How to Forge a Digital Alibi on Mac OS X

Aniello Castiglione, Giuseppe Cattaneo, Roberto Prisco, Alfredo Santis, Kangbin Yim

How to Forge a Digital Alibi on Mac OS X

Aniello Castiglione[1], Giuseppe Cattaneo[1] Roberto De Prisco[1], Alfredo De Santis[1], and Kangbin Yim[2]

[1] Dipartimento di Informatica, Università di Salerno
84084 Fisciano (SA), Italy
[2] Dept. of Information Security Engineering, Soonchunhyang University
Asan, Chunknam, 336-745, Korea

Abstract. *Digital evidence* is increasingly being used in court cases. It consists of traces left on digital devices from which one can infer information about the actions performed on those digital devices. Digital evidence can be on computers, phones, digital cameras belonging either to an alleged offender or to third parties, like servers operated by ISPs or by companies that offer web services, such as YouTube, Facebook and Gmail. Digital evidence can either be used to prove that a suspect is indeed guilty or to prove that a suspect is instead not guilty. In the latter case the digital evidence is in fact an alibi.
However digital evidence can also be forged giving an offender the possibility of creating a *false digital alibi*. Offenders can use false digital alibi in a variety of situations ranging from ordinary illegal actions to homeland security attacks.
The creation of a false digital alibi is system-specific since the digital evidence varies from system to system. In this paper we investigate the possibility of creating a false digital alibi on a system running the Mac OS X 10.7 Lion operating system. We show how to construct an automated procedure that creates a (false) digital alibi on such a system.

1 Introduction

Modern technology permeates everyday life. Computers, tablets, smart-phones, GPS and other digital devices are widespread and are used in all sorts of activities: editing a spreadsheet, downloading a document, listening to a song, watching a TV program, browsing the Internet, paying a bill, chatting on a social network, and much more.

As the use of digital devices increases also the number of criminal or illegal actions perpetrated by using, or at least involving, such devices is growing. The use of digital devices often leaves digital traces. Many computer activities normally leave several traces of what has happened; mobile phones equipped with GPS might record the GPS coordinates of the locations that have been visited carrying the device; Internet activities leave traces in the logfile of the servers and in this last case the servers can be located anywhere in the world. These are just a few example of digital evidence. Any digital device can contain traces of activities performed using the device.

Digital evidence can be involved in court debates and can be used to provide evidence of crimes or more in general of illegal actions. There are several court debates where digital evidence has played a crucial role.

Digital evidence can consist of histories files, emails, content of computer memory, pictures on a digital camera, data on a mobile device. Generally speaking, digital evidence is information stored in a digital device. The online US legal definitions and legal terms dictionary [17] defines digital (or electronic) evidence as any probative information stored or transmitted digitally that can be used during the court trial.

However, digital evidence might also constitute an *alibi* for the defense of an alleged offender. The Latin word *alibi* literally means "*in or at another place*". The Merriam-Webster online dictionary [22], explains *alibi* as "the plea of having been, at the time of the commission of an act, elsewhere than at the place of commission". Digital evidence that somebody was using a specific computer located far away from the place where the offender acted might constitute an alibi for the user of the computer.

There are several examples of legal proceedings in which digital evidence has been considered an alibi that contributed to exonerate the alleged offender. Among these, an interesting case is the one that involved Rodney Bradford [18, 19], accused of armed robbery and released thanks to digital evidence proving that the alleged offender performed activities on his Facebook account at the same time when the crime was committed. The Erb Law Firm, a corporation of lawyers in Philadelphia, emphasized that "Facebook Can Keep You Out of Jail" [25]. Another example is the Italian case named "Garlasco" [20], in which the sentence of the first trial acquitted the alleged murder. The defendant's laptop contained digital evidence of work activity at the time the crime was committed. Offenders can use false digital alibi in a variety of situations. Homeland security attacks might also be performed exploiting a false digital alibi to cover the offender.

Digital evidence is *immaterial*. That is, the traces are bits stored in some storage device (hard disk and similar). Being bits stored somewhere, digital evidence can be modified by whoever has permission to access the memory storage where the bits are stored. For example, the administrator of a server can modify the logfiles that store information regarding all the accesses to the server.

Moreover, it is difficult, if not impossible, to identify the true originator of the digital evidence. Indeed even though we have digital evidence of some activities on a specific device, the digital evidence itself does not provide any information about *who* has produced it. For example, Bob claims that at a given time he has been at home working with his computer and he even posted some comments in a public blog and that there is digital evidence of these activities on his computer; if we are able to assert without any doubt that Bob's computer has indeed been used at the time Bob indicated and the activities performed are those claimed by Bob, we cannot be sure that really was Bob to use his computer. Bob might have asked somebody else to use his computer on his behalf, perhaps providing the password to access the computer.

As a matter of fact, Bob does not even need to ask somebody to use the computer on his behalf. Indeed it is possible to set up a sequence of automated actions that simulate a real user performing activities on a computer leaving digital evidence of the actions but without leaving any trace of the automation. Any action performed by an individual on a computer can be simulated by means of automated tools, including mouse clicks, pressure of keys, writing of texts, web browsing, and so on. Automating Internet accesses also produces traces in the servers that can be considered trusted third parties. This means that it is possible to forge a *false* digital alibi. However, care must be taken in order to not leave digital evidence of the automation.

The digital evidence left on a particular device is strongly dependent on the device and on the operating system running on the device. Hence the construction of a false digital alibi is system-dependent. For example, in a Windows based system the Windows Registry contains a wealth of information about the activities performed on the computer (e.g., [11]). Knowing the details of how the operating system stores information about the activities performed by the user is clearly crucial both for the analysis, for which we are interested in finding the information stored in the system, and for the construction of a digital alibi, for which we want to delete the information about the activities performed on the system. Many other technical details, like the type of filesystem, the use of virtual memory, the presence of automatic backup software play a crucial role.

In a recent paper [6] an automated procedure for the construction of a false digital alibi on systems running Microsoft Windows XP with Service Pack 3 and Microsoft Windows Vista has been described. In this paper we focus the attention on a system running Mac OS X 10.7 Lion and show how to construct an automation procedure for the construction of a false digital alibi.

2 Forging a digital alibi

To create a false digital alibi we design an automated procedure that can be scheduled to run on the chosen computer at a given time (in the absence of the user of the computer, which might be elsewhere at that time). The automated procedure will simulate the use of the computer with some activities that are normally performed by the user, such as text editing, web surfing and other Internet actions, leaving the normal digital evidence of such activities – the exact same digital evidence that would be left if the user performed those actions. To forge the digital alibi the user of the computer needs only to schedule (or run with an appropriate delay) the automated procedure before leaving the place where the computer is located.

Creating the automated procedure is not difficult. There are many tools available that make the task easy to accomplish also for non-expert users. However, the problem is not that of creating the automated procedure but that of deleting the digital evidence of the use of the automated procedure, leaving only the digital evidence of the "normal" actions.

2.1 Unwanted evidence

An automation can leave traces on the system that allow an inspector to realize that the automated procedure was used making void the alibi (actually in this case the false alibi can become evidence against the suspected person). Traces of the automation is referred to as *unwanted evidence*, and should be avoided or removed after the automation. Unwanted evidence can be left, for examples in the execution traces or logon traces. Often the digital evidence is stored in system logfiles. For example, almost all operating systems provide mechanisms to trace the execution of all the processes that get run on the machine, writing in specific logfiles information such as the executable name, the time it was started, the amount of CPU that was allocated during the execution, the maximum resident size for virtual memory and so on. Depending on the OS, the execution of an automation generated with tools like AutoIt also leaves this kind of traces. For example, Windows stores a lot of information in the Registry. In Linux, system logs are stored in the */var/logs* directory and memory map of processes is maintained in the */proc* directory. In a Mac OS X computer system logs are stored in the */private/var/log* directory. Most of recent OSs implement techniques like "Virtual Memory Allocation" and "Prefetch", which also store data about programs on the filesystem. Application specific data can also contain digital evidence. If a specific application used for the automation leaves unwanted evidence we must be careful in using that application. For example, if when using a OS X based system we decide to use an Applescript for the automation we have to be sure that the fact the Applescript gets executed is not logged somewhere (for example, in the shell history if we use a shell to launch the script).

2.2 Avoiding or removing unwanted evidence

In order to avoid traces of the automation one can take several precautions. The specific precautions depend on the particular system that one is using. If, in order to execute the automation, we are forced to create evidence of the automation, then it is necessary to remove the unwanted evidence. Whenever it is not possible either to avoid or to securely remove an unwanted trace (for example, when its location is write-protected), an a-priori obfuscation strategy could be adopted in order to avoid any logical connections between unwanted evidence and the automation procedure, in a way that the unwanted evidence could have been produced by a "normal" system operation.

While wiping unwanted evidence can be easily achieved using several wiping techniques (e.g., [10] [9] [21]), the actual problem is "how to erase the eraser". In [10] several methods that can be exploited to implement an automatic, selective and secure deletion/self-deletion are shown.

2.3 Iterative refinement

The automation needed to construct a false digital alibi can be constructed with an iterative techniques consisting of two phase:

1. development of the automation and
2. testing of the procedure on the target system.

The automation can be refined at every iteration by fine tuning the simulated actions and the wiping of the unwanted evidence. The process will stop when it produces an automated procedure that leaves only the wanted evidence.

Both activities could leave many traces in the target system and thus one must be very careful during the construction of the automation. The best solution is to construct the automation on a completely separated, but identical, system, like another computer or a virtual machine with the same characteristic of the target system. If the automation is constructed on the target system then one must take care also of unwanted evidence relative to the construction of the automation.

The specific strategy that we have used is the following. We started with a first version of the automation procedure, call it $automation_0$. Then we proceeded in constructing refined versions $automation_1$, $automation_2$, ... and so on by using the following technique to decide the refinements. Given version $automation_i$, run it starting from a pre-determined state of the system, say $S0$ and call $S1$ the corresponding final state of the system. Then, starting again from $S0$ perform manually the actions of the automation and call $S2$ the corresponding state of the system. Notice that when performing several actions there are delays between actions. Clearly, it is impossible to match the delays used in the manual execution of the actions with those of the automated execution. However, it is possible to use reasonable random delays in the automation. Having produced state $S1$ (automated execution of the actions) and state $S2$ (manual execution of the actions) we can compare the two states. In particular we can check all the files that have been either accessed or modified. By a careful inspection of the modified or accessed file list for the two states we can infer where the automation has left unwanted evidence. Then we can refine the automation in order to avoid the unwanted evidence. It is necessary to repeat the whole process because the modifications might create new unwanted evidence. The process stops when the states $S1$ and $S2$ obtained for a specific version $automation_n$ of the automated procedure are indistinguishable in the sense that there is no evidence of the use of the automation but only the evidence of the actions. That is, there is no way of telling that it was an automated procedure to execute the actions and not a real user.

3 Case study for Mac OS X 10.7 Lion

In this section we describe the construction of a false digital alibi on a Mac running OS X 10.7 Lion. The construction of a false digital alibi needs an automated tool that simulates the behaviour of a user working at the computer and a mechanism that deletes any evidence of the use of the automated tool. Writing a program or script that simulates a real user using the computer is quite simple. Avoiding to leave traces or deleting all the traces that are left by the program/script can be tricky. The difficulty of wiping all the evidence of the

use of the automated tool depends also on how we implement the automation and this, in turn, can make the automation itself not so easy. In the following section we explain how to create an automated tool for a Mac running OS 10.7 Lion and how to erase any trace left by the automation.

3.1 Unwanted traces on a Mac

It is important that no information is left about the execution of the automation scripts. Hence it is necessary to pay attention to a few things that can potentially leave traces. On the particular system that we are using, we have identified the following potential source of information leakage.

- *Logfiles.* As it happens in other systems, whenever the user executes programs or takes other actions, information about the actions executed gets written in specific files. These files are usually logfiles but any other type of file can be involved. As we will explain in more details in the next sections, we have identified a set of system files that get modified and can potential contain digital evidence of the taken actions. System wide logfiles are stored in */private/var/log*. Application specific logfiles can be anywhere.
- *Virtual memory.* If virtual memory is being used it is possible that a copy of the scripts gets saved in the virtual memory. Virtual memory swap files are written in */private/var/vm/*. To avoid a potential leakage of information the automated procedure will have to make sure that no new swapfile will be left in the directory.
- *Time machine.* Another potential leakage of information derives from the use of the Time Machine backup software. If during the execution of the scripts there is a planned backup session it is possible that relevant files will be copied on the Time Machine backup disk, potentially revealing the execution of the automation. So, it is necessary to disable the backup software so that no backup will be performed during the time when the automated scripts will run. Disabling the Time Machine can raise suspicion only if the user never disables the backup software. It will be enough to disable it randomly in order to not raise suspicions. Alternatively one can modify the backup schedule to obtain the wanted effect (that is, no backup will be performed during the time when the automated scripts will run).
- *Journaled filesystem.* A journaled filesystem could also potentially leave traces due to the storage of metadata about the files. If the chosen Mac does use a journaled filesystem (e.g. HFS+) then it is necessary to use an external device with a non journaled filesystem (e.g., FAT32). For this reason, we use a USB external pendrive to store the scripts needed for the automation procedure; the pendrive uses a FAT32 filesystem.

3.2 The sequence of simulated actions

We start by setting up a pre-determined sequence of actions that we wish to simulate, that is, the sequence of actions that the automated procedure will

take leaving the same evidence that would be left if the user itself takes the actions. These actions have been chosen with two goals: leave on the system digital evidence that constitutes the false digital alibi and facilitate the removal of unwanted evidence (the digital evidence of the execution of the automated procedure).

The specific set of actions that we used is shown in Listing 1.1. Clearly one can decide any arbitrary set of actions. As we will explain later, some of these actions have been chosen because they help in not creating or in removing the unwanted evidence.

```
1  Delay the execution (wait an appropriate time)
2  Launch iTunes and start playing a playlist
3  Launch Safari and use it to post a twit on twitter
4  Launch Pages and start writing a document
5  Go back to Safari and make a Google search
6  Visit a website in the list returned by the search
7  Launch Mail, write and send an email
8  Close iTunes
9  Go back to Page and finish the document, saving it to disk
10 Shutdown the computer
```

Listing 1.1. Simulated actions

3.3 The automated procedure

The automated procedure comprises three files:

1. the *launcher*, an Applescript that simply launches the scheduler-wiper.
2. the *scheduler-wiper*, a Python script that is responsible of launching the simulator and of deleting the traces of the execution of the simulator (clearly not those relative to the simulated activities but only those that might reveal the use of the script to perform the activities).
3. the *simulator*, an Applescript script that simulates the behaviour of a real user using the computer.

The launcher script is needed only to avoid the direct execution of the scheduler-wiper since executing directly the Python script would leave traces of its execution. Indeed, in order to directly launch the scheduler-wiper script, a Python script, we would need to use a shell. Normally the commands executed within a shell are saved in a shell history. Although it is possible to disable the shell history, such a choice is not common and can raise suspicion. So, we decided to avoid running commands directly from the shell. To run the Python script we use an Applescript that simply launches the Python script. The Applescript can be launched through the graphical interface, without opening a shell (and thus without saving any commands in the shell history).

All these files will be stored on an external storage device, such as USB pendrive, to avoid problems with either a journaled filesystem or with a backup

software like Time Machine. Moreover, the script have been saved as *.app* files, which are stand-alone executables. This is especially important for the Python script because no Library function will be called at the time of the execution.

We have chosen Applescript for the simulation because Applescript makes easy to create an automation, as we will explain in the sequel. Python has been chosen because the execution of a Python scripts leaves very few traces so there is little unwanted evidence to delete or to avoid. Moreover, the use of Python is preferable to other interpreted languages (like Java bytecode) since it does not require additional software not already shipped with the operating system (like the Java Virtual Machine).

The launcher Applescript. The launcher is a very simple Applescript since the only action that it has to perform is launching the scheduler-wiper script (which is written in Python). The entire code is a single line and is shown in Listing 1.2:

```
do shell script ``python /Volumes/PENDRIVE/helloWorld.py''
    with administrator privileges
```

Listing 1.2. The launcher Applescript code

Notice that the scheduler-wiper needs to be run with administrator privileges since in order to delete the traces of its own execution it will have to modify some files not accessible to a regular user. As we have already said we use this script only to avoid launching directly the Python scheduler-wiper script.

The scheduler-wiper Python script. The scheduler-wiper script has two functionalities: running the simulator and deleting all the unwanted evidence relative to the execution of the entire automation. Clearly, all the traces that are relative to the simulated activities have to be left on the system. However, no traces of the three scripts and of their execution have to be left on the system.

The first action that the scheduler-wiper takes is that of checking the status of some relevant system files. These files will be modified by the execution of the simulator and by the execution of the wiper, so they will need to be "touched" after the simulation in order to delete the traces of the existence of the scheduler and the wiper.

```
# list of files to restore
listOfFiles=["/usr/bin/srm",
"/System/Library/Frameworks/OSAKit.framework/...",
"/System/Library/Frameworks/ServerNotification.framework/...",
...
...
"/System/Library/ScriptingDefinitions/CocoaStandard.sdef"]

size=len(listOfFiles) # n. of paths

```

```
# init arrays
atimes=[0]*size # access times

# get last access time of each file
for i in range(size) :
  atimes[i]=os.path.getatime(listOfFiles[i])

### AUTOMATION ###

# run the applescript file
os.system("/Volumes/PENDRIVE/helloWorld.app/Contents/MacOS/
    applet")
```

Listing 1.3. Snippet 1 of the scheduler-wiper

Listing 1.3 provides a snippet of code of the scheduler-wiper Python script. The last line of this snippet contains the call to the simulator script which performs all the wanted actions (that is, the ones listed in Listing 1.1) for which we want to leave the digital evidence needed for the false alibi. In the next section we provide more details about the simulator script.

Then the scheduler-wiper deletes in a secure way, using the *srm*, command, the 3 files containing the scripts, which are on the external USB pendrive. Notice that it is possible to delete these files even though the system is executing the scheduler-wiper because the Python language is interpreted and the entire file is loaded by the interpreter when the file gets executed; hence the physical copy can be removed without affecting the execution of the script.

Then the wiper goes through the list of system files that could potentially reveal that the simulation scripts have been executed and restores the initial status of those files in such a way that there is no trace of the execution of the simulation scripts. Also the swapfiles are deleted.

Listing 1.4 shows the relative snippet of code.

```
# delete the applescript launcher
os.system("srm -r /Volumes/PENDRIVE/launcher.app")
# delete the applescript simulator
os.system("srm -r /Volumes/PENDRIVE/simulator.app")
# delete the python script
os.system("srm -r /Volumes/PENDRIVE/helloWorld.app")

#delete the swap file modified during the script's execution
for root,dirs,files in os.walk("/var/vm") :
    for f in files :
        if f!="sleepimage" :
            swapFilePath=os.path.join(root,f)
            mtime=os.path.getmtime(swapFilePath)
            if(mtime>nowMilliseconds) :
                os.system("srm "+ swapFilePath)

### Reset access time ###
```

```
18
19 for i in range(size):
20     atime=os.path.getatime(listOfFiles[i])
21     touchTime = millsToDate(atimes[i])
22     if(atime>atimes[i]) :
23         os.system("touch -c -t " + touchTime + " \"" +
               listOfFiles[i] + "\"") # set both last access and
               last modified time (-c do not create file, -t
               specified time)
24
25 ### Turn off the system ###
26
27 os.system("sudo shutdown -h now")
```

Listing 1.4. Snippet 2 of the scheduler-wiper

The wiper deletes also any swap file left in the */private/var/vm* directory.

The simulator. To simulate a user working at the computer we can use an Applescript. Applescript is a scripting language, integrated in the Mac operating system, specifically designed to control other applications. Using Applescript is very easy to schedule user actions, since it allows to launch specific applications and execute specific actions within the applications. It even allows to simulate keyboard typing. For example, the following code snippet written in Applescript simulates the use of iTunes for listing a playlist:

```
1 tell application "iTunes"
2   delay 3.47
3   play playlist 1
4 end tell
```

Listing 1.5. Applescript code snippet for using iTunes

Note that the *delay* command does not make any guarantees about the actual length of the delay, and it cannot be more precise than 1/60th of a second. However, this is enough to simulate random delays between user actions.

A slightly more complicated code snippet is required to simulate an access to Twitter, by means of Safari, and the posting of a comment:

```
1 tell application "Safari"
2   activate
3   open location "https://twitter.com/"
4   delay 30
5   tell application "System Events"
6     keystroke "USER"
7     delay 5.12
8     keystroke tab
9     delay 7.3
10    keystroke "PASSWD"
11    delay 3.8
```

```
12      keystroke return
13      delay 15.6
14      keystroke tab
15      delay 8.21
16      keystroke tab
17      delay 7.48
18      keystroke "What a nice day!!!"
19      delay 3.21
20      keystroke tab
21      delay 5.45
22      keystroke return
23    end tell
24  end tell
```

Listing 1.6. Applescript code snippet for posting on Twitter

By properly writing the simulator we can simulate almost any real behaviour. Some actions can be more complicated than others. However, the Applescript language is powerful enough to allow the simulation of almost any action.

Particular attention has to be paid to the scheduling of the actions: the timing should be reasonable in order to not create any suspicion. For example, if we are simulating the writing of a long document, then we should leave enough time between the launching of the text editor and the saving of the file so that in the elapsed time a real user can actually type all the necessary keystrokes.

4 Testing

In order to test the automation we have operated in a virtual environment. We have created a virtual machine and installed the Mac OS X 10.7 Lion operating system. The disk for the virtual machine is an external USB hard disk previously formatted with a low-level writing procedure. Beside the operating system we installed the iWork software in the virtual machine and we copied some mp3 files to be used with iTunes. Moreover, we configured the following applications: Mail, iTunes and Pages. After the setting phase the virtual machine has been shut down and the external hard disk containing its filesystem has been copied bit by bit on an another external hard disk having the same physical dimension. We will refer to this disk image as the *initial disk state*. At this point we proceeded with two copies of the virtual machine starting from the initial state.

In the first copy we plugged in the USB pendrive with the scripts that accomplish the automation and we executed them as described earlier in the paper. The *launcher.app* script has been run with a double click. The script requested the administrator password and after that it executed all the actions that we described in the previous sections, without any further human intervention. At the end the virtual machine was automatically shut down. We will call the resulting disk state *automated disk state* (this is the disk state after the automation).

In the second copy, starting again from the initial state, we manually executed the set of actions that the automation comprises and we shut down the machine.

For this case, we will call the resulting disk state *manual disk state* (this is the disk state after the manual execution of the actions).

4.1 Iterative refinement

In order to produce the final version of the automation scripts we have used the technique described in Section 2.3, where $S1$ is the automated disk state and $S2$ is the manual disk state. As an example, we describe in the following one specific iteration.

A file by file analysis of the two states, the automated disk state and the manual disk state, revealed all the files that were either accessed or modified in each of the two cases. The vast majority of the files accessed or modified for both cases were relative to the use of the applications. For example, sending the email causes the creation of files in */Users/userName/Library/Mail*. From these file it is impossible to tell whether they were created by the manual execution or by the automation.

Among the files that were accessed only by the automated procedure we found the following list of files (the dots mean that we specified only the directory under which there are a number of files accesses by the automation):

```
1 /System/Library/Frameworks/OSAKit.framework/...
2 /System/Library/Frameworks/ServerNotification.framework/...
3 /System/Library/PrivateFrameworks/AOSKit.framework/...
4 /System/Library/PrivateFrameworks/AOSNotification.framework
      /...
5 /System/Library/PrivateFrameworks/SyncServicesUI.framework/...
6 /System/Library/ScriptingDefinitions/CocoaStandard.sdef
```

Listing 1.7. Accessed files

We are not sure that one can infer the use of the automation by the fact that these files have been accessed, but to be on the safe side the automation script has been refined in order to restore the access time of these files as in the initial disk state.

Among the files not relative to the set of simulated actions we found the following list of files:

```
1 /private/var/log/asl/2012.05.29.G80.asl
2 /private/var/log/asl/2012.05.29.U0.G80.asl
3 /private/var/log/asl/2012.05.29.U501.asl
4 /private/var/log/asl/AUX.2012.05.29
5 /private/var/log/asl/AUX.2012.05.29/44545
6 /private/var/log/asl/AUX.2012.05.29/44547
7 /private/var/log/asl/AUX.2012.05.29/44549
8 /private/var/log/asl/BB.2013.05.31.G80.asl
9 /private/var/log/asl/StoreData
10 /private/var/log/DiagnosticMessages/2012.05.29.asl
11 /private/var/log/DiagnosticMessages/StoreData
12 /private/var/log/opendirectoryd.log
```

```
/private/var/log/secure.log
/private/var/log/system.log
```

Listing 1.8. System-wide log files

A manual inspection of these files showed no traces of the automation (this in fact depends on the fact that in previous refinement we have taken steps to avoid unwanted evidence). In particular there were no traces relative to the use of Applescript which is the main evidence of the use of an automation.

Clearly we kept refining the scripts until we obtained a version of the automation for which no unwanted evidence was left in the system.

4.2 Forensic analysis

The iterative refinement technique has been used to improve to automation scripts up to the point of obtaining a script that behaves exactly as a real user and does not leave any evidence of the automation. However, to validate the false digital alibi we have to execute a forensic analysis of the state of the system after the automation. For example, it is necessary that there be no unwanted evidence of the automation not only in the files of the filesystem but also on the erased portion of the disk. To cope with leakage of information in deleted files we always use secure deletion. We will provide more details about the forensic analysis in the extended version of this paper.

5 Conclusions

Digital evidence contains information about actions taken on a computer, like logon data, the use of specific applications, web histories, command histories, and much more. Digital evidence is becoming relevant as a consequence of the widespread use of digital devices. Many court cases nowadays involve digital evidence. However, digital evidence can be fake: a *false digital alibi* can be constructed. A false digital alibi can be constructed by setting up an automated procedure that executes actions (writing a document, visiting websites, posting comments, etc) without the physical presence of the user who can be elsewhere when the actions are automatically performed on his computer. The automation can leave digital evidence of itself. However by carefully crafting the automated process one can either avoid the digital evidence of the automation or delete it afterwords. In [6] it has been shown how to set up an automated procedure to create a false digital alibi for a Windows based system (the specific OS considered are Windows XP with Service Pack 3 and Windows Vista). The creation of a false alibi heavily depends on the particular operating system as the digital evidence left is system specific. In this paper we have showed how to construct a false digital alibi on a system running Mac OS X (specifically, 10.7 Lion).

The false digital alibi constructed as a case study comprises a specific set of actions ranging from using iTunes for listening to a playlist to surfing the web using Safari and posting comments on public website. The set of actions

was carefully chosen in order to not leave digital evidence of the automation. Further study might include the investigation of which actions can be safely simulated and which ones create trouble for the deletion of the digital evidence of the automation. The case study has used a specific version of the Mac OS X operating system. An interesting deeper investigation would be that of understanding whether the false digital alibi can be constructed with different versions of the operating system. We believe that this should be doable, perhaps with some modifications to the automated procedure.

Acknowledgements

We would like to thank Dario Di Nucci, Fabio Palomba and Stefano Ricchiuti for helping with the testing of the automated procedure.

References

1. 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*, Proceedings of the Third International Conference on Intelligent Networking and Collaborative Systems (INCoS-2011), pp. 685-690, IEEE 2011, Fukuoka Institute of Technology, Fukuoka, Japan, November 30 - December 2, 2011.
2. H. Carvey, *Windows Forensics Analysis, Second Edition*, Syngress, 2009.
3. V. Chandola, A. Banerjee and V. Kumar, *Anomaly detection: A survey*, ACM Computing Surveys, vol. 41, n. 3, July 2009.
4. W. Craig, K. Dave and S.R.S. Shyaam, *Overwriting Hard Drive Data: The Great Wiping Controversy*, Lecture Notes in Computer Science (Springer), Vol. 5352, pp. 243-257, December 2008.
5. A. Castiglione, G. Cattaneo, A. De Santis and G. De Maio, *Automatic and Selective Deletion Resistant Against Forensics Analysis*, Proceedings of the 2011 International Conference on Broadband, Wireless Computing, Communication and Applications (BWCCA 2011), Barcelona, Spain, IEEE, pp. 392-398, 2011.
6. A. De Santis, A. Castiglione, G. Cattaneo, G. De Maio, M. Ianulardo, *Automated Construction of a False Digital Alibi*, in Proceedigns of ARES 2011, Lecture Notes in Computer Science n. 6908, pp. 359–373, 2011.
7. G. De Maio, A. Castiglione, G. Cattaneo, G. Costabile, A. De Santis, and M. Epifani, *The Forensic Analysis of a False Digital Alibi*, Proceedings of the Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS 2012), IEEE 2012, Palermo, Italy, July 4-6, 2012.
8. N. Fierer, C.L. Lauber, N. Zhou, D. McDonald, E.K. Costello and R. Knight, *Forensic identification using skin bacterial communities*, Proceedings of the National Academy of Sciences, Abstract, March 2010.
   http://www.pnas.org/content/early/2010/03/01/1000162107.abstract
9. P. Gutmann, *Data Remanence in Semiconductor Devices*, 2001 Usenix Security Symposium, Washington DC, August 2001.
   http://www.cypherpunks.to/~peter/usenix01.pdf
10. P. Gutmann, *Secure Deletion of Data from Magnetic and Solid-State Memory*, Sixth USENIX Security Symposium Proceedings, San Jose, California, July 22-25, 1996.

11. V. Mee, T. Tryfonas and I. Sutherland, *The Windows Registry as a forensic artefact: Illustrating evidence collection for Internet usage*, Digital Investigation, vol. 3, pp. 166-173, 2006.
12. R. Poisel, S. Tjoa, and P. Tavolato, *Advanced File Carving Approaches for Multimedia Files*, Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA), Vol. 2, No. 4, pp. 42-58, 2011.
13. M. B. Salem, S. J. Stolfo, *Combining Baiting and User Search Profiling Techniques for Masquerade Detection*, Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA), Vol. 3, No. 1/2, pp. 13-29, 2012.
14. D.E. Shelton; *The 'CSI Effect': Does It Really Exist?*, National Institute of Justice Journal No. 259, March 17, 2008.
15. A. Silberschatz, P. B. Galvin and G. Gagne, *Operating System Concepts, 7th Edition*, Wiley, 2004.
16. *Internet world stats*, June 30, 2010.
http://www.internetworldstats.com/stats.htm
17. U.S. Legal, Inc. "Legal Definitions and Legal Terms Dictionary".
http://definitions.uslegal.com
18. The New York Times, *I'm Innocent. Just Check My Status on Facebook*, November 12, 2009.
http://www.nytimes.com/2009/11/12/nyregion/12facebook.html?_r=1
19. CNN, *Facebook status update provides alibi*, November 12, 2009.
http://www.cnn.com/2009/CRIME/11/12/facebook.alibi/index.html
20. Xomba: A Writing Community, *Garlasco, Alberto Stasi acquitted*, December 2009.
http://www.xomba.com/garlasco_alberto_stasi_acquitted
21. U.S. Department of Defense, *DoD Directive 5220.22, National Industrial Security Program (NISP)*, 28 February, 2010.
22. *Merriam-Webster online dictionary.*
http://www.merriam-webster.com/
23. Wikipedia, *KVM switch.*
http://en.wikipedia.org/wiki/KVM_switch
24. *NIST Special Publication 800-88: Guidelines for Media Sanitization*, p. 7, 2006.
25. The Erb Law Firm, *Facebook Can Keep You Out of Jail*, November 2009.
http://www.facebook.com/note.php?note_id=199139644051
26. Wikipedia, *Five Ws.*
http://en.wikipedia.org/wiki/Five_Ws
27. U.S. Government House of Representative, *Federal Rules of Evidence*, Dec 2006.
http://afcca.law.af.mil/content/afcca_data/cp/us_federal_rules_of_evidence_2006.pdf