The Journal of Systems and Software 83 (2010) 1813–1822

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Security and privacy issues in the Portable Document Format☆

Aniello Castiglione[a,*], Alfredo De Santis[a], Claudio Soriente[a,b]

[a] Dipartimento di Informatica ed Applicazioni, Università di Salerno, Via Ponte don Melillo 1, I-84084 Fisciano (Salerno), Italy
[b] Facultad de Informática, Universidad Politécnica de Madrid, Avenida de Montepríncipe, 28660 Boadilla Del Monte (Madrid), Spain

ARTICLE INFO

Article history:
Received 11 April 2008
Received in revised form 22 March 2010
Accepted 24 April 2010
Available online 15 June 2010

Keywords:
Compound document format
Document security
Electronic document
Information leakage
Portable Document Format (PDF)
Privacy
Security
Information forensics
Digital forensics
Digital investigations

ABSTRACT

The Portable Document Format (PDF) was developed by Adobe in the early nineties and today it is the de-facto standard for electronic document exchange. It allows reliable reproductions of published materials on any platform and it is used by many governmental and educational institutions, as well as companies and individuals. PDF documents are also credited with being more secure than other document formats such as Microsoft Compound Document File Format or Rich Text Format.

This paper investigates the Portable Document Format and shows that it is not immune from some privacy related issues that affect other popular document formats. From a PDF document, it is possible to retrieve any text or object previously deleted or modified, extract user information and perform some actions that may be used to violate user privacy. There are several applications of such an issue. One of them is relevant to the scientific community and it pertains to the ability to overcome the blind review process of a paper, revealing information related to the anonymous referee (e.g., the IP address of the referee).

1. Introduction

The number of files that are published and exchanged through the Internet is constantly growing and electronic document exchange is becoming more and more popular among Internet users. The diversity of platforms, formats and applications has called for a common technology to overcome those differences and produce universally readable documents to be exchanged without limitations. Even though it is supported by nearly every application on any machine, plain text ASCII has failed to become popular because it does not allow text formatting, image embedding and other features that are key to an efficient communication.

The Portable Document Format was developed by Adobe Systems Inc. to solve heterogeneity issues in electronic document exchanges. It is a cross platform page description language that has become the de-facto standard in document publishing and exchange. According to Adobe, more than one billion PDF documents (Adobe Systems Inc., 2009a) are published using the Portable Document Format and a large number of organizations, institutions and companies rely on it to exchange and distribute their official documents. The format has been constantly supported and updated from its first version in 1991, until the last released version 1.7 which became a standard in July 2008, under the name ISO 32000 (AIIM, 2009). The ECM – *Enterprise Content Management* (formerly known as AIIM – *Association for Information and Image Management*) is the leading international non-profit organization focused on helping users to understand the challenges associated with managing documents, contents, records, and business processes. The AIIM/ECM also holds the Secretariat for the ISO (*International Organization for Standardization*) Technical Committee (TC171) focused on "Information Management Compliance" issues, responsible for the standardization process of the Portable Document Format. They also created a public wiki (PDF Standard Committees, 2010) for the centralized management of the complex tasks among the various Committees involved in the standardization process.

While there are several third-party applications that process PDF files, the three main products developed by Adobe to create, distribute and read those documents are:

- Adobe Acrobat, used to create, arrange and edit PDF documents;
- Adobe Distiller, used to convert documents written in PostScript to PDF;

☆ The research has been partially funded by the Spanish National Science Foundation (MICINN) under grant TIN2007-67353-C02, and by the Madrid Research Council (CAM) under grant S2009TIC-1692. This research has been partially performed when Claudio Soriente was with the University of California, Irvine.

* Corresponding author. Tel.: +39 089 969594; fax: +39 089 969600.
E-mail addresses: castiglione@acm.org, castiglione@ieee.org (A. Castiglione), ads@dia.unisa.it (A. De Santis), csoriente@fi.upm.es (C. Soriente).

0164-1212/$ – see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2010.04.062

- Adobe Reader, used to display PDF documents and allow small amendments like annotations, text highlight, etc. This application is freeware and is probably one of the most popular applications among computer users.

The main advantage of the PDF format is that it allows documents created within any desktop publishing package to be viewed in the original typeset design, regardless of the systems where it is being displayed. Documents with texts, images, hyper-links and other desirable features in document authoring, can be easily created with the packages distributed by Adobe or with any other authoring application (e.g., Microsoft Office, OpenOffice, LaTeX, etc.) and then converted to the PDF format. The result is an easy to distribute, small size document, that will be displayed exactly in the way it was created, on any platform and using any viewer application.

Besides being very flexible and portable, PDF documents are also considered to be *secure*. Popular document formats like Microsoft Compound Document File Format (MCDFF) have been proven to have security flaws that can leak private user information (see Castiglione et al. (2007)), while PDF documents are widely regarded as immune to such problems. This is one of the reasons why many governmental and educational institutions have chosen PDF as their official document standard.

The paper will start giving a concise overview of the PDF format, focusing on how data is stored and managed. It will then show how some features of the PDF format, as well as some design criteria, can be exploited to retrieve data that are not meant to be published, as well as private information of both the document editors and readers. In addition, some tools will be introduced in order to analyze PDF documents and to show the results of the experiments. The paper will also highlight that PDF standards include some features that may be used by a malicious user to violate user privacy or to compromise the security of the system on which the PDF document is opened.

1.1. Organization

The rest of the paper is organized as follows. Section 2 surveys related works while Section 3 analyzes the PDF format. Section 4 shows security and privacy issues related to PDF documents and Section 5 introduces some tools developed by the authors for PDF document analysis. Section 6 points out how the research conducted in the paper may be used in a digital forensics investigation. In Section 7 some security considerations and possible solutions are given. Conclusions are given in Section 8.

2. Related work

Information about the Portable Document Format can be found in (Adobe Systems Inc., 2010a; PDF Standard Committees, 2010). Those documents are the main source of information on how PDF documents are structured and managed by variuos PDF compliant applications.

Even though information leakage in published documents is a well known issue, only a few publications investigate the problem. Byers (Byers, 2004) showed how hidden text can be extracted from Microsoft Word documents. He collected over 100,000 documents and compared the text that appears when each document is opened with Microsoft Word, with the text extracted from the same documents using widely known text extraction tools. Almost each processed document had some hidden contents like previously deleted texts, revisions, etc. Castiglione et al. (2007) conducted a more extensive study on the popular document format, investigating the way Microsoft Compound documents are created and structured. The same authors developed some tools to extract document hidden metadata as well as sensitive publisher information and show how to use the slack space responsible of such threat as a steganographic means.

Several companies and institutions have distributed guidelines to avoid information leakage in published documents after that the media reported news about documents published on the Web containing sensitive information which were not supposed to become public. For example, in May 2005 the Coalition Provisional Authority in Iraq published a PDF document on the "Sgrena-Calipari Incident". Black boxes were used to conceal the names of some people involved in the incident, but all of them were easily revealed copying the text from the original document into a text editor (Wikipedia the Online Encyclopedia, 2009). Several papers discuss the PDF structure (King, 2004; Bagley et al., 2007) and some of them introduce tools for content extraction from PDF documents (Chao and Fan, 2004; Futrelle et al., 2003) or tools to use PDF documents as a steganographic means (Zhong et al., 2007).

To the best of our knowledge, there is no paper addressing security and privacy issues in the Portable Document Format. Otherwise, PDF format (and some of the major products that implement it) is constantly monitored for security flaws and new patches are regularly released on the developer website (Adobe Systems Inc., 2009b). A comprehensive list of patches can be found in the section "*Security bulletins and advisories*" of the Adobe website (Adobe Systems Inc., 2009c). Recently, Adobe, aware of the security issues due to the JavaScript code that can be embedded in a PDF file, have alerted the community to disable JavaScript in order to try to contain the vulnerability (Adobe, 2009) and to update their products to the latest version (Adobe, 2010). Nevertheless, it is surprising how some features of the format that can be used to expose sensitive user information are not regarded as "security flaws" (or at least as a possible problem) and are not addressed by any patch.

3. The Portable Document Format

This section will give a brief overview of the PDF format, highlighting the parts that are relevant to our work.

The Portable Document Format is based on the PostScript (Adobe Systems Inc., 1999) page description language and has been introduced to improve performances and provide some form of user interactivity. A PDF document consists of a collection of objects which together describe the appearance of the document pages. Objects and structural information are all encoded in a single, self contained, sequence of bytes. The structure of a PDF file has four main components:

- a *header* identifying the version of the PDF specification to which the file complies;
- one or more *body* sections containing the objects that constitute the document as it appears to the user;
- one or more *cross-reference tables* storing information and pointers to objects stored in the file;
- one or more *trailers* that provide the location of the cross-reference tables in the file.

A newly created PDF document has only one body section, one cross-reference table and one trailer. When a document is modified, its previous version is not updated, but any changes and new contents are appended to the end of file, adding a new body section, a new section of the cross-reference table and a new trailer. The incremental update avoids rewriting the whole file, resulting in a faster saving process, especially when only small amendments are made to very large files. Objects stored in the body section have an object number used to unambiguously identify the object

Table 1
An example of cross-reference table.

```
xref
0 36
0000000000 65535 f
0000076327 00000 n
0000076478 00000 n
0000076624 00000 n
0000078478 00000 n
0000078629 00000 n
0000078775 00000 n
0000080488 00000 n
0000080639 00000 n
.
.
.
0000100661 00000 n
```

within the file, a non-zero generation number and a list of key-value pairs enclosed between the keywords ⟨*obj*⟩ and ⟨*endobj*⟩. Generation numbers are used only when object numbers are reused, that is, when the object number previously assigned to an object that has been deleted is assigned to a new one. Due to incremental updates, whenever an object is modified, a copy of the object with the latest changes is stored in the file. The newly created copy will have the same object number as the previous one. Thus, several copies of an object can be stored in the file, each one reflecting the modifications made to that object from the time it was created, onwards.

The cross-reference table is composed of several sections and allows random access to file objects. When a document is created, the cross-reference table has only one section and new sections are added every time the file is updated. Each section contains one entry per object, for a contiguous number of objects. An example of cross-reference table section is given in Table 1. As shown, each section starts with the keyword ⟨*xref*⟩ followed by the object number of the first object that has an entry in that section and the number of its entries. Table 1 shows a section with entries of 36 objects, from object 0 to object 35. Each entry provides the following information:

- the object offset in the file;
- the object generation number;
- the *free*/*in-use* flag with value *n* if the object is in use or *f* if the object is free, that is, if the object has been deleted.

Object 0 is a special object and it is always marked as free, with generation number 65535. The latest document ⟨*trailer*⟩ is stored at the end of the file and points to the last section of the cross-reference table. A PDF document is always read from the end (apart when generated with the "Fast Web View" flag enabled), looking for the offset relative to the last section of the cross-reference table, required to identify the objects that constitute the latest version of the document. Each time the document is updated – adding new objects or modifying existing ones – a new body, cross-reference table section and trailer are appended to the file. The body section will contain the newly created objects or the updated version of the existing ones, the cross-reference table section will store information to retrieve those objects, while the trailer will have a reference to the newly created cross-reference table section, as well as a pointer to the previous one.

4. Security and privacy issues

Two main issues are investigated in this paper. The first one is related to how changes made to PDF documents are handled, while the second one concerns PDF interactive features.

4.1. Incremental updates

As introduced in Section 3, to speed up document saving, the PDF standard allows the use of incremental updates. Whenever an object is modified, a new version of the object is created and appended to the PDF file. The old version is kept in the file contents but it will not be parsed when rendering the document (for visualization, printing, etc.). In a similar way, when an object is deleted, it is not removed from the file contents but it is only marked as *deleted* so that the viewer application will skip it when parsing the file. Even though such design choices cause the PDF file to grow in size at every modification, it allows for a fast document saving (King, 2004).

In late December 2002, during one of their meetings, the PDF Archive Committee raised interest on incremental updates, questioning the way changes to embedded objects in a PDF document are handled and suggesting to warn users that "deleted pages are not really gone" (PDF Working Group, 2002). An official note to make users aware of the problem was never released and incremental updates are still part of the PDF specification.

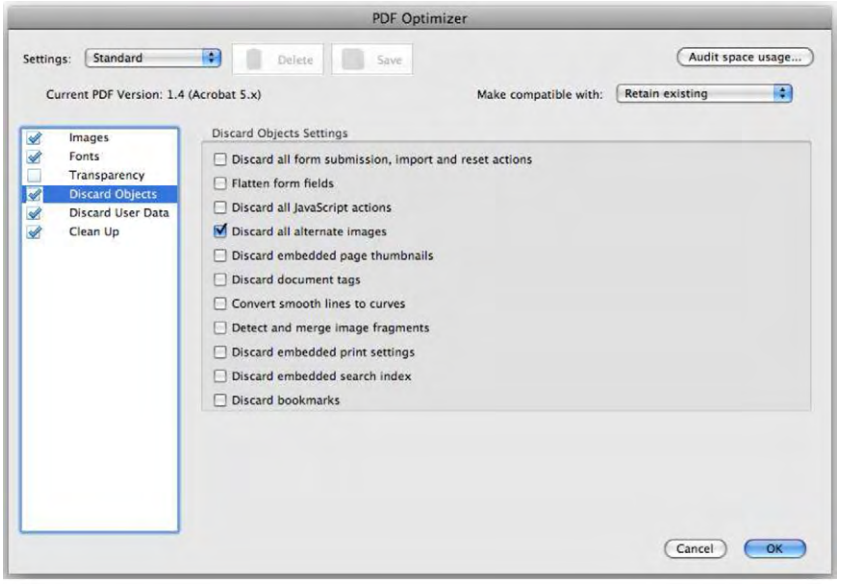
Due to old versions of modified or deleted objects not being deleted from the PDF file contents, but only marked as "not to be processed", it is still possible to retrieve them. Deleted objects can be extracted and visualized while it is possible to display all the versions of a modified object, from its first version to the one actually displayed when rendering the document (for a maximum of 10 levels). Suppose that a company distributes among its managers a PDF document with low confidentiality information, such as the company's layout, as well as sensitive information, such as market strategies and expected balance. Later, the company decides to distribute the document on the Internet deleting all sensitive contents by directly editing that PDF file. As the document will retain deleted objects, it will be possible for anybody to extract those objects from the file and retrieve the company sensitive data.

It is important to mention that, by using the Adobe Acrobat products, it is possible to avoid the problems introduced with the incremental updates by choosing the option "Save As. . ." instead of "Save", where the latter produces an incremental section but the former completely rewrites the PDF file. By choosing the "Save As. . ." option it is possible to set many other parameters which are intended to remove several pieces of information and personal data. Fig. 1 shows the options present in the two sections interesting for cleaning potentially dangerous information. To reach (and set) such options the user should navigate in the menu "File" → "Save As. . ." → "Format" → "Adobe PDF Files, Optimized" → "Settings" of the Adobe products. Unfortunately, the default operation is "Save" and consequently the PDF could be affected by such issue.

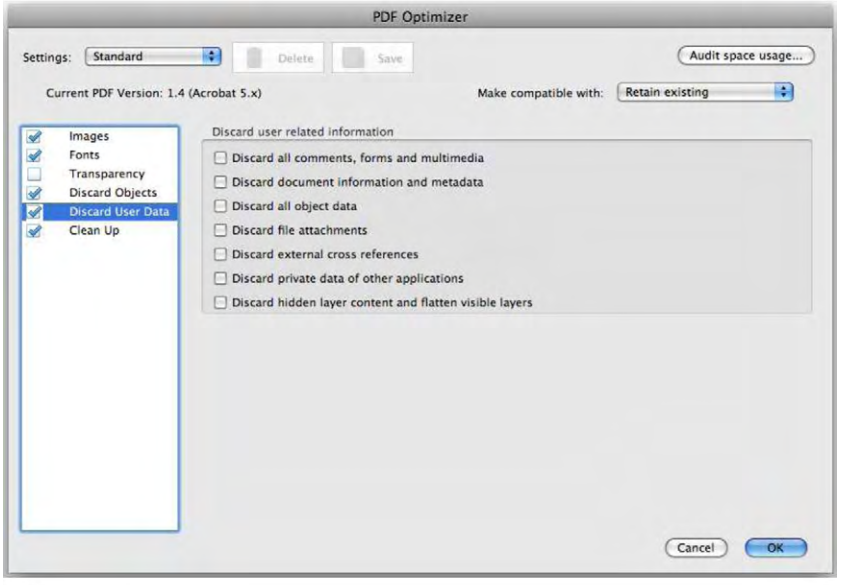
4.2. Interactive features

Adobe PDF specifications include a variety of interactive features that make PDF documents interactive. It is possible to jump to a specific location within the document being displayed or to other documents, run JavaScript code, open URIs or launch applications (see Adobe Acrobat SDK v. 8.1, 2007; Adobe Solutions Network, 2005a,b; Adobe Acrobat 7.0, 2005). Actions are executed in response to several trigger events such as document opening or closing, document printing, etc. Not all the viewer applications implement interactive features and they would not work if the document is opened without one of them. Among applications that process interactive features, not all readers display a dialog to inform the user whenever an action is being executed, with the result that, while reading a PDF document, other programs might be executed or external link might be resolved without user awareness.

In particular, Adobe products warn a user if a PDF tries to connect to an external address. While a user can intentionally dis-



(a)



(b)

Fig. 1. The two panels present in the Adobe products within the "Save As. . ." function which allow a user to remove some potentially dangerous information. (a) First panel and (b) second panel.

able the security prompt, all the tests performed by the authors have been conducted using the default value of this security parameters, which is to check all the connections and ask the user for an authorization. Fig. 2 shows the default values of the configuration that control such security filter. To reach such configuration a user should go through the menu "Preferences" → "Trust Manager" → "Internet Access from PDF Files outside the Web browser" → "Change Settings. . ." → "Manager Internet Access" of the Adobe products.

4.2.1. Interactive features: experimental results and basic attack

The authors have tested the criticality of this issue by preparing an harmful PDF document and sending it to their co-workers and acquaintances asking them to open the document and see if something unusual would happen.

The preparation of this test file has been a very simple task because it consisted in inserting a "trigger" that will perform an "action". This task might be accomplished in a simple way by using the Adobe Acrobat product and such a way has been changed by Adobe during the time. Fig. 3 shows a screenshot of such operation performed by Adobe Acrobat 9.2.0 under Mac OS X. To achieve such a task it is enough to navigate in the menu "Advanced" → "Document Processing" → "Document JavaScripts. . ." and insert the proper code. As an example, the following code performs a test on the version of the PDF reader application (both standalone and browser plug-in) and, depending on such version, opens the website of the EFF, if the version is less than 7, or the FSF website otherwise. The example also shows that the script works well even in presence of a different version of PDF. This is obtained by using the appropriate JavaScript function (i.e., `getURL()` for PDF prior to 1.7 and `launchURL()` for the other

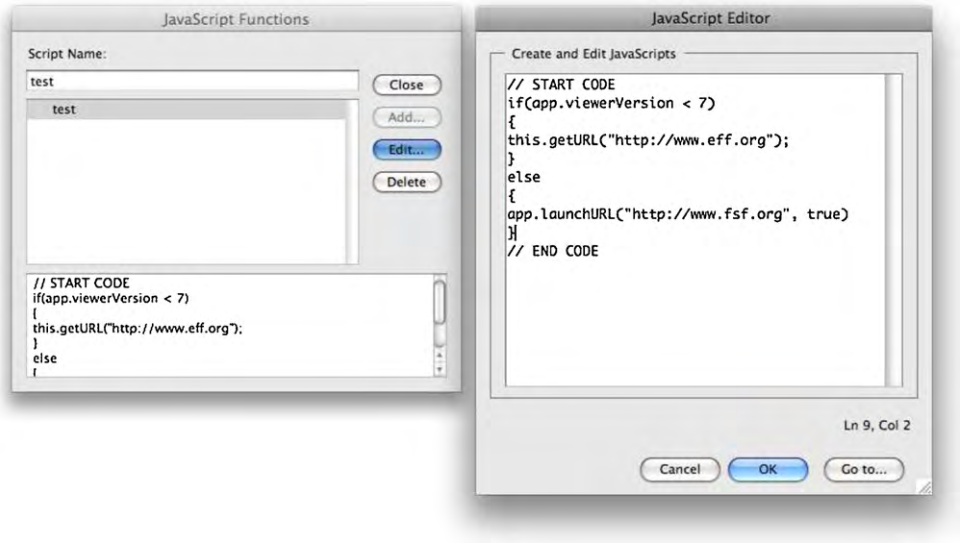


Fig. 3. An example of embedding JavaScript code into a PDF document by using Adobe Acrobat.

Fig. 2. The panel which controls the security warning message of Adobe products (and their default values) resulting when a PDF file attempts to open an external connection.

```
// START CODE
if(app.viewerVersion < 7)
{
this.getURL("http://www.eff.org");
}
else
{
app.launchURL("http://www.fsf.org", true);
}
// END CODE
```

versions).

The people surveyed were also asked to answer a brief questionnaire (that was contained in the PDF document itself) about their Operating System, PDF document reader, etc. On open, the document was intended to connect to a target webpage that had been previously set up. Surprisingly enough, many of the recipients were not warned about any abnormal behaviour by their PDF reader application. Of course, some of them were using PDF readers that do not implement interactive features, but others had their machines connecting to an external link without their consent.

In detail, the PDF test file was sent to over 400 recipients and almost all of them replied to the questionnaire. Recipients were mainly university students, researchers and professors, resulting in a well-educated and technology-savvy participant group. From the experiments, Adobe Acrobat Reader warns the user about an outgoing Internet connection only from version 7.0 or greater (not the initial 7.0 version but all the 7.0.x updated versions). About 75% of the respondents had either Adobe Acrobat Reader version 7.x.x or 8.x.x installed on their machines. Around 20% of those 75% (i.e., 6.66% of the total) opened the document through their Internet browser (they were reading their email through a WebMail client) and had it connected to the website silently. 20% of the respondents opened the PDF document using either Foxit Reader version 2.x (Foxit Software Company, 2010), Evince (The Evince Team, 2009) or KPDF (The KPDF Team, 2008) for Linux or the Mac OS X Preview.app, and no Internet connection was started at all. Surprisingly, the remaining 5% of the respondents had installed a very old PDF reader, i.e., Adobe Acrobat Reader versions 5.x.x and 6.x.x.

As previously mentioned, if the document is opened through a browser, the URI is resolved without user warning and the webpage contained therein is loaded, even if the latest version of Acrobat Reader (ver. 9.3.1) is installed. In fact, with these results in mind, the authors performed a comprehensive test with the major browsers available on the market in order to check their behaviour when dealing with such JavaScript code. The tests have been conducted under Microsoft Windows (XP/Vista/7) and Mac OS X 10.6.2. The results are shown in Table 2 and point out that almost all browsers are still vulnerable to such attack even though all the plug-ins are updated to the latest version (i.e., version 9.3.0 dated 22/12/2009) and an updated antivirus was running on the machines. In greater detail, for each browser, the authors looked for the plug-ins responsible for handling a PDF document (by using the commands resumed in Table 2). The files that implement such

Table 2
Browsers and their vulnerabilities to the PDF interactive features basic attack.

Browser	Windows XP/Vista/7	Mac OS X 1.2	How to check plug-ins	Version
Mozilla Firefox	Vulnerable	Ask action	About:plugins	3.5.8
Opera	Vulnerable	Ask action	About:plugins	10.10
Safari	Vulnerable	Vulnerable	Help, installed plug-ins	4.0.4
Google Chrome	Vulnerable	Ask action	About:plugins	4.0.249 Win; 5.0.307 Mac
Internet Explorer	Vulnerable	N/A	Tools, manage add-ons	8.0.6001

plug-ins are located in:

```
/Library/Internet Plug-Ins/AdobePDFViewer.plugin/
```

for Safari under Mac OS X 10.6.2 and in:

```
C:\Program Files\Common Files\Adobe\Acrobat\ActiveX\AcroIEHelper.dll
C:\Program Files\Common Files\Adobe\Acrobat\ActiveX\AcroIEHelperShim.dll
C:\Program Files\Mozilla Firefox\plugins\nppdf32.dll
```

for Internet Explorer 8.0 (`AcroIEHelper.dll`, `AcroIEHelperShim.dll`) and the other four browsers that all share the same plug-in (the one provided for Mozilla Firefox, `nppdf32.dll`).

For completeness, the authors checked the files responsible of the warning message occurring when opening the test PDF document with the Acrobat products. That files are located in:[3pt]

```
/Applications/Adobe Reader 9/Adobe Reader.app/Contents/Plug-ins/WebLink.acroplugin/
C:\Program Files\Adobe\Reader 9.0\Reader\plug_ins\Weblink.API
```

under Mac OS X 10.6.2 and Windows XP/Vista/7, respectively.

To sum up, regarding the Windows OS family the problem is not a 100% Windows OS flaw or a 100% PDF issue, but can be considered mainly an issue related to the implementation of the PDF plug-in(s) in use by the browsers running the Windows OS family.

Other PDF readers that have been tested did not always resolve the URI embedded in the test PDF document. A more detailed evaluation of the major PDF applications is beyond the scope of this paper.

4.2.2. Interactive features: an alternative attack

As previously discussed, not all the PDF readers can be exploited with the attack presented in the previous section. In fact, either the PDF reader warns the user with a dialog asking to authorize a remote connection, or, a personal firewall may also warn (and block) such outgoing connection. To overcome such a problem, this paper will propose an alternative attack that makes use of DNS resolver of the machine where the PDF document is opened. The key idea is to bind a PDF document to a fake Fully Qualified Domain Name.

These are the sample steps that must be performed in order to implement such attack:

1. create a fake DNS zone on a machine operated by the malicious (and controlling) user and enable the verbose log in that DNS server (at least for the ad-hoc DNS zone) in order to read all the requests that will arrive;
2. choice the PDF document *W* that one would like to "track" and calculate an HMAC (NIST, 2002) of that file:
   `PDFhash=hmac(W);`
3. prepare the JavaScript code to embed inside the malicious PDF file:
   ```
   cDate = util.printd("hhmmssDDMMYYYY", new Date());
   app.LaunchURL("http://" + cDate + "_" +
      PDFhash + "fakezone.evil.com", true);
   ```
   where `cDate` contains a kind of timestamp that is created when the code is executed (i.e., when the document is opened);
4. inject the prepared JavaScript code inside the PDF file in the same way as in subsection 4.2.1;
5. the resulting URL just constructed will be something like: http://18304516092007_a6e49bc20a12326ac590ef.fakezone.evil.com if the PDF has a HMAC of a6e49bc20a12326ac590ef (by using a 20-bytes SHA-1 function for the HMAC calculation) and supposing that it was opened at 18:30:45 of the 16th September 2007.

The `cDate` and `PDFhash` are necessary to generate an always different identifier that will result in an always different DNS query. This is performed in order to avoid the caching name servers and to uniquely identify the PDF file and the time when it was opened.

This attack does not warn the PDF reader and is not detected by nearly the majority of the personal firewalls on the market. In such a way, the DNS resolver capabilities can be used to trace a PDF document and track every single user that opens it. It only requires inserting an action in the document which would connect to a host that logs the IP addresses of incoming connections.

4.2.3. Interactive features: scenarios

Many newspapers that sell their issues over the Internet in PDF format might trace IP addresses to check whether a single copy of an issue is read only by the customer who purchased it or if it has been illegally redistributed over the Internet and it is read by multiple unauthorized users. Instead of distributing the newspaper in PDF format, one customer may share his/her own account. There are several existing methods to check if someone shares his own account with other people. Shortly, the newspaper website administrators can substantially check the number of hits to their service coming from different IP addresses in a given (short) time. An electronic newspaper system can make use of the idea by embedding in each copy of the newspaper a call back action to the newspaper server (or an ad-hoc system) revealing something that uniquely identifies that copy. In such a way, administrators can discover abuses and disable malicious accounts.

Another prominent example comes from the review process of scientific papers submitted for publication (in a journal, a conference or elsewhere), either if the papers are submitted in PDF format or if a PDF file is generated upon the submission of document sources (for example LaTeX sources). To guarantee fairness in the selection criteria, many conference committees endorse a double-blind review process where reviewers and authors keep their identity hidden from each other (Snodgrass, 2007; Wikipedia the Online Encyclopedia, 2010; McKinley, 2008). If the PDF document opens a connection to an host controlled by the authors, they will be able to identify the reviewers and later try to influence their decision. As online review platforms are becoming popular, PDF documents are more likely to be opened through a browser, for example when a paper is opened for the first time from the reviewing system. As stated in Section 4.2.1, even when using an updated version of the Acrobat Reader, when opening a PDF document from a link on a webpage, it does not warn the user of the outgoing connection, letting the malicious PDF connect to the tracking website

Table 3
The `MasterTable.log` generated by the tool *ModPDF*.

```
16: Type = /EmbeddedFile, subType = no type
0 – 0000006735 00000 n

17: Type = /Page, subType = no type
0 – 0000000018 00001 f

18: Type = /Page, subType = no type
0 – 0000000023 00001 f

19: Type = /Pages, subType = no type
0 – 0000010787 00000 n
1 – 0000183745 00000 n

20: Type = no type, subType = no type
0 – 0000010849 00000 n
```

created by the authors of the paper. Moreover, by using the attack proposed in Section 4.2.2, a malicious author would probably result in a more hidden action of tracking his paper. It is worth noting that, even if a PC expert referee could be alerted – either by a connection attempt or by the opening of a webpage without asking for it – such actions, probably, would not alarm the average PC user.

5. Tools

The paper will introduce three tools to analyze PDF documents and to retrieve hidden data and information regarding the past version of a PDF document. They can be downloaded at the project website http://www.dia.unisa.it/research/pdf. The tools, written in C language, are just a simple proof of concept and are not intended to be a full application. The authors tested such tools on more than 35000 PDF files, downloaded from various Internet domains. All the documents are publicly available and their download has been automated using free Web downloader applications in conjunction with popular search engines.

AnalyzePDF analyzes PDF documents and provides information about the number of stored cross-reference table sections, that is the number of times one or more document objects were modified. It takes as input a directory containing PDF documents and produces a log file with the list of the processed PDF documents, their size in bytes and the number of cross-reference table sections found. From the experiments, more that 10% of analyzed documents had two or more cross-reference tables, that means that they were edited at least one time.

Once PDF documents with two or more cross-reference table sections have been identified, a further analysis is performed with *ModPDF*. It takes a single PDF document as input and provides information on all its objects with a list of their changes. *ModPDF* produces three output files:

- `MasterTable.log`
- `ModifiedObjects.log`
- `MetadataList.dat`

`MasterTable.log` lists all the objects stored in a PDF document. For each object, *ModPDF* provides the object number, its type, subtype and a list with one entry for each time the object was modified. Each entry provides the offset where that copy of the object is stored in the file, the object generation number and the *free*/*in-use* flag. Objects which have never been modified will only have one entry. Table 3 shows an example of MasterTable.log. It shows that object 19 was modified once: its original version is stored at offset 10787 while its updated version is stored at offset 183745. Only the latter will be parsed by the PDF reader application during rendering. Objects 16–18 and object 20 were never modified after their creation.

Table 4
An example of the file `MetadataList.dat` generated by the tool *ModPDF*.

```
Session #0
ModDate D:20060210171944+01'00'
CreationDate D:20060210165921+01'00'
Title Indice
Creator QuarkXPress: pictwpstops filter 1.0
Author Gizzo
Producer Acrobat Distiller 6.0.1 for Macintosh

Session #1
ModDate D:20060212173149+01'00'
CreationDate D:20060210165921+01'00'
Title Indice
Creator QuarkXPress: pictwpstops filter 1.0
Author Gizzo
Producer Acrobat Distiller 6.0.1 for Macintosh
```

(a) **Legge Costituzionale n. 1 del 23 gennaio 2001** *"Modifiche agli articoli 56 e 57 della Costituzione concernenti il numero di deputati e senatori in rappresentanza degli italiani all'estero"* **link dal parlamento**

(b) **Legge Costituzionale n. 1 del 23 gennaio 2001** *"Modifiche agli articoli 56 e 57 della Costituzione concernenti il numero di deputati e senatori in rappresentanza degli italiani all'estero"*

Fig. 4. Two different versions of the same paragraph of a PDF document. (a) First version and (b) final version.

`ModifiedObjects.log` provides the same information as the former log file, but only for objects that were modified at least once.

For each update to the PDF document, `MetadataList.dat` lists the following information:

- date/time when modifications were applied to the document;
- date/time when the document was created;
- document title, creator, author, and producer.

Table 4 shows the content of `MetadataList.dat` resulting from the analysis of one of the collected documents with *ModPDF*. It shows that the file was created by user "Gizzo" on a Macintosh. He started editing the file at 4:59 p.m. of the 10th February 2006 and saved his work at 5:19 p.m. on the same day. The document was updated 2 days later and saved at 5:31 p.m.

The third and last tool, *MakeOldPDF* takes as input a PDF document and reconstructs all its previous versions, starting from the original one. While previous tools (*AnalyzePDF* and *ModPDF*) identify objects that have been modified, with *MakeOldPDF* it is possible to recreate the different versions of the document and, of course, the different versions of modified objects. Fig. 4 shows two different versions of the same paragraph within a PDF document. The second one is a noticeable example of the *MakeOldPDF* which produced 13 different versions. Fig. 4(a) shows a paragraph as it appeared when the document was created, while Fig. 4(b) shows the same paragraph as it appears in the final version of the document. After several modifications, the link that originally appeared at the end of the paragraph was removed before publication on the Internet.

6. Information forensics

The issues illustrated in the paper may also be used for a good reason and not only to undermine the security and privacy of users. Nowadays, most of the investigations carried out by Public Prosecutor's Officers or Law Enforcement Agencies entwine with digital information. The simplest type of digital information is "the file" and, among such countless vastness of files, PDF files are the commonest. Thus, the studies performed in this paper may be of great help to investigators and forensic analysts who day by day come across such files.

The way in which PDF files are managed, with the consequential problem of incremental updates (see Section 4.1), and the metadata contained therein, may be used during a digital forensics analysis to look for circumstantial evidence which, linked together with other data, may give useful directions to ascertain the causes of a crime/offence. For example, by using the tools introduced in Section 5, an investigator may extract a lot of information regarding the time a given PDF file has been created/modified, the Operating System running on the machine that produced that document together with the relative application that has been used to create it, the author name of the document which, if not explicitly modified, is the same of the logon name, and other useful information. Furthermore, the possibility to reconstruct all the versions of a given PDF file may be very helpful in the case of a dispute in which there is the eventuality that a file has been deliberately tampered with.

The interactive features shown in Section 4.2 may be used during a digital investigation to root out a criminal who is hiding behind an Internet connection. An investigator should prepare a well-crafted PDF document that in some way should be obtained by the prosecuted person. The PDF file will act as a bait and, depending on what kind of attack will be successful (the one in subsection 4.2.1 or the one in subsection 4.2.2), would provide the investigators with the IP address of the crook together with all the data that a browser can provide such as Operating System, browser type, browser plugins, time-zone, language, etc. (see Electronic Frontier Foundation, 2010; Gemal, 2010 for the latest information on how to acquire a very large quantity of user data just "looking" at the browser's behaviour). It is important to highlight that with the second kind of attack (subsection 4.2.2) an investigator will find out the IP address of the criminal even if the connection if performed by means of a proxy or a chain of proxies because the logged IP address will be the one resulting from the DNS query and not from the HTTP connection. To overcome such attack, a criminal should use a DNS-proxy which, to our knowledge, is not so familiar and simple to find/use.

The information forensics is not the main topic of the paper but the authors would like to encourage the digital forensics community to focus their attention on a more detailed analysis of the file contents and not only on the information enclosed on the filesystem that hosts that file.

7. Security considerations and possible solutions

The scenarios illustrated in Section 4.2.3 are given only as examples. We can go far with our imagination by creating other kinds of attacks trying to embed a malicious executable code into a PDF file[1] that in a given moment (as always triggered by an event) drops an executable code into the filesystem of the machine on which the PDF file was opened. It is like a virus-dropper, popular in the nineties (the MS-DOS age). This kind of attack is possible only with very early versions of Adobe Acrobat (until version 4.x.x) because Adobe released patches to limit the access to the filesystem by a PDF file. This flaw resulted in the proliferation of some worms/virus for PDF files that has been in the wild until few years (see F-Secure Corporation, 2001; US-CERT, 2000; Shankland, 2001; Avira GmbH, 2008). A slightly different kind of attack, similar to the previous one, is to consider the PDF file as a "container" in which private information (i.e., IP address and so on) may be stored. The key idea is that the PDF file itself steals and stores the information to be carried out. When the PDF file travels out from the victim machine, it will leak the stolen information. Like the previous attack, even this cannot be performed due to limitations provided by recent modifications to the PDF standard.

[1] The executable file may be encoded in Base64 and embedded in a section marked as "deleted" in the PDF file in order to not be processed by the PDF reader.

Of course firewalls, NAT devices and anonymizers would prevent such scenarios, but, as the experiments show, many users are not enough concerned with privacy to set up necessary countermeasures. Moreover, while users are suspicious about opening executables or Microsoft Office files, PDF documents are not (yet) regarded as harmful files.

The paper does not give comprehensive solutions to the threats shown by the authors because their main intention is to raise the awareness of the security and privacy issues that may occur by using the PDF standard. Moreover, it is not easy to suggest possible solutions because of the complexity of the problem and due to the wide diffusion of the format. Also, the features that are exploited as possible problems has been introduced to improve the format, and removing such features would not be a good idea. The hope is that the Standardization Committee (AIIM, 2009) will consider such privacy and security issues. This is the direction that the Committee has already taken with of the creation of several type of PDF (PDF Standard Committees, 2010) according to the area of use (PDF/A [*Archive*], PDF/E [*Engineering*], PDF/UA [*Universal Accessibility*], PDF/H [*Healthcare*], PDF/X [*Exchange/Production Printing*], PDF/VT [*Variable & Transactional Printing*]).

It is worth noting that, starting from Acrobat 8, it seems possible to remove sensitive information from a PDF file. This can be achieved by using the "Examine Document..." function in the "Document" menu within the Acrobat application (Adobe Systems Inc., 2010b). Unfortunately, even after having performed such a cleaning operation, the problem still remains unresolved with the resulting file again having the same "functionality" of the input file. The motivations of such behaviour are not clear but it could depend on the fact that the cleaning operation performed by the "Examine Document..." does not inspect in the proper way the PDF document. Other data such as author information and comment references are removed in the right way after the sanitization process performed by the previously described operation. A knowledgable user, aware of the problem, can easily remove (almost) all this unwanted information by using the redaction tool provided by Adobe Acrobat (see Adobe Inc., 2010) and by saving the PDF file with the function "Save As..." in place of "Save". Moreover, the vast majority of PDF files are created and never manipulated, or at most they contain comments and annotations that (usually) do not have security or privacy implications.

It is fair to say that the problem of information leakage with many file formats usually depends on the application that is consuming them that does not behaves sensibly and not depends on the file format itself. The same considerations also apply to the problem introduced by the execution of JavaScript code, at least for the first kind of attack. In order to avoid the second attack, the Operating System or any firewall – that possibly could be in use on the Operating System where the PDF consumer's application runs – should be in charge of the protection.

Despite the statistics resulting from the conducted experiments, which show that only few users have encountered the problem of the automatic webpage opening, it is important to recall that today most of the users, open a PDF file during a Web surfing and in consequence they are mostly still vulnerable to such issues.

To successfully contain, and possibly avoid, problems related to the review process of scientific papers (see Section 4.2.3), the online Conferencing Systems should take measures to control (and eventually "clean") the content of a PDF file upon submission or just after the automatic compilation from the source (for example in presence of LaTeX sources). This is the direction undertaken by some major scientific publishers (i.e., the IEEE PDF-eXpress system at IEEE, 2008) which established a service that aims to perform many checks to a paper before the final submission. Unfortunately such checks are only carried out when submitting the final camera-ready version of the paper and not before the review phase, in which

the papers are dispatched to the referees and when anonymity is the key factor to guarantee the fairness of the review process.

Another example of the awareness that a PDF file can be "dangerous" in the review process of a paper is the sanitization operation that the publisher Elsevier has introduced in the EES (Elsevier Editorial System) in which, in order to maintain the reviewer anonymity, the system EES sanitize the reviewer attachment (Elsevier B.V., 2009) by removing personal information from Microsoft Office and PDF files uploaded by Reviewers with their reviews. While, on one hand, this method sanitizes the file(s) containing the review (i.e., the Reviewers comments), on the other hand it does not sanitize the file containing the manuscript uploaded by the authors. It is worth noting that, at least for the Journal of Systems and Software, the EES does not allow authors to upload a PDF file but only the source files (i.e., the LaTeX sources) and as a consequence it is immune from such kind of attacks.

8. Conclusions

Portable Document Format is the most popular standard for document exchange over the Internet. Besides its portability, it provides its own security features such as encryption and digital signature and it is regarded as a secure document standard if compared to other popular document file formats. The paper discusses that the reputation of *secure* format for the PDF documents is not completely right and shows that the standard is not immune to some privacy issues that affect its competitors. Given a PDF file, it is possible to retrieve a previous version of the document and display information not meant to be published. The authors developed some simple tools that can provide sensitive document information as well as any version of the document, from its creation to its most up-to-date version. Another security issue, is the possibility to trigger events when a PDF document is opened, printed, etc. The experiments showed that it is possible to trace the reader of a PDF document any time it is accessed, provided that the device where the file is read is connected to the Internet. The authors also give some cues on how to use their findings in a digital forensics investigation.

The authors consider such threats of great interest to the whole community due to the large use of electronic documents in today communications. In particular, they think that the scientific community have to consider such issues because it might compromise the fairness of the review process of scientific papers for journal or conference publication.

Acknowledgments

The authors wish to thank people who participated in the survey. They also wish to thank Vincenzo Spina and Antonio Fiorillo for their valuable discussions and help. A special thank goes to the JSS anonymous reviewers for their beneficial comments and directions.

References

Adobe, 2009. Security Advisory for Adobe Reader and Acrobat. http://www.adobe.com/support/security/advisories/apsa09-07.html (December 15 2009).

Adobe, 2010. Security Updates Available for Adobe Reader and Acrobat. http://www.adobe.com/support/security/bulletins/apsb10-07.html (February 17 2010).

Adobe Acrobat 7.0, 2005. Acrobat JavaScript Scripting Reference. http://partners.adobe.com/public/developer/en/acrobat/sdk/AcroJS.pdf (June 2005).

Adobe Acrobat SDK v. 8.1, 2007. JavaScript for Acrobat API Reference. http://www.adobe.com/devnet/acrobat/pdfs/js_api_reference.pdf (April 2007).

Adobe Inc., 2010. Adobe 9 Pro Extended, Removing sensitive content. http://help.adobe.com/en_US/Acrobat/9.0/3D/WS4E397D8A-B438-4b93-BB5F-E3161811C9C0.w.html (February 2010).

Adobe Solutions Network, 2005. Acrobat JavaScript Scripting Guide. http://partners.adobe.com/public/developer/en/acrobat/sdk/pdf/javascript/AcroJSGuide.pdf (September 2005).

Adobe Solutions Network, 2005. Adobe Acrobat 7.0: PDF Open Parameters. http://partners.adobe.com/public/developer/en/acrobat/PDFOpenParameters.pdf (July 2005).

Adobe Systems Inc., 1999. PostScript Language Reference, Third Edition. http://partners.adobe.com/public/developer/ps/index_specs.html (February 1999).

Adobe Systems Inc., 2009. Adobe Portable Document Format. http://www.adobe.com/products/acrobat/adobepdf.html (Last updated December 2009).

Adobe Systems Inc., 2009. Latest Product Updates. http://www.adobe.com/downloads/updates/ (Last updated December 2009).

Adobe Systems Inc., 2009. Security bulletins and advisories. http://www.adobe.com/support/security/ (Last updated December 2009).

Adobe Systems Inc., 2010. Adobe PDF Reference Archives. http://www.adobe.com/devnet/pdf/pdf_reference_archive.html (Last updated January 2010).

Adobe Systems Inc., 2010. Examine a PDF for hidden content. http://help.adobe.com/en_US/Acrobat/8.0/Professional/help.html?content=WS7E9FA147-10E3-4391-9CB6-6E44FBDA8856.html.

AIIM, 2009. PDF Reference Bibliography. http://www.aiim.org/standards/article.aspx?ID=33223 (Last updated December 2009).

Avira GmbH, 2008. Avira issues a warning about polymorphous harmful PDFs. http://www.avira.com/en/security_news/polymorphous_harmful_pdfs.html (November 2008).

Bagley, S.R., Brailsford, D.F., Ollis, J.A., 2007. Extracting reusable document components for variable data printing. In: DocEng'07: Proceedings of the ACM Symposium on Document Engineering. ACM Press, New York, NY, USA, pp. 44–52.

Byers, S., 2004. Information leakage caused by hidden data in published documents. IEEE Security and Privacy 2 (2), 23–27.

Castiglione, A., De Santis, A., Soriente, C., 2007. Taking advantages of a disadvantage: digital forensics and steganography using document metadata. Journal of Systems and Software, Elsevier 80, 750–764.

Chao, H., Fan, J., 2004. Layout and Content Extraction for PDF Documents. Lecture Notes in Computer Science LNCS 3163, 213–224.

Electronic Frontier Foundation, 2010. How Unique – and Trackable – Is Your Browser? http://panopticlick.eff.org/ (January 2010).

Elsevier B.V., 2009. Customer Support, Reviewer Attachments Not Sanitised by EES. http://epsupport.elsevier.com/al/12/1/article.aspx?aid=2090 tab=browsebt=4n (December 2009).

Foxit Software Company, 2010. Foxit Reader. http://www.foxitsoftware.com (February 2010).

F-Secure Corporation, 2001. F-Secure Virus Descriptions: PDF Worm. http://www.f-secure.com/v-descs/pdf.shtml (August 2001).

Futrelle, R.P., Shao, M., Cieslik, C., Grimes, A.E., 2003. Extraction, layout analysis and classification of diagrams in PDF documents. In: ICDAR'03: Proceedings of the Seventh International Conference on Document Analysis and Recognition. IEEE Computer Society, Washington, DC, USA, pp. 1007–1013.

Gemal, H., 2010. Browser Spy. http://www.browserspy.dk/ (January 2010).

King, J.C., 2004. A format design case study: PDF. In: HYPERTEXT'04: Proceedings of the Fifteenth ACM Conference on Hypertext and Hypermedia. ACM Press, New York, NY, USA, pp. 95–97.

McKinley, K.S., 2008. Improving Publication Quality by Reducing Bias with Double-Blind Reviewing and Author Response, ACM SIGPLAN Notices. http://www.cs.utexas.edu/users/mckinley/notes/blind.html (August 2008).

National Institute of Standards and Technology (NIST), 2002. The Keyed-Hash Message Authentication Code (HMAC) (FIPS PUB 198). http://csrc.nist.gov/publications/fips/fips198/fips-198a.pdf (March 2002).

PDF Standard Committees, 2010. PDF Standards Implementations Wiki. http://pdf.editme.com/ (Last updated February 2010).

PDF Working Group, 2002. PDF-Archive Draft Meeting Minutes. http://www.aiim.org/documents/standards/pdf-a2003-001_dec_min.pdf (December 2002).

Shankland, S., 2001. New virus travels in PDF files. http://www.news.com/New+virus+travels+in+PDF+files/2100-1001_3-271267.html (August 2001).

Snodgrass, R.T., 2007. Editorial: single versus double-blind reviewing. ACM Transactions on Database Systems 32 (1), 1.

The Evince Team, 2009. Evince—Simply a Document Viewer. http://www.gnome.org/projects/evince/ (Last updated September 2009).

The Institute of Electrical and Electronics Engineers, 2008. IEEE PDF-eXpress. http://www.pdf-express.org/ (August 2008).

The KPDF Team, 2008. KPDF Reader. http://kpdf.kde.org/ (Last updated August 2008).

United States Computer Emergency Readiness Team (US-CERT), 2000. Vulnerability Note VU#31554. http://www.kb.cert.org/vuls/id/31554 (November 2000).

Wikipedia the Online Encyclopedia, 2009. The Calipari Incident. http://en.wikipedia.org/wiki/Nicola_Calipari/, http://en.wikipedia.org/wiki/Rescue_of_Giuliana_Sgrena/ (Last updated December 2009).

Wikipedia the Online Encyclopedia, 2010. Peer Review Process. http://en.wikipedia.org/wiki/Peer_review (Last updated January 2010).

Zhong, S., Cheng, X., Chen, T., 2007. Data hiding in a kind of PDF texts for secret communication. International Journal of Network Security 4 (1), 17–26.

Aniello Castiglione joined the Dipartimento di Informatica ed Applicazioni "R. M. Capocelli" of Università di Salerno in February 2006. He received a degree in Com-

puter Science and his Ph.D. in Computer Science from the same university. He is a reviewer for several international journals (Elsevier, Hindawi, IEEE, Springer) and he has been a member of international conference committees. He is a Member of various associations, including: IEEE (Institute of Electrical and Electronics Engineers), of ACM (Association for Computing Machinery), of IEEE Computer Society, of IEEE Communications Society, of GRIN (Gruppo di Informatica) and of IISFA (International Information System Forensics Association, Italian Chapter). He is a Fellow of FSF (Free Software Foundation) as well as FSFE (Free Software Foundation Europe). For many years, he has been involved in forensic investigations, collaborating with several Law Enforcement agencies as a consultant. His research interests include Data Security, Communication Networks, Digital Forensics, Computer Forensics, Security and Privacy, Security Standards and Cryptography.

Alfredo De Santis received a degree in Computer Science (cum laude) from the Università di Salerno in 1983. Since 1984, he has been with the Dipartimento di Informatica ed Applicazioni of the Università di Salerno. Since 1990 he is 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, Università di Salerno. From November 1996 to October 2003 he was the Chairman of the PhD Program in Computer Science at the Università di Salerno. From September 1987 to February 1990 he was a Visiting Scientist at IBM T. J. Watson Research Center, Yorktown Heights, New York. He spent August 1994 at the International Computer Science Institute (ICSI), Berkeley CA, USA, as a Visiting Scientist. From November 2009 he is in the Board of Directors of Consortium GARR (the Italian Academic & Research Network). His research interests include Algorithms, Data Security, Cryptography, Information Forensics, Communication Networks, Information Theory, and Data Compression.

Claudio Soriente received a Ph.D. in Computer Science from the Università di Salerno and a Ph.D. in Networked Systems from University of California, Irvine. He is currently a post-doctoral researcher at the Facultad de Informática of the Universidad Politécnica de Madrid. His research interests include Security of Distributed Systems, and Wireless Sensor Networks.