITAL EVIDENCE OF AN AUTOMATION

An automation is a program which consists of files (compiled executables, scripts, etc.) and may access resources on the TS (other files, libraries, peripherals, etc.). It is worth noting that any accesses to system resources produce a potential modification of the system state which may be revealed by a digital forensic analysis. Unfortunately for the AM, not all the traces left by the automation are suitable to prove his presence in a given place at a given time. Some of these traces may reveal the execution of the automation and expose the attempt of false alibi instead. Substantially, two categories of evidence can be distinguished: *wanted* and *unwanted* evidence.

Wanted Evidence: The term *wanted* refers to all the traces suitable to prove the presence of the user at a given place during the entire timeline of the alibi. The browser history, the creation time of a document, the access time of an MP3 file, as well as the timestamp of an email, the time of a post on an online social network may be exploited as wanted evidence.

Unwanted Evidence: On the other hand, the term *unwanted* refers to any information that enables the DFA to uncover the use of an automation. Data remanence of the automation on the system storage (automation files or metadata), presence of suspicious software (e.g., tools and libraries used to build the automation) falls into this category. Even an anomalous user behaviour (e.g., use of applications never used before, abnormal accesses to files) may be considered an unwanted evidence.

The construction of an automation should include methods to detect and handle unwanted evidence. The traces left by the automation strongly depends on the software environment where it is executed, with its own operating system, resources and services. An automation is a piece of software stored in one or more files and resources, which is executed as one or more processes on the current operating system. Of course, the AM has to be logged onto the system to start the procedure.

These considerations give an abstraction of the possible unwanted evidence that can be produced by the automation.

Filesystem Traces: The code of an automation may be contained in a binary executable, in a script file, or even in multiple files implementing different modules. Moreover, it may require additional dependencies to be installed on the system such as dynamic libraries. The recovery of such elements may lead back with no doubts to the use of an automation. It is worth highlighting that a filesystem assigns some metadata to each file stored on the media. It typically includes file name, size, type (regular file, directory, device, etc.), owner, creation time, last access time, last modification time, last metadata change time and so on. Such information is typically maintained in filesystem-specific structures stored on particular disk locations (e.g., the Allocation Table in FAT, the Master File Table in NTFS, the i-nodes in the *NIX filesystem). The metadata management algorithm may vary depending on the filesystem implementation. Generally, whenever a file is deleted, the respective metadata may persist on the disk for a long time. Whenever recovered, such information may reveal useful traces to the DFA.

Execution Traces: In any OSes the *process* is the basic execution unit [24]. Whenever a program is executed, different modules of the OS are in charge of creating the necessary data structures in memory, loading the code from the executable file and scheduling its execution. Even early OSes were equipped with kernels able to trace the process activity and record information such as executable name, process creation time, amount of CPU cycles and memory used. These records are generally referred to as *accounting data*. Support for process accounting is provided by most of *NIX kernels. Whether enabled, such feature may constitute a critical source of unwanted evidence. More generally, any logging mechanisms, not necessarily kernel modules, can produce unwanted evidence. Examples of logging services are the *syslogd* [25] on *NIX and the Event Log Service [26] on Windows. Recent OSes often make use of caching mechanisms to improve system response, which basically consist in recording ready-to-use information about programs and data (e.g., the Prefetcher feature [27] on Windows) or libraries (e.g., the *ldconfig* service [28] on Linux). In recent OSes, the memory management subsystem typically implements the Virtual Memory [24] feature, which allows to swap memory pages to and from the hard drive in order to free up sufficient physical memory to meet RAM allocation requirements. In the case in which memory pages belonging to the automation process are swapped from the physical memory, unwanted information might be stored onto the disk unbeknown to the user.

Logon Traces: For auditing and security purposes, logging services typically record login and logout operations of each user. Logon traces may include information like local login time, local logout time and address of the remote host that performed the operation. Even though containing no information related to the automation process, logon traces may be an important source of information for the DFA and therefore should be carefully managed by the AM. Traces like system accesses performed at unusual time, or even login operations followed by many hours of inactivity may be classified by the DFA as being part of an anomalous behaviour.

While automating the logout procedure is relatively simple (e.g., most of the OSes provide facilities to power-off the computer, like the `shutdown.exe` program on Windows and the `shutdown` command on *NIX), automating the user's login is more tricky. Some BIOSes provide the Automatic Power Up functionality, which may be leveraged to schedule the beginning of the automation procedure. A trivial solution to directly access the user's environment is to disable the login prompt. Alternatively, the automation could be set-up in order to be launched at boot time with system privileges, so that it would be able to insert the user's credentials to access the system. However, such a solution is not easy to implement. For sake of simplicity, in this work we assume that the AM can access the system in order to manually launch the automation procedure, which will sleep until the the beginning of the activity simulation.

B. UNWANTED EVIDENCE HANDLING

The handling of unwanted traces is the most delicate part of the automation methodology, since it might require a reasonable knowledge of the underlying operational environment. The AM should be aware of any OS modules, services and applications running on the system in order to identify all possible sources of unwanted evidence [29]–[31]. In general, it is possible to identify some basic principles in order to accomplish unwanted evidence handling. In the following, three complementary approaches are documented: *a-priori avoidance*, *a-posteriori removal* and *obfuscation*.

A-Priori Avoidance: The software environment hosting the automation may be prepared in order to generate as less unwanted evidence as possible. Such a-priori avoidance may require a certain knowledge of the underlying operating system, features and services running on the machine. A naive approach consists of disabling any logging mechanisms and other services that may record information about files and processes belonging to the automation. For example, features like Virtual Memory, Prefetch and Volume Shadow Copy could be disabled on Windows. Similarly, logging services such as *syslogd* can be disabled on Linux. However, the presence of *disabled* features that are typically enabled by default may be considered suspicious by the DFA. Alternatively, the AM could temporarily disable these services, but also this solution might produce suspicious traces, such as ''temporal holes'' in the system history.

In some cases the AM can avoid unwanted evidence by exploiting some tricks. For instance, on Windows, it is possible to launch the automation from the command prompt in order to avoid logging of the application path in the Registry. On *NIX systems, the AM can avoid logging of last executed commands by killing the terminal process. On both Windows and *NIX systems, enough RAM can avoid swapping of memory pages of the automation process to the disk. A system-independent trick to reduce unwanted evidence consists of executing the automation from an external device (e.g., an USB flash drive), which avoids the presence of anomalous files on the main filesystem. However, the AM should wipe or destroy the external memory before the intervention of the DFA.

A-Posteriori Removal: The a-posteriori removal is an alternative or a complementary approach to the a-priori avoidance and obfuscation. It is based on the removal of unwanted traces by means of a *secure deletion* procedure. According to the definition given in [32], the secure deletion of an information is a task that involves the removal of any traces of this information from the system, as well as any evidence left by the deletion procedure itself. The core of a secure deletion procedure is the wiping function, which should guarantee complete removal of the selected data from the system. Typically, the common file deletion procedure provided by the OS does not meet this requirement. In fact, this operation is typically implemented by means of the `unlink` [33] *system call*, which only marks the blocks occupied by the file as free space. This implies that file data may persist on the disk until it is overwritten. The amount of rewritings required to completely remove any traces of certain data from an electromagnetic disk has been a controversial theme [34]–[36]. However, *Wright et al.* [37] demonstrated that even a single low-level overwrite of the disk blocks containing a certain data is sufficient to guarantee its unrecoverability, as previously stated in [38].

Obfuscation: It is very difficult for an average user to both avoid and remove evidence like Windows Registry keys, system log entries, filesystem metadata and so on. Typically, such resources are encoded or protected, and cannot be accessed at system runtime. Modifying this information may require a certain level of expertise with the underlying OS. In some cases, the simplest solution for the AM is to obfuscate the automation traces by making them appear like produced by common application or system activities. It can be accomplished by adopting simple shrewdness such as using common filenames, storing the suspicious files in system folders, mixing automation files with non-suspicious padding files (images, videos), etc..

In general, the more the AM is able to avoid unwanted evidence, the less the probability is to make mistakes in handling them. However, as mentioned before, the AM should be careful in disabling system features, since it could be considered, in turn, a suspicion. The better solution to divert a forensic analysis is to make all the evidence appear as ''usual''. An optimal trade-off between system alteration and unwanted evidence removal can be reached by combining all three approaches presented before.

The a-posteriori removal technique is meant to remove all the residual unwanted evidence of an automation. There are basically two approaches to accomplish this task: *manual* and *automatic*.

Manual Deletion: In case the AM has physical access to the TS before the DFA intervention, he can manually delete the unwanted traces produced by the automation. In particular, he could wipe any files belonging to the automation, remove any suspicious entries in application logs and clean-up system records. The AM could use OS-specific tools to accomplish this task. For example, the shred tool [39] on *NIX systems and the SDelete software [40] on Windows systems. However, this approach has two disadvantages. First, some protected resources such as filesystem structures cannot be accessed at system runtime. Second, the use of wiping tools may recursively determine the generation of other unwanted evidence. A better approach is to indirectly access the system storage, for example, by means of a live Linux distribution such as the Deft [41] suite. In this way, the AM can access and modify system protected resources. Since the entire software environment used to accomplish this task is maintained in memory, with the due precautions, this approach produces no unwanted traces on the TS.

Automatic Deletion: The system clean-up procedure could be part of the automation itself. After producing the

digital alibi, the script should be able to *automatically* locate all the unwanted evidence. It can be accomplished in a static way (i.e., hardcoding file paths into the automation), or even in a dynamic manner (i.e., searching for all the occurrences of a particular string in both allocated and unallocated space of the hard disk). Finally, a secure deletion module has to be invoked to complete the work. A secure deletion procedure should not be limited to the removal of a specific information, but it should also remove any evidence of its presence from the system. In other words, it should be also able to perform a *self-deletion*. This is not a simple task, since executable files are typically locked by the operating system loader. This is done to preserve the read-only property of the text segment of the executable code.

This issue can be solved by exploiting some characteristics of the interpreted programming languages [32]. Typically, an interpreted program does not use native machine code, but an intermediate language instead (e.g., Java bytecode, Python script, etc.), which is indirectly executed by means of an interpreter. Despite the interpreter being a binary executable, and therefore locked by the loader, most of interpreters does not apply the lock in scripts in execution. Under such conditions, the interpreted program can perform self-modification and, as a consequence, self-deletion.

The main advantage of the manual method for a-posteriori deletion is its simplicity. It does not require particular skills, but only a little knowledge of the operational environment. However, it requires physical access to the TS before its seizure. On the contrary, the automatic method does not require a further intervention of the AM. Nevertheless, it has two disadvantages. First, the automatic clean-up process runs on the same operational environment of the automation, which might produce unwanted evidence, in turn. Second, it requires technical expertise for the implementation of the self-deletion procedure.

IV. DEVELOPMENT OF AN AUTOMATION

The development of an automation can be divided into five phases: (1) preparation of the development environment; (2) implementation of the automation; (3) testing of the automation procedure; (4) exportation of the automation; (5) destruction of the development environment. Clearly, the steps (1) and (5) are strongly related and therefore are discussed together.

A. PREPARATION AND DESTRUCTION OF THE ENVIRONMENT

Not only the execution of an automation may produce unwanted evidence, but also its development process. For example, traces of software, libraries, files used to implement the automation may remain on the TS in the form of temporary data, unlinked blocks on the disk, filesystem metadata, operating system records, etc.. In general, the development environment should meet some specific requirements: (1) it should be totally *isolated* from the TS, so that no evidence of the development process is left on the TS; (2) it should be *temporary*, since it is, in its complexity, an unwanted evidence to be destroyed; (3) it should be as *similar* to the TS as possible, since the automation implemented/tested on this environment should be exported and executed on the TS without a hitch. There are many techniques to create a proper development environment:

- **Virtual machine:** A virtual machine running the same OS of the TS can be executed within the TS itself. In this case the AM can exploit the isolation of the virtual machine, which guarantees that any actions performed on the guest system do not affect the host system. The only issue of this approach regards the destruction of the development environment, which can be reduced to the secure deletion of the file(s) storing the virtual drive.
- **Live OS:** A live version of the target system may be used as a development environment. Currently, live versions of all the most used operating systems are available on the Web. The main advantage of this approach is that the AM needs not worry about destroying the development environment, since a careful use of the live OS may prevent any accesses to the disk. However, it has some disadvantages such as limited resources and data volatility upon reboots.
- **Physically isolated system:** The automation can be implemented/tested on a different computer. In this case, the AM has complete freedom about system configuration and automation testing. Clearly, the AM should get rid of this device after the development is completed.

B. IMPLEMENTATION OF THE AUTOMATION

The implementation of an automation strongly depends on the choice of the automation techniques and tools. It may be accomplished by using easy-to-use frameworks such as AutoIt [42], or by writing hundreds code lines in a whatever scripting language, or even by combining both techniques. Generally, the implementation of an automation is strictly related to the operational environment. Most of automation tools and APIs suitable to simulate user interactions are based on screen coordinates. Therefore, different screen resolution or even different position of GUI elements on the screen may cause a malfunction of the automation. Since the automation aims to simulate a real human behaviour, all the automated operations should be carefully synchronized (e.g., writing into a document only after it has been loaded). However, different operational environments may have a different response time to the same input (e.g., due to different system load), which may undermine the correct synchronization of the operations. In order to tackle these issues, the AM has to be able to set-up a development system which is as similar as possible to the TS.

C. TESTING OF THE AUTOMATION PROCEDURE

The testing phase has a twofold objective: (1) verify that the automation acts correctly, so that it fulfills all and only the expected operations; (2) identify all the unwanted artifacts left by the automation. The first task can be accomplished by executing the automation procedure several times, even in different system conditions (high CPU or memory load, low network bandwidth, etc.). The AM should always ensure that all the wanted evidence are correctly produced (documents written, browser history updated, email sent, etc.). The second task is more complex since it includes the identification of all the resources accessed and modified by the automation process, after which the AM should be able to discern whether such system changes may disclose unwanted information about the automation procedure. Some specific tools can turn out to be useful to accomplish this task:

- **Process monitoring tools:** It is extremely useful to examine the automation at runtime in order to detect all changes it makes to the system. It may be accomplished by means of a process monitoring tool such as Process Monitor [43] on Windows. Such application allows to monitor real-time filesystem accesses, Registry changes and process/thread activities. On *NIX environment, `lsof` [44] may reveal useful information about files, pipes, network sockets and devices accessed by the automation.
- **Digital forensic tools:** The AM may simulate the activity of the DFA by performing a self post-mortem analysis of the development system. There are many *NIX forensic distributions freely available on the Web (e.g., DEFT [41], CAINE [45], etc.), which contain lots of professional computer forensics software to accomplish this task.

D. EXPORTING THE AUTOMATION

Before executing the automation, all the necessary files should be exported from the development system to the TS. There are a number of viable strategies, with each one producing different kinds of unwanted evidence. A couple of possible approaches are described below.

- **Network Transfer:** All the needed files can be sent to the TS through a shared folder on the LAN, can be attached to an email, or can be uploaded on a free hosting server. Subsequently, the automation can be downloaded on the TS in order to be executed. After the download, any suspicious file should be securely removed from the hosting server. This last action could be problematic as the AM typically does not have full access to the remote machine. It can be resolved, for example, by uploading the automation files in an encrypted archive, which should prevent any recovery attempts.
- **External Memory Transfer:** In case the automation is loaded on the TS, all the suspicious files should be securely deleted after its execution. Alternatively, the automation could be executed from an external support, such as a CD-ROM, an USB flash drive or a SecureDigital card (SD card from now on). The execution of the automation does not require to copy any unwanted files on the TS. In case the AM cannot physically destroy the external memory before the DFA intervention, an automatic secure deletion procedure should be implemented. However, the device can be equipped with a non-transactional filesystem, like FAT, in order to reduce the risk of data remanence. Using some implementation tricks (see Section VI-B), the external storage can be removed immediately after the launch of the automation, with it completely avoiding the requirement of a secure deletion procedure.

E. ADDITIONAL CAUTIONS

Considering that the list of sequenced eubacterial genomes [46] contains all the bacteria known to have publicly available complete genome sequences, it is possible to recognize who have used a computer by analyzing the bacteria left by its fingertips on the keyboard and mouse [47]. The imprints left by the bacteria may persist for more than two weeks. Potentially, this is a new tool for forensic investigations. Of course, this kind of analysis can be also exploited by the AM to enforce his digital alibi. If the suspected is the only one to have used the computer, the defendant can request a forensic analysis which may confirm that the bacterial traces on the keyboard and mouse are those of the suspect.

People have their habits and then follow a predictable pattern. For example, it may be usual for the AM to establish an Internet connection during the morning, access a mailbox, browse some websites and work on some documents. For an airtight alibi, the behaviour of the AM inferred by the DFA from the TS should be very similar to his typical behaviour. Here, the possibility to use digital profiling techniques [48]–[50] could be very useful to the DFA in order to deal with difficult cases.

The post-mortem analysis of a computer may reveal a large number of digital evidence due to data remanence on the storage media. There are some forensic tools, such as log2timeline [51], which can be used to extract time-referenced information from a disk at different abstraction levels (filesystem, application, etc.). Such information may be used to reconstruct a timeline of the actions performed by the user. These results could be used, in turn, to extract the user's behavioral pattern, i.e., the correlation of the activities performed by the user over time. It could be accomplished, for example, by means of machine learning techniques. In order to avoid anomalies, the AM should always verify that the traces produced on the system fit with its usual behavior.

V. AUTOMATION TOOLS

An *automation tool* has been referred to as a framework that allows the implementation of a program able to simulate user activities. There are a number of easy-to-use applications which can be leveraged to accomplish this task. In general, any programming languages providing support for

simulation of GUI events are potential automation tools. Typically, a programming language allows a finer development with respect to a ready-to-use application, but it requires a technical expertise. A series of experiments has been conducted on different operating environments by using different automation tools. They turned out to differ in ease of use, effectiveness, portability and unwanted evidence generation. In effect, there is no best solution: the choice of an automation tool strictly depends on the environment, on the user skills and on the complexity of the alibi to be constructed.

A. THE SIMPLEST APPROACH: EXISTING SOFTWARE

There is a variety of existing software able to record and reply user actions, mostly intended for GUI testing. One of the most used on Windows environments is AutoIt [42]. It provides both a tool for recording user actions and a fully-fledged scripting environment to refine the script produced by the recording tool. An useful characteristic is that an AutoIt script can be compiled into a standalone executable, which does not require the presence of the AutoIt interpreter on the TS to be executed. A valid alternative to AutoIt is AutoHotkey [52], an open-source utility which basically provides the same features of AutoIt.

For Linux environments, it is worth mentioning GNU Xnee [53], which allows to record and replay user events under the X11 environment [54]. It can be invoked by both command line and GUI. An alternative is Xautomation, a suite of command line programs which enables to interact with most of objects on the screen and to simulate basic user interactions under the X11 environment. Another valid choice is `xdotool` [55], an application providing advanced features to interact with the X11 environment. In addition to Xautomation, it allows to search, focus, move among windows and virtual desktops, waiting for loading of application interfaces and so on. The xdotool-gui application provides an easy-to-use visual interface to xdotool. A key feature is that it is implemented by just two files, the xdotool itself and the libxdo.so.2 library.

The Apple Mac OS also offers a number of automation tools. An example is Automator [56], which provides an easy-to-use interface where the user can construct an automation by drag-and-dropping a series of predefined actions. An advantage is that Automator comes pre-installed with Mac OS X, thus not requiring installation of third-party software. However, it is not suitable to simulate complex user actions such as Web browsing. In addition, it requires access to several system resources, with it producing many unwanted traces.

B. ADVANCED APPROACHES

The use of an existing automation software may turn out to be restrictive whether advanced simulation features are required. In these cases, an AM with some expertise could write a fully customized automation by using a programming language. The choice of an interpreted language should be preferred as it simplifies the implementation of some features like the self-deletion [32].

VBScript (see Section VI-B) is a scripting language supported by default on any recent Microsoft OSes. It provides some basic procedures to simulate user interactions, such as mouse movements, clicks and keystrokes. The main advantage of using VBScript to implement an automation would be that it does not require any third-party resources.

In Mac OS, the AppleScript [57] language provides an useful framework to build automations. It is supported by default on Mac OS and does not require external modules to be installed. AppleScript files can be also compiled into standalone executables. In substance, it can be compared to VBScript in characteristics and functionalities.

Advanced programming languages often provide support for the simulation of user actions. An example is the Robot [58], [59] package included in the Java runtime environment. Although providing a fine-grained control of the system, building an automation by means of a programming language requires high expertise in code writing and can result in long and complex code to be carefully tested.

A hybrid approach would provide the better trade-off between potentiality and ease of implementation. A hybrid automation consists of two or more modules, each implementing a specific feature. In this way, the better automation tool can be chosen to accomplish a particular task. A basic hybrid automation could be composed by a *launcher* and a *simulator*. The launcher would be a program written in a certain programming language, in charge of accessing low-level system features and managing the event timeline. The simulator would be in charge of simulating user actions and producing wanted evidence. It could be implemented by means of a high-level automation tool such as xdotool or AutoIt.

In the presented experiment the automation is exported through an SD card, connected to the target system by means of an USB adapter. As previously stated in IV-D, this approach allows to avoid suspicious files on the TS, without the need to adopt a secure deletion technique to clean-up the system. The only traces that might remain on the system are information about the USB adapter, such as the Device ID and the Vendor ID, and the name of the automation script in (unreferenced) Registry entries. It is assumed that the AM has access to the SD card before the DFA intervention, so that its content can be replaced with non-suspicious files or destroyed.

VI. CASE STUDY: WINDOWS 7

This case study presents an implementation of the automation methodology. The user activities simulated by the automation are summarized in Table 2. The automation can be set in order to start the simulation at a given time, and the overall duration of the alibi is spread on a timeline lasting about 30 minutes.

Creating an advanced automation for Windows 7 is a delicate task, because of the large amount of unwanted evidence that should be taken into account. The OS implements lot

TABLE 2. Alibi Timeline.

Time	Activity
t_0	Execution of a Web browser.
t_1	Access to Facebook.
t_2	Posting of a message on Facebook.
t_3	Execution of a word processor.
t_4	Use of the word processor to write a document.
t_5	Use of the browser to access GMail.
t_6	Sending of an email.
t_7	System shutdown.

of features and services which perform intensive logging of information about executed programs and accessed data. Moreover, it is worth noting that Windows 7 uses the NTFS filesystem [60], which implements conservative policies for space allocation. In practice, each time a file is modified, the filesystem allocates new blocks, with it leaving unallocated blocks on the disk containing previously deleted data. The filesystem *journal* could also be a source of data remanence. All the techniques and tools presented in this section are also suitable to construct an automation on older Microsoft OS versions such as Windows XP. Besides, the unwanted traces to be managed on Windows 7 are a superset of those to be managed on past versions, due to the increasing number of features.

A. UNWANTED EVIDENCE IN WINDOWS 7

As discussed in the previous sections, an automation can leave a number of unwanted traces, such as the presence of suspicious files on the filesystem. In addition to that, lots of OS components rely on massive recording of information about used applications and accessed files. Such characteristics could determine the presence of unwanted evidence of the automation being recovered in the post-mortem analysis by the DFA. In this section, some possible solutions to avoid as much unwanted traces as possible are discussed. The main sources of unwanted evidence on Windows 7 are discussed below.

Prefetch/Superfetch: The *Prefetcher* module aims to speed-up the launch of applications executed at system start-up by pre-loading information into the memory. Portions of code and data used by these programs are recorded in specific files stored in a caching location on the filesystem (i.e., the `C:\Windows\Prefetch\` folder). The *Superfetch* module, active by default on Windows 7, enhances this feature by extending it to any application on the system. Basically, frequency of use of programs and files is monitored and recorded in a sort of history files stored in the prefetch folder. Lots of information (such as a list of recently accessed files) can be extracted from these logs by means of the ReWolf's tool [61]. The Superfetch module aims to pre-load applications and data accessed more frequently. If an automation is executed on a Windows 7 system, some traces about its use may be logged by the Superfetcher. However, unless the automation is executed lot of times, no meaningful information about code and data should be recorded due to the prefetching feature. Although the Superfetch service can be disabled [62], a better method to avoid unwanted traces in the logs is obfuscation (i.e., the use of non-suspicious filenames).

Pagefile: The virtual memory technique allows to the existing processes of an OS to use an overall amount of memory that exceeds the available RAM. The OS can move pages from the virtual address space of a process to the disk in order to make that memory free for other uses. In the Windows systems, the swapped pages can be stored in one or more files (`pagefile*.sys`) in the root of a partition. On the modern PCs the use of memory rarely exceed the available RAM as it is sufficient to address all the common system activities. Therefore, it is common among Windows users to disable such feature in order to gain more disk space. Although it is very unlikely that pages of an automation are swapped onto the disk (unless some heavy processes are launched during its execution), disabling the virtual memory could be a better solution to prevent eventual clues.

Restore Points: System Restore is a component of Windows 7 that periodically backups critical system data (Registry, system files, local user profile, etc.) in order to allow roll-back to a previous state of the system in case of malfunction or failure [63]. The System Restore service can be manually configured by the user and also automatically triggered by specific system events.

The creation of restore points can be predicted and thus avoided. In fact, it typically only occurs when an application is installed by means of an installer compliant with the System Restore, when Windows Update installs new updates, or when no other restore points have been created in the last seven days.

Hibernation: The *hibernation* technique allows to power-off a computer without losing its state, which can be resumed at the next power-on. It is implemented by dumping the content of the RAM onto the disk. In Windows 7 a memory dump is saved into the file `C:\hiberfile.sys`, which can be straightforwardly converted into a readable format in order to be analyzed. Typically, the hibernation module is enabled by default on laptops and can be automatically triggered upon long time of system inactivity. In order to avoid dump of information about the automation into the hibernation file, it is preferable to keep such service disabled.

Registry: The Windows Registry is a database aimed to store system settings as well as application and user settings. It contains various kinds of information about the OS, device drivers, services, user credentials, applications and so on [64]. The Registry is organized in a hierarchical structure whose content is stored on multiple files. Typically, global system settings are stored in the `C:\windows\system32\config` folder, while user-specific information is stored within its home directory (in `NTUSER.DAT` and `USRCLASS.DAT`). Since the integrity of the Registry is fundamental for the proper functioning of the system, the OS frequently backups these files. Registry backups can be found on different locations of the filesystem, such

as in the `C:\windows\system32\config\RegBack` folder or within the restore points created by the System Restore service. In the context of a computer forensic analysis, the Registry is the larger source of evidence on a Windows-based system. By analyzing the Registry, it is possible to reconstruct any ordinary user actions such as hardware plugged-in to the computer, executed software, network activities and opened documents.

The secure deletion of a Registry key in order to hide some information is very challenging. The removal of a Registry key is typically implemented by marking the respective cell as *deallocated*, but its content may persist on the disk until it gets overwritten. It is definitely hard for an AM to avoid any information about the automation being recorded in the Registry. For this reason, it could be preferable to adopt a different strategy such as obfuscation.

An example of system feature making extensive use of the Registry is the UserAssist feature. It is used since Windows XP to populate the user's *Start Menu* with the most frequently used applications. This is accomplished by maintaining application names and relative frequency counters in a specific Registry key of the user's hive (within the `NTUSER.DAT` file):

```
HKEY_CURRENT_USER\Software\Microsoft\Windows\
    CurrentVersion\Explorer\UserAssist
```

Logging of information in the UserAssist key can be avoided by running the automation from the command line.

B. IMPLEMENTATION

This section presents the implementation of the VBScript automation used for this case study. VBScript is a scripting language developed by Microsoft and modeled on Visual Basic. The interpreter for standalone scripts is provided by the Windows Script Host (WSH) environment, installed by default on Microsoft OSes since Windows 98.

The use of VBScript to implement an automation has a number of advantages. First of all, no third-party automation tools are required, since any required resources are installed by default on Windows 7. The VBScript interpreter typically loads the entire script into the memory before its execution and does not lock the relative file. This characteristic allows to physically remove the support containing the automation immediately after the procedure is started. In addition, VBScript can use the Component Object Model (COM) to interact with the system. In particular, the `Wscript.Shell` COM is used. It provides the basic mechanisms of interaction with the operating system. More in details, the `Run` method is used to execute external commands, the `AppActivate` method to change the focus of a running application, the `Sleep` method to pause the script execution and the `SendKeys` methods to send keystrokes to the currently active window. The automation has been coded into a script named `HexToDec.vb`, which performs all the activities summarized in Table 2. The script has been loaded onto a SD card containing other multimedia files—referred to as *padding files*—like videos and images. Once launched, the WSH interpreter loads the entire automation code into the memory, so that the SD card can be safely removed from the TS without interrupting the execution of the automation. In this scenario the transfer device (i.e., the SD card) can be physically destroyed before the DFA intervention.

C. EXECUTION

Once the SD card storing the automation is plugged into the computer, the system automatically mounts the removable device and assigns a drive letter to it (e.g., `E:`). The SD content is accessed by means of the *File Explorer* and the script `HexToDec.vb` is launched with a simple double-click. The automation delays the execution of the first action of the alibi to the time t_0. The starting time is hardcoded into the script by means of the `Sleep(milliseconds)` method. The automation simulates the use of a Web browser and some other applications present by default on the system. The launch of an application is performed by leveraging the search functionality of the *Start Menu*. The actions simulated to accomplish this task have been: (1) the pressure of the "Windows" key in order to open the *Start Menu* and get the focus of the *Search* text box; (2) the typing of the application name in order to get the application link; (3) the pressure of the "Enter" key in order to execute the selected application.

The following code excerpt shows the implementation of this technique. `Type` is a custom function used to simulate random delays between each keystroke (see Listing 4). Some details have been simplified for the sake of clarity.

Listing 1: Launch AppName by means of the Search functionality

```
1 funct RunByStartMenu( AppName )
2   Type( "{ALT}" + "{ESC}" )
3   Sleep( 3546 + getRand(5000) )
4   Type( AppName )
5   Sleep( 4635 + getRand(5000) )
6   Type("{ENTER}")
```

It is worth noting that this function does not produce unwanted evidence. In fact, the produced evidence are exactly those that would be produced whether the application is executed by the real user. The use of the application can be confirmed by the analysis of the following Registry key, which contains the list of the recently executed applications and from the UserAssist key, as mentioned before:

```
HKEY_CURRENT_USER\Software\Classes\Local Settings\
    Software\Microsoft\Windows\Shell\MuiCache
```

1) Execution of a Web Browser

One of the main activities of the automation is the use of a Web browser. Although any Web browser supporting keyboard shortcuts can be used to accomplish this task, in this case Internet Explorer 8 has been chosen as it is installed by

default on Windows 7. It is important to note that Internet Explorer is launched at time t_0 and closed at the end of the timeline. The automation uses the Web browser at times t_1 to access the Facebook website, at time t_2 to post a message, at time t_5 to access the GMail website and at time t_6 to send an email (see Tab. 2). The browser is left open for the entire alibi timeline in order to simulate the contemporary use of different applications. Internet Explorer is launched by means of the VBScript function defined in Listing 1. In order to avoid failures, it is crucial that the following precautions are taken:

- The Internet connection must be functioning and stable for the entire timeline. A connection interruption or an excessive loading time could interfere with the correct behaviour of the automation.
- Eventual certificates required for secure connections with the websites visited by the automation must be preventively installed. On the contrary, the browser might show a warning dialog that would remove the focus from the main window.
- It could be necessary to disable the automatic saving of login information in order to prevent errors when refilling authentication forms.
- The block of pop-ups should be disabled whether the automation uses pop-up frames to interact with some websites.
- All the websites accessed by the automation should be added to the "Trusted Sites" of Internet Explorer.

Listing 2: Posting of a message on Facebook

```
funct PostOnFacebook()
  RunByStartMenu("Internet Explorer")
  Type("{CTRL}" + "L")
  Type("http://www.facebook.com" + "{ENTER}")
  Type( User + "{TAB}" + Password + "{ENTER}")
  Type("{CTRL}" + "F" + "What's on your mind" + "{ESC}")
  Type(Message + "{TAB}" + "{ENTER}")
```

2) Use of Facebook

Taking a cue from the case of Rodney Bradford (see Section I-A), the false digital alibi of this case study includes the posting of a status message on Facebook. The procedure in charge to accomplish this task is started at time t_1. After the browser is launched, the automation simulates the `CTRL + L` keystroke in order to gain the focus of the address bar. Subsequently, the address `www.facebook.com` is typed in order to access the Facebook website. Once the page is loaded, the focus automatically passes to the login form. The required information is typed and submitted for authentication. Once the personal page of the user is loaded, a sequence of keystrokes (`CTRL + F`, `What's on your mind?`, `ESC`) is sent to the browser in order to gain the focus of the *Update Status* input box, then a message is typed and submitted to Facebook. The result is a new *Update Status* message with a timestamp compliant with the alibi timeline. It is important to highlight that all the actions performed by the automation are interleaved with appropriate time delays in order to allow the proper loading of the visited webpages. Moreover, each delay is also randomized in order to comply with the normal user behaviour. An excerpt of the VBScript code performing this task is shown in Listing 2. Some details, such as random delays, have been omitted for the sake of simplicity.

Since the automation code is strictly related to the page layout, this procedure may fail on unexpected changes of the Facebook website.

3) Use of a Word Processor

At time t_3 the automation executes the WordPad application in order to simulate a document writing. The text to be written is embedded in the automation script. Once the writing operation is completed, the *Save As* dialog is triggered by the `CTRL + S` keystroke, a name for the document is typed, then the `ENTER` keystroke is sent to the application in order to save such document. An excerpt of the code performing these operations is shown in Listing 3. As for the previous cases, some details have been omitted for the sake of clarity.

Listing 3: Use of WordPad for the creation of a textual document

```
funct CreateDocument()
  RunByStartMenu("Wordpad")
  Type(Content)
  Type("{CTRL}" + "S")
  Type(DocName + "{ENTER}")
  SendKeys ("{ALT}" + "{F4}")
```

A digital forensic analysis could reveal an anomaly if the creation date of the document is very close to the last modified date. Such information can be retrieved from the filesystem metadata. This issue is addressed by introducing a random delay between each keystroke by means of the `Type()` function. An excerpt of the `Type` function is shown in Listing 4. The variables `min_delay` and `max_delay` indicate, respectively, the minimum and maximum delay between each keystroke.

Listing 4: Function used to simulate the typing of a text

```
funct Type(str)
  for each c in str
    Sleep( min_delay + rnd()*max_delay )
    SendKeys( c )
```

4) Sending of an Email

The last activity performed by the automation is the use of the GMail service in order to send an email. At time t_5, after closing the WordPad window, the input focus passes to Internet Explorer. Similarly to the case of Facebook, the automation exploits the keyboard shortcut `CTRL + L` to acquire the focus of the location bar, types the URL of the GMail website and sleeps some seconds in order to allow its loading. At this point the script fills the login form and waits for the loading of the service. The page used for the composition of a new email is invoked by exploiting the

Find function of the browser (`CTRL + F`) and searching for the string `COMPOSE`. This trick enables to acquire the focus of the *COMPOSE* button. After inserting the recipient and the message, a `TAB` keystroke is sufficient to acquire the focus of the *SEND* button in order to send the email. As for the case of Facebook, the proper functioning of the automation strictly depends on the layout of the GMail website.

D. ANALYSIS

The analysis phase has a twofold objective: (1) verify that the digital evidence produced by the automation on the TS is coherent with the alibi timeline; (2) discover any unwanted evidence left by the automation. In order to accomplish these tasks a digital forensic analysis has been arranged according to the methodology presented in [65]. With respect to a real case, the DFA has full knowledge about methods, procedures and technologies adopted to construct the digital alibi. As a consequence, the analysis may be better targeted. The TS has been implemented as a Virtual Machine (VM) in order to speed-up and simplify the overall analysis procedure. Moreover, the use of a VM allows to create different snapshots of the system in order to analyze differences between the state before and the state after the execution of the automation procedure. The analysis mainly focused on the following aspects.

Operating System and Applications: All the system structures containing information about executed applications (e.g., Registry, Prefetch and Superfetch files, Pagefile, and so on) have been analyzed in order to verify whether the automation produced artifacts. Any evidence being part of the alibi (accesses to websites, creation of documents, etc.) has been also collected in order to reconstruct the alibi timeline.

Timeline: This analysis focused on the identification of all the files accessed or modified during the construction of the alibi in order to detect any relationship with the automation that generated them.

File Content: All the files modified during the alibi timeline have been analyzed in order to find any suspicious trace that may be linked to the execution of the automation.

Low-level search: A set of signatures of the automation has been used to perform a deep low-level scan of the entire hard disk (including allocated and unallocated space) in order to find any possible clue of the automation.

The above activities have been carried out by using the following digital forensic tools. RegRipper2 [66] has been used to analyze the Windows Registry. IECookiesView [67], IEHistoryView [68] and IECacheView [69] have been used to analyze the browser activities. AccessData Forensic Toolkit [70] has been used for the storage media analysis, and in particular the *DT Search engine* has been adopted to perform the low-level signature search. The Superfetch files have been analyzed by means of SuperFetch Dumper [61]. All the traces revealed by the analysis confirmed the alibi, while no clue about the automation was found.

VII. CONCLUSION

This paper presents a methodology to generate a set of digital evidence that could be exploited by a party in a trial in order to claim an alibi. With respect to common anti-forensic techniques, which are typically based on information tampering, our methodology relies on the construction of an automation. The automation is a program able to simulate a series of human activities on a computer at a given time. They include local operations such as document writing and music playing, as well as remote operations such as Web surfing and email sending. Using this approach, it is possible to construct a digital alibi involving trusted-third-parties such as ISPs and companies providing services via Internet. The problem of avoiding or deleting unwanted traces left by the automation is also addressed. Finally, a case study on a target system running Windows 7 is presented in order to show a real application and implementation of the proposed methodology. A computer forensic analysis of the target system has confirmed the alibi and has revealed no unwanted evidence about the presence and execution of the automation. Apart some differences, the same methodology can be extended to any digital device equipped with a modern operating system, such as Android smartphones [71], [72].

This work highlights the need of an evolution in approaching legal cases that involve digital evidence. Digital evidence are circumstantial evidence and should be always considered as part of a larger behavioural pattern, which requires to be reconstructed by means of traditional investigation techniques. To sum up, the plausibility of a digital alibi should be always verified *cum grano salis*.

ACKNOWLEDGMENT

The authors would like to thank friends from IISFA (International Information System Forensics Association) for their support, their valuable suggestions and useful discussions during the research phase. In particular to Gerardo Costabile (President of IISFA Italian Chapter), Francesco Cajani (Deputy Public Prosecutor High Tech Crime Unit Court of Law in Milano, Italy), Mattia Epifani and Litiano Piccin of IISFA Italian Chapter.

We would like to thank V.Q.A. Elvira D'Amato (Head of the Centre Against Child Pornography on the Internet, Postal and Communications Police, Italian Ministry of the Interior), Lieutenant Colonel Antonio Colella (cybercriminologist and Italian Army Officier) and Moti Yung (Google Inc. and Columbia University) for their worthy advices and support.

A special thank goes to Mario Ianulardo, Computer Crime Lawyer (Naples, Italy) for the endless and interesting discussions on the probative value of a false digital alibi.

REFERENCES

[1] Miniwatts Marketing Group. (2012, Jun.). *Internet World Stats*, Bogota, CA, USA [Online]. Available: http://www.internetworldstats.com/stats.htm

[2] W. Kruse and J. Heiser, *Computer Forensics: Incident Response Essentials*. Upper Saddle River, NJ, USA: Pearson Education, 2001.

[3] Symantec Corporation. (2013, Feb.). *Norton Cybercrime Report 2012*, Sunnyvale, CA, USA [Online]. Available: https://www.norton.com/2012cybercrimereport

[4] U.S. Legal, Inc. (2001–2013). *Digital Evidence Law & Legal Definition*, Flowood, MS, USA [Online]. Available: http://definitions.uslegal.com/d/digital-evidence/

[5] U.S. Legal, Inc. (2001–2013). *Legal Definitions and Legal Terms Dictionary*, Flowood, MS, USA [Online]. Available: http://definitions.uslegal.com/

[6] Msnbc News, Company. (2013, Mar.). *Facebook Message Frees NYC Robbery Suspect*, New York, NY, USA [Online]. Available: http: // www.msnbc.msn.com / id / 33883605 / ns/technology_and_science-tech_and_gadgets/

[7] The New York Times Company. (2013, Mar.). *I'm Innocent. Just Check My Status on Facebook*, New York, NY, USA [Online]. Available: http://www.nytimes.com/2009/11/12/nyregion/12facebook.html?_r=1

[8] V. Juarez. (2009, Nov.). *Facebook Status Update Provides Alibi—CNN Website* [Online]. Available: http://edition.cnn.com/2009/CRIME/11/12/facebook.alibi/index.html

[9] S. Vitelli. (2013, Feb.). *Sentenza Di Primo Grado Del Processo Stasi* [Online]. Available: http://static.repubblica.it/laprovinciapavese/pdf/SENTENZA_STASI.pdf

[10] F. Bravo. (2013, Jan.). *La Computer Forensics Nelle Motivazioni Della Sentenza Sull'Omicidio Di Garlasco* [Online]. Available: http://internetsociety.wordpress.com/2010/03/16/ la-computer-forensics-nelle-motivazioni-della-sentenza-sullomicidio-di-garlasco /

[11] Corte di Assise di Appello di Milano. (2013, Jan.). *Sentenza Di Appello Del Caso Garlasco*, Milan, Italy [Online]. Available: http://www.giuristiediritto.it/le-sentenze-dei-processi-penali-celebri/880-sentenza-appello-caso-garlasco-alberto-stasi.html

[12] Corriere della Sera, Redazione Online. (2013, Jan.). *Garlasco, L'accusa Ricorre in Cassazione: Illogica L'assoluzione di Alberto Stasi*, Milan, Italy [Online]. Available: http://www.corriere.it/cronache / 12_aprile_24 / garlasco-procura-ricorre-cassazione-assoluzione-alberto-stasi_b42e721c-8e18-11e1-839c-11a4cf6ed581.shtml

[13] D. McCullagh. (2013, Jan.). *Police Blotter: Web Cookies Become Defendant's Alibi* [Online]. Available: http://news.cnet.com/Police-blotter-Web-cookies-become-defendants-alibi / 2100-1047_3-6129993.html?tag=mncol;txt

[14] D. McCullagh. (2013, Jan.). *Police Blotter: Computer Logs as Alibi in Wife's Death* [Online]. Available: http://news.cnet.com/Police-blotter-Computer-logs-as-alibi-in-wifes-death/2100-1030_3-6167028.html

[15] A. De Santis, A. Castiglione, G. Cattaneo, G. De Maio, and M. Ianulardo, "Automated construction of a false digital alibi," in *MURPBES* (Lecture Notes in Computer Science), A. M. Tjoa, G. Quirchmayr, I. You, and L. Xu, Eds. New York, NY, USA: Springer-Verlag, 2011, pp. 359–373.

[16] TeamViewer GmbH. (2012, Aug.). *TeamViewer—Free Remote Control, Remote Access & Online Meetings*, Goppingen, Germany [Online]. Available: http://www.teamviewer.com/

[17] A. Weber. (2012, Aug.). *Portable RealVNC Server* [Online]. Available: http://www.andysblog.de/portable-realvnc-server

[18] GoTo Servers. (2013, Oct.). *VNC Server Java Applet (GSVNCJ)*, New York, NY, USA [Online]. Available: http://www.gotoservers.com/gsvncj.html

[19] PRO Group, Inc. (2013, Jan.). *ProRat*, Englewood, CO, USA [Online]. Available: http://www.prorat.net/

[20] Nuclear Winter Crew. (2013, Feb.). *Bandook*, New York, NY, USA [Online]. Available: http://www.nuclearwintercrew.com/

[21] Opengear, Inc. (2013, Mar.). *IP-KVM 1001*, Sandy, UT, USA [Online]. Available: http://www.opengear.com/product-IP-KVM.pdf

[22] T. Myer, *Apple Automator with AppleScript Bible*. New York, NY, USA: Wiley, 2009.

[23] (2013, Feb.). *Video Game Bot—Wikipedia, the Free Encyclopedia* [Online]. Available: https://en.wikipedia.org/wiki/Video_game_bot

[24] A. Silberschatz, P. B. Galvin, and G. Gagne, *Operating System Concepts*, 8th ed. New York, NY, USA: Wiley, 2008.

[25] (2012, Dec.). *Linux System Administration (8)—SYSKLOGD* [Online]. Available: http://www.linuxmanpages.com/man8/syslogd.8.php

[26] Forensics Wiki. (2013, Feb.). *Windows Event Log (EVT)*, Pittsburgh, PA, USA [Online]. Available: http://www.forensicswiki.org/wiki/Windows_Event_Log_(EVT)

[27] M. Russinovich, *Microsoft Windows internals: Microsoft Windows Server 2003, Windows XP, and Windows 2000*. San Francisco, CA, USA: Microsoft, 2005.

[28] Linux Man-Pages. (2012, Dec.). *Linux Programmer's Manual (8)—LDCONFIG*, San Francisco, CA, USA [Online]. Available: http://www.linuxmanpages.com/man8/ldconfig.8.php

[29] Microsoft. (2013, Jan.). *Resources and Tools for IT Professionals—TechNet*, San Francisco, CA, USA [Online]. Available: http://technet.microsoft.com/en-us/

[30] M. Kerrisk. (2013, Jan.). *Linux Man-Pages Online* [Online]. Available: http://man7.org/linux/man-pages/index.html

[31] Digital Forensics Community. (2013, Feb.). *Forensics Wiki*, Rockland, MA, USA [Online]. Available: http://www.forensicswiki.org/

[32] A. Castiglione, G. Cattaneo, G. De Maio, and A. De Santis, "Automatic, selective and secure deletion of digital evidence," in *Proc. Broadband, Wireless Comput., Commun. Appl., Int. Conf.*, Oct. 2011, pp. 392–398.

[33] M. Kerrisk. (2012, Sep.). *Unlink(2)—Linux Manual Page* [Online]. Available: http://man7.org/linux/man-pages/man2/unlink.2.html

[34] P. Gutmann, "Secure deletion of data from magnetic and solid-state memory," in *Proc. 6th Conf. USENIX Security Symp., Focusing Appl. Cryptography*, vol. 6. Jul. 1996, pp. 77–89.

[35] P. Gutmann, "Data remanence in semiconductor devices," in *Proc. 10th Conf. USENIX Security Symp.*, vol. 10. Aug. 2001, pp. 1–4.

[36] U.S. Department of Defense. (2010, Feb.). *DoD Directive 5220.22, National Industrial Security Program (NISP)*, New York, NY, USA [Online]. Available: http://www.dtic.mil/whs/directives/corres/html/522022m.htm

[37] C. Wright, D. Kleiman, and R. S. Sundhar, "Overwriting hard drive data: The great wiping controversy," in *ICISS* (Lecture Notes in Computer Science), R. Sekar and A. K. Pujari, Eds. New York, NY, USA: Springer-Verlag, 2008, pp. 243–257.

[38] (2006, Sep.). *NIST Special Publication 800-88: Guidelines for Media Sanitization* [Online]. Available: http://csrc.nist.gov/publications/nistpubs/

[39] P. Christias. (2012, Sep.). *UNIX Man Pages: Shred (1)* [Online]. Available: http://unixhelp.ed.ac.uk/CGI/man-cgi?shred+1

[40] M. Russinovich. (2013, Jan.). *SDelete* [Online]. Available: http://technet.microsoft.com/en-us/sysinternals/bb897443

[41] S. Padrepietro. (2013, Jan.). *DEFT Linux—Computer Forensics Live CD* [Online]. Available: http://www.deftlinux.net/

[42] J. Bennett. (2012, Jan.). *AutoIt v3.3.6.1* [Online]. Available: http://www.autoitscript.com/autoit3/

[43] M. Russinovich and B. Cogswell. (2012, Apr.). *Process Monitor* [Online]. Available: http://technet.microsoft.com/en-us/sysinternals/bb896645

[44] V. Abell. (1994). *LSOF(8)—List Open Files* [Online]. Available: http://unixhelp.ed.ac.uk/CGI/man-cgi?lsof

[45] N. Bassetti. (2012). *CAINE (Computer Aided INvestigative Environment)* [Online]. Available: http://www.caine-live.net

[46] Wikipedia. (2013, Apr.). *List of Sequenced Bacterial Genomes* [Online]. Available: http://en.wikipedia.org/wiki/List_of_sequenced_bacterial_genomes

[47] N. Fierer, C. L. Lauber, N. Zhou, D. McDonald, E. K. Costello, and R. Knight, "Forensic identification using skin bacterial communities." *Proc. Nat. Acad. Sci. United States Amer.*, vol. 107, no. 14, pp. 6477–6481, Jan. 2010.

[48] C. Colombini and A. Colella, "Digital profiling: A computer forensics approach," in *Availability, Reliability and Security for Business, Enterprise and Health Information Systems* (Lecture Notes in Computer Science), A. Tjoa, G. Quirchmayr, I. You, and L. Xu, Eds. New York, NY, USA: Springer-Verlag, 2011, pp. 330–343.

[49] C. M. Colombini, A. Colella, A. Castiglione, and V. Scognamiglio, "The digital profiling techniques applied to the analysis of a GPS navigation device," in *IMIS*, I. You, L. Barolli, A. Gentile, H.-D. J. Jeong, M. R. Ogiela, and F. Xhafa, Eds. Piscataway, NJ, USA: IEEE Press, 2012, pp. 591–596.

[50] C. Colombini, A. Colella, M. Mattiucci, and A. Castiglione, "Network profiling: Content analysis of users behavior in digital communication channel," in *Multidisciplinary Research and Practice for Information Systems* (Lecture Notes in Computer Science), G. Quirchmayr, J. Basl, I. You,

L. Xu, and E. Weippl, Eds. New York, NY, USA: Springer-Verlag, 2012, pp. 416–429.
[51] K. Gudjonsson. (2013, Apr.). *Log2Timeline—A Framework to Extract Timestamps from Various Artifacts and Combine Into a Single Timeline—Google Project Hosting* [Online]. Available: https://code.google.com/p/log2timeline/
[52] (2013, Jan.). *AutoHotkey* [Online]. Available: http://www.autohotkey.com/
[53] H. Sandklef. (2013, Jan.). *GNU Xnee* [Online]. Available: http://www.gnu.org/software/xnee/
[54] (2012, Oct.). *X11*, X.Org Foundation [Online]. Available: http://www.x.org/wiki/
[55] J. Sissel. (2013, Jan.). *Xdotool—Fake Keyboard/Mouse Input, Window Management, and More* [Online]. Available: http://www.semicomplete.com/projects/xdotool/
[56] Apple Inc. (2013, Feb.). *Apple Automator*, Cupertino, CA, USA [Online]. Available: http://www.macosxautomation.com/automator/
[57] Apple Inc. (2013, Jan.). *AppleScript Overview*, Cupertino, CA, USA [Online]. Available: https://developer.apple.com/library/mac/#documentation/AppleScript/Conceptual/AppleScriptX/AppleScriptX.html
[58] Oracle Java Documentation. (2012, Dec.). *Robot (Java Platform SE 6)*, Santa Clara, CA, USA [Online]. Available: http://docs.oracle.com/javase/6/docs/api/java/awt/Robot.html
[59] Oracle Java Documentation. (2012, Dec.). *Robot (Java Platform SE 7)*, Santa Clara, CA, USA [Online]. Available: http://docs.oracle.com/javase/7/docs/api/java/awt/Robot.html
[60] Microsoft. (2013, Feb.). *NTFS Technical Reference: Local File Systems*, San Francisco, CA, USA [Online]. Available: http://technet.microsoft.com/en-us/library/cc758691(v=ws.10).aspx
[61] ReWolf's blog. (2011, Oct.). *Windows SuperFetch File Format—Partial Specification*, Salt Lake, UT, USA [Online]. Available: http://blog.rewolf.pl/blog/?p=214
[62] Microsoft Support. (2013, Feb.). *Windows 7 & SSD: Defragmentation, SuperFetch, Prefetch*, San Francisco, CA, USA [Online]. Available: http://support.microsoft.com/kb/2727880
[63] Microsoft. (2013, Feb.). *System Restore—Microsoft Windows*, San Francisco, CA, USA [Online]. Available: http://windows.microsoft.com/en-US/windows7/products/features/system-re%store
[64] M. Russinovich. (2013, Feb.). *Inside the Registry* [Online]. Available: http://technet.microsoft.com/en-us/library/cc750583.aspx
[65] A. Castiglione, G. Cattaneo, G. De Maio, A. De Santis, G. Costabile, and M. Epifani, "The forensic analysis of a false digital alibi," in *Proc. 6th Int. Conf. Innovative Mobile Internet Services Ubiquitous Comput.*, Jul. 2012, pp. 114–121.
[66] H. Carvey. (2012, Feb.). *RegRipper* [Online]. Available: http://regripper.wordpress.com/
[67] N. Sofer. (2013, Feb.). *IECookiesView: Cookies Viewer/Manager for Internet Explorer* [Online]. Available: http://www.nirsoft.net/utils/iecookies.html
[68] N. Sofer. (2013, Feb.). *IE HistoryView: Freeware Internet Explorer History Viewer* [Online]. Available: http://www.nirsoft.net/utils/iehv.html
[69] N. Sofer. (2013, Feb.). *IECacheView—Internet Explorer Cache Viewer* [Online]. Available: http://www.nirsoft.net/utils/ie_cache_viewer.html
[70] AccessData Group, LLC. (2011). *Forensic Toolkit v4*, Lindon, UT, USA [Online]. Available: http://accessdata.com/products/computer-forensics/ftk
[71] P. Albano, A. Castiglione, G. Cattaneo, G. De Maio, and A. De Santis, "On the construction of a false digital alibi on the Android OS," in *INCoS*, F. Xhafa, L. Barolli, and M. Köppen, Eds. Arrowhead, CA, USA: IEEE Computer Society, 2011, pp. 685–690.
[72] P. Albano, A. Castiglione, G. Cattaneo, and A. De Santis, "A novel anti-forensics technique for the Android OS," in *Proc. Broadband, Wireless Comput., Commun. Appl., Int. Conf.*, 2011, pp. 380–385.

ANIELLO CASTIGLIONE (S'04–M'08) received a Degree and Ph.D. degree from the University of Salerno, Salerno, Italy, both in computer science. He joined the Dipartimento di Informatica ed Applicazioni "R. M. Capocelli" of University of Salerno.

He is a Reviewer for several international journals (Elsevier, Hindawi, IEEE, Springer, Inderscience, Wiley) and he has been a Program Chair and Member of international conference committees. He acted as a Guest Editor in several journals and several editorial board of international journals.

He is a member of various associations, including the Association for Computing Machinery, the IEEE Computer Society, the IEEE Communications Society, of GRIN (Gruppo di Informatica) and the International Information System Forensics Association, Italian Chapter (IISFA). He is a fellow of the Free Software Foundation (FSF) as well as Free Software Foundation Europe (FSFE). For many years, he has been involved in forensic investigations, collaborating with several Law Enforcement agencies as a consultant. His current research interests include data security, communication networks, digital forensics, computer forensics, security and privacy, and security standards and cryptography.

GIUSEPPE CATTANEO received a Degree in computer science from the Universita' di Salerno in 1983. Since 1986 he has been a Research Associate with the Dipartimento di Informatica ed Applicazioni, where he is currently a Associate Professor. From 1987 to 1990, he has been a Visiting Researcher at Laboratoire d'Informatique Theorique et Programmation (LITP), Universite Paris 6, Paris, France, working on a project aimed to the development of a Parallel Lisp Machine designing and implementing special purpose extensions to the functional language dedicated to the explicit parallelism management. Since 1993, he has been involved in research on system security, in particular experimental algorithm evaluation and algorithm engineering. In the last ten years, he has been a Team Leader, responsible for the local unit of 8 ICT projects co-funded by national large companies.

GIANCARLO DE MAIO received the Bachelors and Masters degrees in computer science from the University of Salerno, Salerno, Italy.

He is currently pursuing the Ph.D. degree in computer science under supervision of Prof. G. Cattaneo with the University of Salerno. In 2013, he was a Visiting Scholar with the University of California, Santa Barbara, CA, USA. He is a computer security enthusiast and his research interests mostly focus on Web Security, Mobile Security and Digital Forensics.

ALFREDO DE SANTIS received a Degree in computer science (*cum laude*) from the Universitá di Salerno, Salerno, Italy, in 1983. Since 1984, he has been with the Dipartimento di Informatica ed Applicazioni, Universitá di Salerno. Since 1990, he has been a Professor of computer science. From November 1991 to October 1995 and from November 1998 to October 2001, he was the Chairman of the Dipartimento di Informatica ed Applicazioni, Universita di Salerno.

From November 1996 to October 2003, he was the Chairman of the PhD Program in computer science with the Universitá di Salerno. From September 1987 to February 1990, he was a Visiting Scientist at IBM T. J. Watson Research Center, Yorktown Heights, NY, USA. He was with the International Computer Science Institute (ICSI), Berkeley CA, USA, in 1994, as a Visiting Scientist. From November 2009 to October 2012, he was with the Board of Directors of Consortium GARR (the Italian Academic & Research Network). His current research interests include algorithms, data security, cryptography, information forensics, communication networks, information theory, and data compression.