Automatic, Selective and Secure Deletion of Digital Evidence

Aniello Castiglione, Giuseppe Cattaneo, Giancarlo De Maio, Alfredo De Santis

Dipartimento di Informatica *"R. M. Capocelli"*
Università degli Studi di Salerno, I-84084, Fisciano (SA), Italy
castiglione@acm.org, cattaneo@dia.unisa.it, demaio@dia.unisa.it, ads@dia.unisa.it

***Abstract*—The secure deletion of sensitive data can improve user privacy in many contexts and, in some extreme circumstances, keeping some information private can determine the life or death of a person. In fact, there are still several countries where freedom of expression is limited by authoritarian regimes, with dissidents being persecuted by their government. Recently, some countries have begun to make an effort to aid these people to communicate in a secure way, thus helping them to gain freedom. In this context, the present work can be a contribution in spreading the free use of Internet and, in general, digital devices.**

In countries where freedom of expression is persecuted, a dissident who would like to spread (illegal) information by means of the Internet should take into account the need to avoid as many traces as possible of his activity, in order to mislead eventual forensics investigations. In particular, this work introduces a methodology to delete a predetermined data set from a digital device in a secure and fast way, for example, with a single click of the mouse. All the actions required to remove the unwanted evidence can be performed by means of an automation, which is also able to remove traces about its execution and presence on the system. A post-mortem digital forensics analysis of the system will never reveal any information that may be referable to either the deleted data set or automation process.

***Index Terms*—Digital Forensics; Privacy; Secure Deletion; Wiping; Anti Forensics; Free communication; Anti-Forensics; Counter-Forensics.**

I. Introduction

The secure deletion of data from a media is an extensively covered topic which finds numerous applications in many areas of Computer Science. In particular, this work focuses on the importance of secure deletion for user privacy and Digital Forensics purposes. The key concept is that erasing data from digital devices is not a simple task, since some inherent characteristics of digital systems can retain recoverable information about it (i.e., data remanence). In substance, data remanence can be considered a problem to solve in order to grant user privacy in many situations where secure deletion is crucial, as well as a benefit to exploit so as to gain useful evidence in a digital forensics analysis.

In this paper, a new methodology for secure deletion is proposed. With respect to existing works, the challenge has been that of adding more strict requirements to the entire process. In particular, methods and techniques to realize an Automatic, Selective and Secure Deletion (*ASSD*) of digital evidence, avoiding suspicious artifacts, are discussed. Such methodology can be useful both in improving user privacy in some contexts as well as providing guidelines for digital forensics analysts interested in finding evidence of automated deletion procedures.

This work can be also considered a small contribution in spreading the freedom of using the Internet and, in general, digital devices in everyday life. In fact, there are still countries where freedom of expression is persecuted, even though, recently, some efforts have been made to aid those who are oppressed in using the Internet [1]. The methods presented in this paper can be useful for a dissident who intends to delete some information in a fast and secure manner (i.e., in a few seconds during a police inspection), in order to divert a digital forensics analysis on his devices.

A. Secure Deletion

In this work the *secure deletion* of sensitive data is considered a task which involves the removal of any information regarding such data as well as any evidence left by the deletion procedure. This includes the following steps:

1) the rewriting of the physical locations where sensitive data is memorized (wiping), in order to prevent any possibility of recovery;
2) the removal or obfuscation of metadata, which might be recorded by the filesystem or the OS, having reference to the wiped data;
3) the removal or obfuscation of "unwanted" and "suspicious" traces.

The term "unwanted" refers to traces, evidence or data through which it is possible to recover deleted information, for example by means of a file carving operation.

When considering a digital forensic analysis on the system, it is important to note that some traces — though containing no meaningful information about the erased data — can be considered "suspicious" if linkable to the deletion process. For example, the presence on the system of a wiped disk, or an encrypted partition, or a wiping software is an atypical evidence that may create suspicion.

B. Selectivity

The secure deletion of unwanted data can be implemented by using many techniques. The simplest one consists of wiping the entire partition where the information resides, but this

approach raises at least two problems: in a digital forensics analysis, the presence of a void partition can create suspicion, while wiping an entire partition also requires a great deal of time.

The choice of the right technique depends on the context, with a simple one not necessarily being viable in situations where a *selective* deletion is required. An individual could be interested in removing only specific software from his PC which is used, for example, to create a false digital alibi [2].

C. Automation and Self-Deletion

In addition to the selectivity, another important requirement could be that of *automating* the deletion procedure, when certain data have to be deleted repeatedly or in a systematic manner. An automate and selective deletion procedure could be useful in many situations, such as to speed up and simplify the systematic deletion of certain data from a system, to guard a person (e.g., a spy) who has been uncovered and does not want to reveal some information, to aid a political dissident who has to quickly hide suspicious data on his PC before it is confiscated by a Law Enforcement Agency.

Therefore, an automatic and secure deletion procedure, along with the deletion of a predetermined data set, should also include the removal of any traces left by its execution and presence on the system. In other words, the procedure has to implement a *self-deletion* technique, which should take into account the presence of the procedure stored on the disk and any evidence of its execution. An *ASSD* process should include procedures to remove or obfuscate both unwanted and suspicious traces. Digital evidence removed using these precautions should not leave any meaningful information and therefore should be resistant against a digital forensic analysis.

II. Deletion Issues

Typical operations implemented by the OS, such as removing a file or formatting a storage media, do not guarantee that any unwanted data is effectively removed. In general, a number of rewritings of the occupied physical locations are necessary to accomplish this task. A residual representation of data, called *data remanence*, can persist on the drive for many reasons:

- the OS could provide a file retention facility, in charge of moving deleted files to a holding area which allows users to easily revert mistakes (e.g., the Recycle Bin);
- when a file is deleted, the OS generally removes its entry from the filesystem directory marking the previously occupied space as “unused” and “free” for subsequent writings, but data of the deleted file remains on the media until rewritten;
- filesystem metadata (i.e., pathname, last modified time, etc.) about deleted files can be maintained by the system for a certain period of time;
- files are often stored in non-contiguous physical locations, making it more difficult to overwrite them;
- the filesystem could implement optimization techniques which do not guarantee that file rewriting happens onto the previous occupied device locations;
- reformatting an entire drive or a partition generally implies the destruction of the filesystem metadata, but does not guarantee that the data present in the formatted area is completely destroyed;
- some actual storage technologies, such as flash memory, have inherent limitations that make it difficult to rewrite a specific physical location;
- even when the storage medium is overwritten, physical properties of the medium could make it possible to recover some previous contents by means of electron microscopes.

Since bypassing hardware constraints, as those regarding the solid state memories, is sometimes impracticable [3], in this work only magnetic memories such as hard disks are taken into account.

Performing a secure deletion is more complicated when different kinds of data should be erased. For example, the secure deletion of software requires the deletion of applications and their data, which is typically contained in various files and system structures (e.g., the Windows Registry). In this case, a deep analysis of digital evidence produced by the software should be performed by using specific tools [2]. An alternative approach is to “organize” the data to be deleted in order to simplify the deletion procedure. In this work various methods belonging to the latter approach are analyzed in Section III-G.

A. Wiping Issues

Deleting a file using the OS-specific functions does not guarantee that it is completely removed from the drive. In fact, the sectors that were occupied by the file become available for a new writing operation, but the previous data remains on the disk until it is overwritten: this operation is commonly known as “unlinking” [4].

The amount of rewritings required to completely remove any traces of certain data from a media is a controversial theme. Some works support the thesis that electromagnetic evidence can be recovered from medias by means of electron microscopes. The Gutmann method [5] [6] asserts that 35 rewritings with different patterns are needed to guarantee irrecoverability of data, while the DoD technique [7] claims that 7 rewritings should be sufficient. On the contrary, in the NIST Special Publication 800-88 [8] it is stated that, on modern storage media, a single overwrite of data should guarantee its irrecoverability. This last thesis is also supported by Wright et al., who in a recent work [9] have proven that “Although there is a good chance of recovery for any individual bit from a drive, the chances of recovery of any amount of data from a drive using an electron microscope are negligible”.

Even if any of these methods can be adopted to realize the methodology discussed in this paper, a wiping technique that relies on a single overwriting operation has been adopted in the experiments conducted by the authors. This method turned out

to be resistant against an analysis performed by using typical digital forensics tools.

B. Automation and Self-Deletion Issues

Automating a secure deletion procedure implies the execution of a process which is in charge of performing all the required actions. Most modern OSes optimize the execution of programs by recording information in some internal structures for user profiling, caching and so on. It can occur that also evidence regarding the automation are recorded for these purposes. In this case, it should be cleaned by the automation itself or obfuscated in order to divert any eventual forensic analysis.

Self-deletion is a crucial feature which a fully-fledged deletion procedure should implement. Unfortunately, OSes typically lock running executables forbidding any processes to modify them. This is done to preserve the read-only property of the *code segment*. Thanks to this property, multiple processes (e.g., resulting from a `vfork()` operation) can run the same program because the same executable code is safely shared among them [4]. As a result, a program cannot modify/delete/wipe itself from the disk whilst running.

The methodology proposed in this paper bypasses this limitation exploiting the properties of the interpreted programming languages. An interpreted program is not directly executed by the OS, with the "interpreter" being in charge of executing its instructions. While in a compiled language the whole program is first decoded and then executed, in an interpreted language each line of the code is decoded and executed in sequence. In general, especially when the program is small, the interpreter loads all the instruction into the memory and does not lock the executing file, thus permitting (self) modifications.

III. The Proposed Methodology

Secure deletion of a predetermined data set can be performed using different techniques involving existing tools or ad-hoc crafted solutions. The use of existing software has the advantage of simplicity but can leave more unwanted traces on the system. However, auxiliary tools are useful in some cases where the amount of evidence to be deleted is large or "unpredictable". In particular, the secure deletion of an application requires the removal of all data left on the system by that application; it is generally stored in different files, directories and system structures (such as the Windows Registry), which makes it quite impossible to identify all the traces to remove.

Depending on the context, there are different strategies to realize an *ASSD* procedure. In Section IV, the implementation of a possible solution for a "basic" case is presented. In order to adopt the appropriate actions, it is necessary to identify all the elements that can influence the operational scenario.

A. Unwanted Traces

All evidence left by the automated deletion procedure and referable to the wiped data should be considered unwanted. While some precautions may be taken in order to prevent as many unwanted traces as possible, there are some OS and filesystem mechanisms, crucial for the system functioning, that record information about applications, files and user activities. Moreover, data remanence on the media can be due to the storage technology.

Mechanisms that may interfere with the deletion procedure can be organized into categories, each of which has to be addressed in a different manner:

1) OS recordings maintained in internal structures such as the Windows Registry;
2) filesystem structures such as the file table or the journal, which respectively maintain metadata (file name, size, creation time, etc.) of files and operations (accesses, modifications, etc.) made on them even after their unlinking;
3) algorithms used to manage and access data on the storage device.

OS or hardware constraints often make it difficult to modify internal structures whilst the system is running in order to prevent faults. In such cases, an obfuscation strategy should be adopted to mask non-removable traces as something that is not suspicious. For example, one should use common filenames, as well as launch applications from the command line in order to avoid meaningful recordings in the Registry. One can also use portable version of the software from a removable device to avoid the creation of temporary files on the main drive.

B. The Basic Case

In the simplest case, which is referred to as the "basic" case, the predetermined data set is a list of files storing sensitive information (e.g., text files containing a political propaganda) that has to be deleted in a secure and fast manner. No applications are involved, thus having no unwanted traces dispersed on the system. However, traces left by the deletion procedure (i.e., the automation program) should be considered and removed. A methodology viable for resolving the basic case is presented, which consists of different and mutually dependent phases. Finally, some methods useful in setting up a system in order to realize an *ASSD* process for complex data sets, including applications and their traces, are discussed. An aim of these approaches is to encapsulate as much unwanted data as possible into few system locations, thus reducing a complex problem into a basic one.

C. Wiping

The first choice for realizing an *ASSD* procedure regards the wiping method. It is important to note that, especially on journalized filesystems such as NTFS [10], overwriting a file does not guarantee that the physical locations occupied by it on the media are rewritten. This risk increases when the file is large and fragmented. It is mostly due to optimization techniques that are in charge of reducing access time to the disk, maintaining data in the memory until an optimal amount of it is ready to be written. Moreover, coherency mechanisms of transactional filesystems could also postpone writings. In both cases, the filesystem can relocate the file to different

physical locations for some reasons, such as recovering data in the presence of faults or obtaining a better occupation of free space, thus reducing file fragmentation.

Due to the reasons described above, data wiping on modern filesystems is not a simple task. Most programming languages do provide neither API which include primitives for wiping nor primitives for overwriting files in place, that is, using the same physical blocks. It is interesting to note that some common utilities deployed with OSes, such as `shred` on Linux, are not guaranteed to work properly on modern transactional filesystems (see `man shred`).

These problems can be bypassed by opening a file using the "random access" method, which is provided by most of the OSes through the `open()` system call. The semantics of this operation, in fact, permits nonsequential (or random) access to the file contents, making it possible to use the `seek` system call in order to read from or write to a particular location. Several experiments on different filesystems (including NTFS and EXT3) have confirmed that files are overwritten in place when using this technique.

The new values of the blocks to be cleaned can be chosen according to different methods. The choice depends on a trade-off between security and efficiency, since some values that can be easily and quickly generated could be considered suspicious after a forensics analysis. Four viable methods, which adopt different data sources for the overwriting operations, are presented below.

1) Naive Method: The simplest approach is to overwrite the file with a fixed pattern, such as all zeros. This technique is very fast as no more operations except writings on the media are needed, with it thus being possible to reach the maximum transfer rate of the device. The disadvantage of this method is that a digital forensics analysis can reveal anomalies as a repeated data pattern in some disk blocks, which can raise suspicions of having performed a wiping operation.

2) Randomization Method: The physical blocks occupied by the file can be overwritten with data randomly generated or obtained by means of a Pseudo-Random Number Generator (PRNG). The PRNG should be cryptographically secure in order to prevent that a deep analysis on the written data can reveal that such a sequence has been generated by an algorithm.
This approach has the advantage that it can be straightforwardly implemented, due to (pseudo) random number generators being implemented for most of programming language and are ready-to-use. Moreover, the overhead introduced to generate the data to be written, in terms of execution time, is typically low on common PCs. The main drawback comes from the consideration that an analysis of the overwritten blocks will reveal an atypical high entropy of data, with it being suspicious.

3) Data Replication Method: A more secure approach can be that of moving data already present on the disk onto the blocks to overwrite. The blocks to replicate can be chosen by the automation at runtime in a random or predetermined manner. In the first case, data can be read from some randomly chosen files or raw drive locations. Alternatively, a specific file or set of files can be used as the source for the wiping procedure.
Assuming that data replication is common when using modern OSes and filesystems, due to the mechanisms described in the previous sections, even a detailed forensics analysis cannot establish whether data copied onto the overwritten blocks has been transferred by means of an ad-hoc procedure or a common system process. Although this technique guarantees a high level of robustness, it has a weighty drawback: reading at runtime the data to be copied requires further accesses to the drive, which can determine a performance degradation, especially when the file to be overwritten is large.
Alternatively, a certain amount of data can be copied at runtime from the media into the memory, and then used as the pattern to rewrite the entire set of blocks. With respect to the previous solution, this can improve performance at a small expense of security.

4) Code Mutation Method: A more sophisticated method consists of transforming the content of the files to be cleaned into something different but more plausible. In the case of a program, it can be "polymorphed" into another program. The automation, created by means of an interpreted language, can contain a bytecode that itself can inject into its executable at runtime. In this case, it is not crucial that the entire executable content is overwritten. In fact, the probability that a forensics analysis on the unused disk blocks reveals meaningful traces about the automation can be considered negligible when a consistent portion of the program is lost.
This method is very robust due to the rewritten files having a plausible content which should divert any suspicions on their nature. However, this is strictly dependent on the size of the data to wipe. In fact, it could be difficult to generate a plausible new content for a large file.

The first method is the simplest and fastest, while the fourth one is the most secure as well as the most difficult to implement. Clearly, it is possible to adopt more than one overwriting method, based on a per-file policy: for example, code mutation for cleaning the automation program, data replication for small-sized files as well as randomization for large files.

D. Automation and Self-Deletion

The second step involves the choice of the technique for automating the deletion procedure and removing its presence from the system. The automation should make use of the wiping procedure to remove a predetermined set of files from the system, including itself, as well as require no human intervention. The procedure should take a list of files as argument and iterate the wiping procedure for each element, including itself in order to perform the self-deletion. This methodology proposes the use of an interpreted language to implement the automation, which allows a program to modify (i.e., wipe) its executable file(s) whilst running, as discussed in Section II-B. A separate shell script can be prepared in order to implement the "one-click" deletion feature. It could maintain

the list of files to be deleted and pass it to the automation process.

E. Memory wiping

The methodology proposed in this work can be enhanced by introducing an additional requirement: the resistance against a live forensics analysis. In order to accomplish this task, it is necessary to remove any traces left by the *ASSD* procedure from the main memory. Some possible solutions are:

- *Execute a memory diagnostic tool*: It is possible to execute a memory diagnostic tool as the last statement of the automation. These tools generally rewrite the memory many times in order to test the memory banks. The process will continue after the automation ends, and will overwrite the memory frames used by it.
- *Execute a heavy process*: A videogame or a similar memory-consuming process can be launched just before the termination of the automation. This should overwrite any free memory frame, including those previously reserved to the automation.
- *Perform a system shutdown*: A system shutdown will reset any connected volatile memories, including the main memory.

F. Testing

A testing phase should verify that the implemented *ASSD* procedure meets all the requirements discussed in Section I. Moreover, a significant number of experiments with different classes of input (i.e., size and number of files to be removed) should be performed in order to be confident that the procedure works correctly.

Advanced tools should be used to verify that no remanence of wiped data persists on the system. In particular, the deletion procedure should resist file carving as well as to deeper low-level analysis. The approach proposed in this work consists of setting data to be deleted with a predefined pattern, and subsequently verifying that this pattern is no longer present on the entire drive by means of a raw scanning of the involved partition. Although this technique can require a long time, it is as general as possible since it is totally independent from the used filesystem and storage media.

The testing should also include the research for suspicious traces, such as those regarding the execution of applications used for implementing the *ASSD*. Also in this case, the most general approach is to perform a raw scan of the media in order to find references to such applications. It is well-known that some information regarding removed files, such as that contained in the internal OS structures, may remain because it is difficult to delete it whilst the OS is running. In this case, filenames should be previously chosen with the aim of avoiding suspicious traces. Whenever the testing phase reveals the presence of suspicious digital evidence, it is necessary to refine the implemented *ASSD* procedure and then rerun the testing phase.

G. Deletion of complex data sets

There are circumstances where the secure deletion could involve several data sparse on the system, for example an entire data repository of a dissident. The deletion of entire applications and respective traces is a hard task, since it is sometimes impossible to predict "what" information these programs will generate and "where" it will be recorded. Applications often keep temporary data (e.g., for caching) and permanent data (e.g., for configuration) using files stored on different locations of the disk, with the OS recording traces about the programs and files used in some protected structures (such as the Windows Registry). A general approach to manage such cases is to encapsulate the involved applications in order to avoid information leakage. Data can be encapsulated using different methods, as discussed below.

1) External Local Disk: Applications and data can be maintained on an external local disk, such as an USB flash drive. A portable version of the software can be used in order to avoid the use of the main system drive to store application data. The automation should implement the wiping of the entire external drive, and eventually the copying of plausible data (e.g., multimedia files) onto it. The main drawback of this method is that information about the used applications can already be logged by the OS in its internal structures unless specific precautions are taken, such as launching the *ASSD* procedure from the command prompt of Windows.

2) Remote Disk: The involved data set can reside on a different device which can be accessed remotely. In this case, any traces linking the local system with the remote one should be removed or avoided. These include eventual applications (and respective traces) used to access the remote devices, as well as suspicious connections registered at the ISP side.

3) Virtualization: A virtual machine can be used to contain sensitive applications and data. The main advantages of this method are that the data is maintained locally, thus having the full control of it, and at the same time applications can be executed in an environment decoupled from the main system. In general, the main OS records information about the virtualization software but nothing regarding applications and data of the hosted OS. The presence of a virtualization software on a system cannot be considered suspicious, since the use of virtual machines nowadays is a common practice. The wiping process should only involve the cleaning of the files containing the virtual machine. An ad-hoc procedure could consider rewriting these files by copying data of another virtual machine onto them.

4) Disk Encryption: A complex data set could also be maintained in an encrypted disk or partition. This method can be used in conjunction with virtualization in order to improve the hiding of sensitive data. Furthermore, this method can be useful when it is not possible to run the entire wiping process (e.g., due to very strict time constraints). In fact, without the appropriate key, even a deep forensics analysis cannot recover the encrypted information. However, traces left by software used to manage encrypted disks [11] might raise considerable suspicions, and in some Countries are illegal.

IV. Case Study

Experiments on various systems, with different storage technologies, OSes and filesystems, have been and are being conducted in order to demonstrate that the automatic, selective and secure deletion of data can be performed and does not require advanced skills. In this section, an experimental study, implementing the basic case (see Section III-B), is dealt with. Experiments have been conducted on a sample hardware platform with CPU Intel Core I5-480M, RAM 4GB DDR3, hard disk SATA 640GB 5400RPM, running Microsoft Windows 7 with Service Pack 1 and NTFS filesystem.

The procedure has been implemented using the Java language, showing that a detailed knowledge of programming is not necessary in order to create an automation meeting all the requested features. Moreover, implementing the self-deletion functionality is quite easy since a Java executable, being indirectly executed (i.e., interpreted) by the Java Virtual Machine (JVM), can modify its physical image at runtime.

A. Wiping

An ad-hoc procedure has been implemented to wipe unwanted data from the disk. It performs a single rewriting of the physical locations occupied by a file using one of the overwriting methods discussed in Section III-D. Subsequently, it calls the standard deletion function of the OS which unlinks the file. However, this procedure can be straightforwardly replaced by any other wiping technique, i.e., performing more rewritings or overwriting files with different patterns.

The wiping procedure implemented for this case study is based on the `RandomAccessFile` class provided by the `java.io` package included in the JDK, which allows a file to be opened with the random access method. As discussed in Section III-C, this operation prevents that blocks of a file are moved onto different disk locations when modifying it. In the following code snippet, it is shown how is possible to use the `SecureRandom` class, from the `java.security` package, to overwrite a file using a secure pseudo-random pattern. A proprietary PNRG algorithm based on SHA-1, compliant with the FIPS 140-2 [12] requirements, has been chosen to accomplish this task. Since hard disks are block devices, they only supports reading or writing a whole physical block at a time. In general, the logical block size in file systems is a multiple of the physical block size. Thus, in the code listed below, writings are performed `BLOCKSIZE`-by-`BLOCKSIZE`, where `BLOCKSIZE` is 512 byte as the NTFS sector size.

```
RandomAccessFile randomAccessFile = new
    RandomAccessFile(file, "rw");
SecureRandom PRNG = SecureRandom.getInstance("
    SHA1PRNG");
byte[] buffer = new byte[BLOCKSIZE];
...
for( i = 0; i < fileSize; i += BLOCKSIZE ){
    ...
   PRNG.nextBytes(buffer);
   randomAccessFile.write(buffer);
    ...
}
...
file.delete();
```

In the experiments, data files have been overwritten with values obtained by means of the Randomization method, whose implementation has been presented above. The automation, named `HelloWorld.class`, has been overwritten by using the Code Mutation method, which transforms it in a program that prints the string "Hello World" on the standard output.

B. Automation and Self-Deletion

The automation of this case study has been implemented using the Java language and needs to be executed by a JVM, which generally does not lock the executing class files. After removing the predetermined list of files, the automation can finally remove itself from the disk using the same wiping procedure shown in Section IV-A.

A batch (`.bat`) script has been prepared in order to implement the "one-click" deletion semantic. It maintains the list of files to be removed, including itself. When executed, it passes that list to the automation, which performs all the required actions. It is important to note that starting the deletion process by means of a batch script makes it possible to execute the core procedure through the `cmd.exe` shell. As shown in Section IV-C, this approach avoids traces about the automation recorded into the Registry.

C. Precautions

Some evidence can be classified as *hard-to-remove*. This includes information about the automation recorded in Registry keys and other system structures, whose files are often locked by the OS and are difficult to remove whilst it is in use. In the experiments carried out, some precautions have been taken in order to minimize these traces:

- *Prefetcher*: The Prefetch mechanism of Microsoft Windows, which stores information about recently used programs onto the filesystem in order to speed up their execution, has been been disabled.
- *Virtual Memory*: The Virtual Memory, which extends the main memory by means of the storage media, has been disabled by setting its size to zero.
- *Registry*: It has been avoided that the OS stores meaningful information about the automation into the Registry, since it is not executed through the file manager but through the shell (`cmd.exe`).

A more detailed explanation of techniques adopted to avoid unwanted traces related to the automation is presented in [2].

Even though the content of the files passed in input to the automation is completely removed and unrecoverable from the media, some metadata related to such files may remain on the system. It includes entries in the MFT containing information such as name, creation time, modification time, access time, size etc.. Moreover, a log of the operations (openings, writings, truncations, renamings etc.) performed onto the files also remains in the filesystem journal. Due to these considerations, it is crucial that the files to be removed, including the automation itself, do not have suspicious names. In the conducted experiments, data files have been

TABLE I
PERFORMANCE OF THE IMPLEMENTED *ASSD* PROCESS USING THE FOUR OVERWRITING METHODS

Method	Average units of time
Naive	1
Randomization	1.6
Replication (random)	>250
Replication (sequential)	1.7
Code Mutation	1

TABLE II
PERFORMANCE OF THE RANDOMIZATION METHOD ON DIFFERENT DATA SETS

File size	Average time
1 MB	131 ms
10 MB	682 ms
100 MB	4946 ms
1000 MB	36523 ms

named `songs_2001.doc`, `songs_2002.doc`, ..., `songs_2010.doc`, while the automation `HelloWorld.class`.

D. Testing and Performances

The proper functioning of the secure deletion has been properly verified by the authors. In particular, DiskView 2.4 [13] and WinHex 16.0 [14] were used to verify that the physical sectors of the deleted files were overwritten, thus being unrecoverable. Subsequently, forensics tools included in the DEFT [15] distribution were used to perform a post-mortem digital forensics analysis on the data and metadata present on the disk. The result confirmed that no meaningful information remains on the system, with it thus being impossible to carve the removed files.

The time taken by the automation to complete the secure deletion of a specified data set is a crucial requirement in some situations. For example, a dissident subject to a sudden inspection by the government authorities should delete any traces of his "illegal" activities in a few seconds.

It is important to note that performance of any *ASSD* procedures is strictly connected to the transfer rate of the storage media. For these reasons, a deep analysis of the performance obtained by the implemented prototype is out of the scope of this article. The only operation introducing a considerable overhead during the *ASSD* process is the overwriting procedure.

Several tests have been performed, each one passing a single file with different size to the implemented prototype. It has been measured that the Naive method is only influenced by the drive transfer rate, due to it not requiring additional computations. On the sample hardware/software configuration, the Randomization method is about 1.6 times slower than the first one, with it having a considerable overhead introduced by the pseudo-random number generation. Performances of the Data Replication method depend on the strategy implemented to select the data source: `seeking` to random locations of the disk at each block rewriting is very slow, with it being more than 250 times slower than the Naive method. Use of sequential locations considerably increases performances (due to the caching mechanisms of the drive), which are similar to those measured for the Randomization method. As expected, performances of the Code Mutation method are the best ones because the data to be written are previously loaded into the memory and a complete overwrite of the entire file is not required. The results are summarized in Tab. I.
Table II shows the average performances of the implemented prototype, which uses the Randomization method, measured on different settings, depending on the size of the file to be deleted. It is important to highlight that these time values are strictly related to the hardware platform used for the experiments.

ACKNOWLEDGMENTS

The authors would like to thank Mario Ianulardo for many and long interesting discussions on the false digital alibi, and especially on its legal implications. A special thanks goes to Mattia Epifani for his encouragement and useful exchange of ideas.

REFERENCES

[1] J. Glanz and J. Markoff, "U.S. Underwrites Internet Detour Around Censors," *New York Times*, June 12, 2011.
[2] A. De Santis, A. Castiglione, G. Cattaneo, G. De Maio, and M. Ianulardo, "Automated Construction of a False Digital Alibi." A M. Tjoa et al. (Eds.): ARES 2011, Lecture Notes in Computer Science 6908, pp. 359–373.
[3] B. Lee, K. Son, D. Won, and S. Kim, "Secure Data Deletion for USB Flash Memory," *J. Inf. Sci. Eng.*, vol. 27, no. 3, pp. 933–952, 2011.
[4] D.M. Dhamdhere, *Operating Systems: A Concept-based Approach*. McGraw-Hill Companies, January 2008.
[5] P. Gutmann, "Secure Deletion of Data from Magnetic and Solid-State Memory," in *Proceedings of the 6th conference on USENIX Security Symposium, Focusing on Applications of Cryptography - Volume 6*. Berkeley, CA, USA: USENIX Association, 1996, pp. 77–89.
[6] P. Gutmann, "Data Remanence in Semiconductor Devices," in *Proceedings of the 10th conference on USENIX Security Symposium - Volume 10*. Berkeley, CA, USA: USENIX Association, 2001, pp. 4–4.
[7] *DoD Directive 5220.22, National Industrial Security Program (NISP)*, U.S. Department of Defense, February 2010.
[8] *NIST Special Publication 800-88: Guidelines for Media Sanitization*, pp. 7, National Institute of Standards and Technology, September 2006.
[9] C. Wright, D. Kleiman, and S. Sundhar R. S., "Overwriting Hard Drive Data: The Great Wiping Controversy," in *ICISS*, ser. Lecture Notes in Computer Science, R. Sekar and A. K. Pujari, Eds., vol. 5352. Springer, 2008, pp. 243–257.
[10] *NTFS Technical Reference*, Microsoft Corporation, March 2003. [Online]. Available: http://technet.microsoft.com/en-us/library/cc758691(WS.10).aspx
[11] S. Lim, J. Park, K. Lim, C. Lee, and S. Lee, "Forensic Artifacts Left by Virtual Disk Encryption Tools," in *Human-Centric Computing (HumanCom), 2010 3rd International Conference on*, August 2010, pp. 1–6.
[12] Federal Information Processing Standards Publication, "FIPS 140-2, Security Requirements for Cryptographic Modules," section 4.9.1, December 2002.
[13] B. Cogswell, "DiskView v2.4," Windows Sysinternals. [Online]. Available: http://technet.microsoft.com/en-us/sysinternals/bb896650
[14] "WinHex 16.0," X-Ways Software Technology AG. [Online]. Available: http://www.winhex.com/winhex/
[15] "DEFT Linux 6.1." [Online]. Available: http://www.deftlinux.net/