OpenOffice Security and Viral Risk

Eric Filiol and **Jean-Paul Fizaine**
French Army Signals Academy (ESAT)
Virology and Cryptology Lab
efiliol@esat.terre.defense.gouv.fr

Documents malware essentially exist for Microsoft Office: the sadly known macro-viruses which still represent a numeric nuisance nowadays. The recent evolution of office suite towards free software – thus providing a high compatibility with existing office software – makes very necessary to determine and evaluate the exact level of risk of the OpenOffice suite with respect to document malware, This paper present an up to date evaluation of its security (release 2.2.x) based on the results established during the summer 2006. Worrying security weaknesses have been identified since. They may still be used by malware to spread through innocuous-looking documents by exploiting the feeling of trust based on encryption and digital signature. However, this study clearly shows the interest of open software with respect to security evaluation.
Finally this paper will discuss the pro and cons of both open and proprietary solutions, on a purely technical basis, as far as security is concerned. There is no such thing as perfect solution. There lies all the complexity of doing computer security.

1.- Introduction

Since two years, the OpenOffice suite is positioning as an open and free alternative solution to the existing commercial counterparts. Equipped with a lot of sophisticated functionalities, it now represents a credible solution of high quality for users. Since version 2.x.x, a rich evolution enables to have an ergonomics environment at user's disposal. Several development environments are available as well as dedicated tools that greatly enhance the overall functional quality. But the existence of such environment and capabilities raises the problem of the OpenOffice security with respect to malware. The existence of macros, as in Microsoft Office, strongly suggests that the malware hazard is not negligible at all.

From June 2005 to July 2006, a deep study of OpenOffice environment has been conducted (release 2.0.2 and 2.0.3 under Linux, Mac OS X and Windows) in the Virology and Cryptology lab at the French Army Signals Academy. A first synthesis of the results has been published in [4] in July 2006. This study has been technically validated by some *proof-of-concept* codes. But most of the most worrying results were only evoked in [4] and have been published in French only in September 2006 [7]. This paper presents those very last technical results, for the first time in English. They have been updated since on t he last OpenOffice release (2.2.x). At the time of publication of [4] a lot of stupid comments have been made about this study, It is the occasion to dispassionately take stock of OpenOffice security and to enable anyone to check on a reproducible basis what can be claimed and what cannot.

The aspects presented here essentially refer to macro security and to the OpenDocument format with respect to encryption and digital signature built-in capabilities. They represent in their own most of the OpenOffice security issues we have identified. The proof-of-concept codes will not be presented. They are useless to the understanding of the present paper. Due to lack of space, we have limited the number of technical details (file dumps in particular). However, they are all available upon request for IT security professionals only.

Aside ours [4], there are only a few studies about OpenOffice security, regardless of the problem of software flaws. Let us mention the main ones:

- in 2003, Rautiainen [5] has presented a short analysis of macros with respect to OpenOffice versions 1.x;
- in june 2006, Kaspersky claimed to have detected the first OpenOffice virus, called a StarDust. But no technical evidence enables to support this claim. Later, OpenOffice denied the self-reproducing nature of the malware;
- in may 2007, a multiplatform OpenOffice virus, called *BadBunny* has appeared. This malware seems to be a direct illustration of the risk pointed out by our own work, last year, in particular with respect to some advanced programming languages: Python, Js and Ruby.

- in June 2007, a deep, comparative study on the security of both OpenXML and OpenDocument formats has been published by P. Lagadec [6]. This study is somehow a sequel of [4] with respect to the viral hazard and information leakage in documents.

Apart from the search for software flaws, the security analysis quite never considers the functional evaluation, in other words the core algorithmic choices, nor the formal analysis: flow matrix, state matrix, protocol analysis... This is unfortunately true both for open and proprietary software.

2.- OpenDocument Format Structure

Due to lack of space, we will not recall the structure of the OpenDocument format (ODF for short). For a detailed presentation on this format, the reader will refer to [4]. Let us just precise that an OpenOffice document is in fact a simple compressed ZIP archive. It is thus possible to decompress it and to access and manipulate all the different document components (data, meta-data, macros...) very easily. Our study has been conducted for ODF documents under Linux, Windows and Mac OS. For sake of clarity (use of command lines) and without loss of generality, we give here the results for the Linux part.

2.1 Unprotected document

Among all these component, the most important one, in terms of security issues is undoubtly the *manifest.xml* file located in the *META-INF* directory of the ZIP archive. This file totally describes the document ODF structure and all the data which are essentially relevant to the different security functionalities: macros, encryption, digital signature.... The other files are:

- content.xml file: this file is present in every OO document and simply contain the visible part of the document;
- the meta.xml file contains all the document meta-information (author's data, access data...);
- the styles.xml file contains the document formatting option;
- the setting.xml file contains all the document configuration data (window size, printing parameters...).

Whenever one or more macros are used, a new directory is created as shown hereafter:

```
./Basic:
total 8
drwxr-x-rx   4 lrv  lrv  138 Mar  2 01:47 Standard
-rw-r--r--   1 lrv  lrv  338 Mar  2 00:38 script-lc.xml
./Basic/Standard:
total 16
-rw-r--r--  1 lrv  lrv   350 Mar  2 00:38 script-lb.xml
-rw-r--r--  1 lrv  lrv  2049 Mar  2 00:38 a_macro.xml
./META-INF:
total 8
-rw-r--r--  1 lrv lrv  1465 Mar   2 00:38 manifest.xml
```

This directory called *Basic* (without loss of generality we will consider macros written in the default scripting language *OOBasic*) and contains all the macros file tree of the document. In addition, the *manifext.xml* file has been modified accordingly to record the macros and their data (access path in particular).

```
<manifest:file-entry manifest:media-type="text/xml"
 manifest:full-path="Basic/Standard/a_macro.xml"/>
<manifest:file-entry manifest:media-type="text/xml"
 manifest:full-path="Basic/Standard/script-lb.xml"/>
<manifest:file-entry manifest:media-type=""
 manifest:full-path="Basic/Standard/"/>
<manifest:file-entry manifest:media-type="text/xml"
 manifest:full-path="Basic/script-lc.xml"/>
```

There exists a lot of possible scripting language to develop OpenOffice macros: OOBasic, JS, Python, Ruby....Whatever maybe the language used, the general management scheme remains the same. Moreover, what has been presented for a single macro still holds for complete libraries of macros [4].

2.2 Encrypted document.

Whenever a user applies a document password, the document is encrypted. Let us consider a macro containing a single macro. All encryption technical data are included as properties within XML tags. The encryption algorithm is Blowfish in CFB mode, the keys are derived from the PBKDF2 key management protocol while the hashing algorithm is SHA1.

As an ODF file is in fact a ZIP archive, it is necessary to define which files in the archive are encrypted or not. To see where the encryption takes place, let us compare the manifest.xml file of the r*eference_file_encrypt.odt* (encrypted) file and the *reference_file.odt* (unencrypted) file respectively. Here follows the dump (excerpt) of the diff command between the two files:

```
< <manifest:file-entry manifest:media-type="text/xml" manifest:full-path="content.xml"/>
< <manifest:file-entry manifest:media-type="text/xml" manifest:full- path="Basic/Standard/HelloWord.xml"/>
< <manifest:file-entry manifest:media-type="text/xml" manifest:full-path="Basic/Standard/script-lb.xml"/>
---------
>  <manifest:file-entry manifest:media-type="text/xml" manifest:full-path="content.xml" manifest:size="2626">
>   <manifest:encryption-data manifest:checksum-type="SHA1/1K"
manifest:checksum="ITmRG2GO+QEChZSdWuHnELeNmoU=">
>    <manifest:algorithm manifest:algorithm-name="Blowfish CFB" manifest:initialisation-vector="UnteGYIbs8Q="/>
>    <manifest:key-derivation manifest:key-derivation-name="PBKDF2" manifest:iteration-count="1024"
manifest:salt="A1jwblqPdaNcWUpdgOF9Kg=="/>
>   </manifest:encryption-data>
>  </manifest:file-entry>
>  <manifest:file-entry manifest:media-type="text/xml" manifest:full-path="Basic/Standard/HelloWord.xml"
manifest:size="339">
>   <manifest:encryption-data manifest:checksum-type="SHA1/1K"
manifest:checksum="zCCMsJxNl78Fzcpe/CnNEHgo4Bs=">
>    <manifest:algorithm manifest:algorithm-name="Blowfish CFB" manifest:initialisation-
vector="auXTuBEHXHQ="/>
>    <manifest:key-derivation manifest:key-derivation-name="PBKDF2" manifest:iteration-count="1024"
manifest:salt="l/McRciiGElm7EIyxQDvRQ=="/>
>   </manifest:encryption-data>
>  </manifest:file-entry>
>  <manifest:file-entry manifest:media-type="text/xml" manifest:full-path="Basic/Standard/script-lb.xml"
manifest:size="350">
>   <manifest:encryption-data manifest:checksum-type="SHA1/1K"
manifest:checksum="kL8H/WhawMbDZeY47uBLZGY30qQ=">
>    <manifest:algorithm manifest:algorithm-name="Blowfish CFB" manifest:initialisation-vector="5Y8OYH/JTkc="/>
>    <manifest:key-derivation manifest:key-derivation-name="PBKDF2" manifest:iteration-count="1024"
manifest:salt="UIv7yfLKliAq8yN5ukoI3g=="/>
>   </manifest:encryption-data>
>  </manifest:file-entry>
```

All the archive files are encrypted, except the *manifest.xml* file. This is a very surprising point since all the attacks presented here have been made possible by simply editing and modifying this particular file.

2.3 Digitally signed document.

Using digital signature requires getting a certificate. OpenOffice relies on external components for that purpose. We will not focus on how to import those certificates (see the OpenOffice Inline Help). Let us just precise that we used X509 certificates for our attacks. Let us mention that the ODF specifications [3] does not explicit the use of digital signature. We thus had to analyse how signature applies. For that purpose, let us consider an encrypted OpenOffice document containing a macro called *ref_mac_enc_sig.odt.*

Due to lack of space, we will just give the most relevant data. Other detailed data will be provided on request.

```
ZZR:~/Research/Analysis/OpenOffice.org/Work/Attaque_odf_190507/Struct_study lrv$ unzip
ref_macro_enc_sig.odt -d ref_macro_enc_sig_ext
Archive:  ref_macro_enc_sig.odt
 extracting: ref_macro_enc_sig_ext/mimetype
   creating: ref_macro_enc_sig_ext/Configurations2/statusbar/
 extracting: ref_macro_enc_sig_ext/Configurations2/accelerator/current.xml
   creating: ref_macro_enc_sig_ext/Configurations2/floater/
   creating: ref_macro_enc_sig_ext/Configurations2/popupmenu/
   creating: ref_macro_enc_sig_ext/Configurations2/progressbar/
   creating: ref_macro_enc_sig_ext/Configurations2/menubar/
   creating: ref_macro_enc_sig_ext/Configurations2/toolbar/
   creating: ref_macro_enc_sig_ext/Configurations2/images/Bitmaps/
  inflating: ref_macro_enc_sig_ext/META-INF/macrosignatures.xml
  inflating: ref_macro_enc_sig_ext/META-INF/documentsignatures.xml
 extracting: ref_macro_enc_sig_ext/content.xml
 extracting: ref_macro_enc_sig_ext/Basic/Standard/HelloWord.xml
 extracting: ref_macro_enc_sig_ext/Basic/Standard/script-lb.xml
 extracting: ref_macro_enc_sig_ext/Basic/script-lc.xml
 extracting: ref_macro_enc_sig_ext/styles.xml
  inflating: ref_macro_enc_sig_ext/meta.xml
 extracting: ref_macro_enc_sig_ext/Thumbnails/thumbnail.png
 extracting: ref_macro_enc_sig_ext/settings.xml
  inflating: ref_macro_enc_sig_ext/META-INF/manifest.xml
ZZR:~/Research/Analysis/OpenOffice.org/Work/Attaque_odf_190507/Struct_study lrv$
```

We can easily notice that two new files have been added into the archive:

META-INF/macrosignatures.xml and *META-INF/documentsignatures.xml*. The *manifest.xml* file contains the relevant data for those two files:

```
<manifest:file-entry manifest:media-type="" manifest:full-path="META-INF/macrosignatures.xml"/>
<manifest:file-entry manifest:media-type="" manifest:full-path="META-INF/documentsignatures.xml"/>
.....
```

Those two new files are not encrypted. This point is very essential to fully understand the attack mechanisms of encrypted documents. The *document.xml* file contains the detailed data that are required during the signature process itself (not given here). Every file in the archive is not signed while macros signature are declared within the *macrosignatures.xml* file. To be more precise, the digital signature is applied to any file containing or relating to macros [1, 2]

3.- A formal approach of OpenOffice digital signature

OpenOffice.org's security is based on two essential mechanisms: password based-encryption and digital signature. Both aim at preventing an illegitimate use or manipulation of a document, in the particular context of document malware, any weakness with respect to any of these mechanisms could be exploited in a dramatically powerful way to fool the user's trust in both cryptographic protection.

Since there exist a lot of way of using encryption and signature to protect an OpenOffice document, we are going to use a formal graph-based approach to describe them all. Every node in our graph describes a user's action. A given path in our graph will just describe a sequence of such actions to encrypt and/or signed a document. This approach is very powerful at detecting security flaws, in other words the cases where the mechanisms are supposed to have been successfully applied while in reality they are not (the document is not encrypted or not signed contrary to the user's intent).

Our graph-based formalization aims at identifying weaknesses in the signature process, in particular with respect to the macros. To summarize, we are going to show that signature of document visible content and signature of the macros are mutually exclusive: we cannot signed them at the same time. Using technical

examples, we will prove in the subsequent sections that it constitutes a serious design flaw that can be efficiently exploited by malware.

Every node describes a possible status for the document:

- D: modified document,
- MD: modified document with macro,
- ED: saved document,
- EMD: saved document with macro,
- SED: document is signed and saved,
- MSD: document with a macro added AFTER the document has been signed,
- EMSD: document with a macro added AFTER the document has been signed but BEFORE the document is saved,
- SEMD: signed and saved document with a macro.

Nodes are connected by possibly labeled directed arc. The label describes a user's command/action applied to the document:

- add M: add a macro,
- save: save document,
- sig 1: sign through *File → Digital Signatures...* menus,
- sig 2: sign through *Tools → Macros → Digital Signature...*,
- sig 3: sign through the second bottom right box in the OpenOffice GUI.

The corresponding graph is depicted in Figure 1.

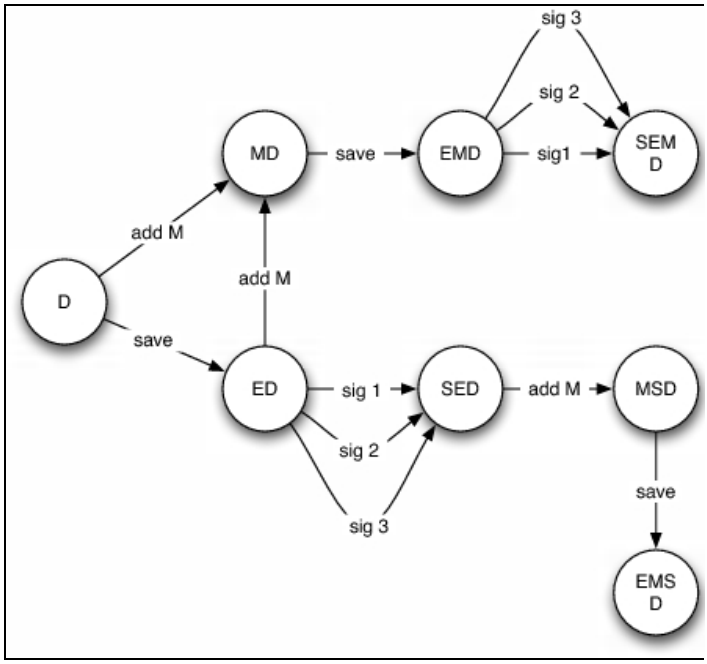


Figure 1.-OpenOffice Signature Graph

In Figure 1, we notice that the graph is divided into two connected component. A first sub-graph is made up of nodes {MD, EMD, SEMD} while the second one contains nodes {ED, SED, MSD, EMSD}. This supposes two different possible uses of digital signature that can be applied at any time in the life of the document. However, our experiments have proved that it may be quite different indeed. Let us explain why.

When considering signature AND encryption at the same time, our approach remains essentially the same. The set of nodes is generalized as follows:

- D: modified document,
- MD: modified document with macro,
- (SE)MD: encrypted and saved document with macro,
- S(SE)MD: signed, encrypted and saved document with a macro,
- (SE)D: encrypted and signed document,
- M(E)D: a macro is added to an encrypted and saved document,

- SM(E)D: a macro is added to an encrypted, saved and finally signed document,
- S(SE)D: modified document which has been saved, encrypted and signed,
- MS(E)D: a macro is added to an encrypted and signed document,
- EMS(E)D: signed then encrypted document with macro.

We thus obtain the graph depicted in Figure 2. It is very powerful at identifying potential misuses of digital signature in OpenOffice. Such misuses could be potentially by malware as we will show it in the next sections. Two classes of design flaws have been identified. The first class deals with the lack of OpenOffice built-in document integrity management. The second class refers to problems that may occur when macros and/or document are signed. We will not discuss the critical of trust macros. They are presented in [4].

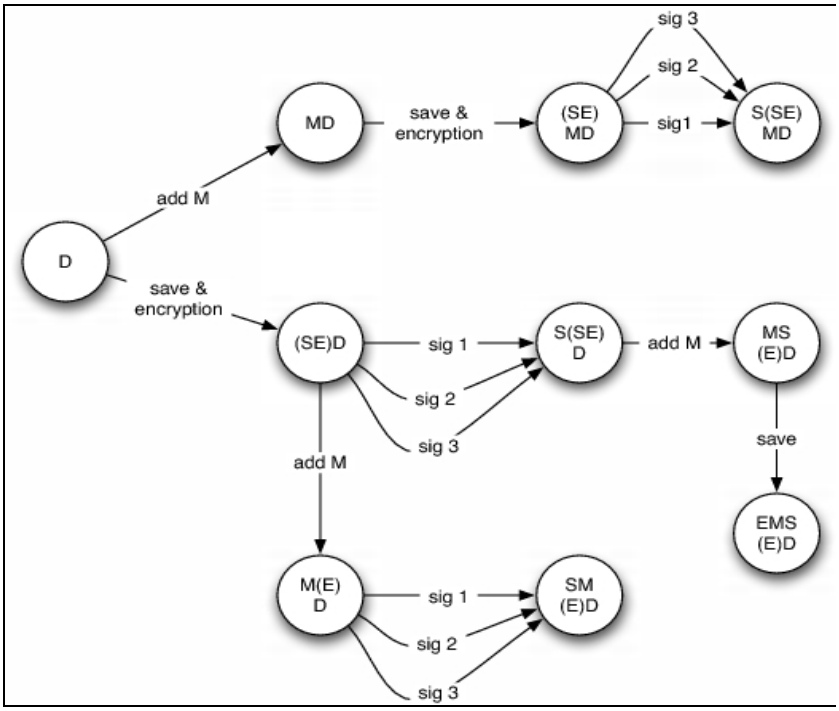


Figure 2.-OpenOffice Encryption and Signature Graph.

4.- Security issues in OpenOffice encryption and signature

The main problems do not lie in the cryptographic tools themselves but on their implementation and their management. The main consequence is that a malware could be able to identify some interesting blocs of instructions and thus adapt itself to the target document. This is particularly worrying with respect to macros which constitute execution points that can be subverted by a malware, despite the apparent use of digital signature.

In this section, we are going to consider some cases where it is possible for a malware to bypass digital signature or to exploit misuse of it. All these experiments have been successfully carried without causing any integrity violation alert while preserving the document usability.

Let us first mention that the use of digital signature can be identified within the archive in two ways:
- an additional file, denoted *documentsignatures.xml* is created in the META-INF directory of the archive,
- an entry is added into the META-INF/*manifest.xml* file:

```
<manifest:file-entry manifest:media-type="" manifest:full-path="META INF/documentsignatures.xml"/>
```

4.1.- First case: signed OpenOffice document with an unsigned macro

Let us first create a signed document with a macro. Contrary to the user's belief, the macro itself is not signed. As a consequence (see next sections), a malware can modify a macro without triggering any integrity

violation alert while abusing the user's security feeling with respect to digital signature.

The listing of the archive clearly shows that there is a signature which is applied to the document only but to the macro. The META-INF/*documentsignatures.xml* does not refer to any macro signature.

```
ZZR:~/Research/Analysis/OpenOffice.org/work/Attaque_odf_190507 lrv$ unzip doc.odt -d doc_dir
Archive:  doc.odt
 extracting: doc_dir/mimetype
   creating: doc_dir/Configurations2/statusbar/
  inflating: doc_dir/Configurations2/accelerator/current.xml
   creating: doc_dir/Configurations2/floater/
   creating: doc_dir/Configurations2/popupmenu/
   creating: doc_dir/Configurations2/progressbar/
   creating: doc_dir/Configurations2/menubar/
   creating: doc_dir/Configurations2/toolbar/
   creating: doc_dir/Configurations2/images/Bitmaps/
  inflating: doc_dir/META-INF/documentsignatures.xml
  inflating: doc_dir/content.xml
  inflating: doc_dir/Basic/Standard/Hello.xml
  inflating: doc_dir/Basic/Standard/script-lb.xml
  inflating: doc_dir/Basic/script-lc.xml
  inflating: doc_dir/styles.xml
  inflating: doc_dir/meta.xml
  inflating: doc_dir/Thumbnails/thumbnail.png
  inflating: doc_dir/settings.xml
  inflating: doc_dir/META-INF/manifest.xml
```

4.2.- Second case: signed OpenOffice document with a signed macro

In this second case, the signature is applied independently to the visible part of the document and to the macro itself. When is it possible to apply the signature to both of them simultaneously? The only solution is to successively follow path sig2 AND paths sig1 OR sig3 in the graph of Figure 2.

To illustrate this, let us first sign the macro:

```
ZZR:~/Research/Analysis/OpenOffice.org/work/Attaque_odf_190507/App_dsig_doc_macro lrv$ unzip
doc1.odt -d doc1_dir
Archive:  doc1.odt
 extracting: doc1_dir/mimetype
   creating: doc1_dir/Configurations2/statusbar/
  inflating: doc1_dir/Configurations2/accelerator/current.xml
   creating: doc1_dir/Configurations2/floater/
   creating: doc1_dir/Configurations2/popupmenu/
   creating: doc1_dir/Configurations2/progressbar/
   creating: doc1_dir/Configurations2/menubar/
   creating: doc1_dir/Configurations2/toolbar/
   creating: doc1_dir/Configurations2/images/Bitmaps/
  inflating: doc1_dir/META-INF/macrosignatures.xml
  inflating: doc1_dir/content.xml
  inflating: doc1_dir/Basic/Standard/Hello.xml
  inflating: doc1_dir/Basic/Standard/script-lb.xml
  inflating: doc1_dir/Basic/script-lc.xml
  inflating: doc1_dir/styles.xml
  inflating: doc1_dir/meta.xml
  inflating: doc1_dir/Thumbnails/thumbnail.png
  inflating: doc1_dir/settings.xml
  inflating: doc1_dir/META-INF/manifest.xml
```

Only the macro is signed. Let us then sign the document content itself. The listing clearly shows two different signature files:

```
ZZR:~/Research/Analysis/OpenOffice.org/work/Attaque_odf_190507/App_dsig_doc_macro lrv$ unzip
doc1.odt -d doc1_dir
Archive:  doc1.odt
 extracting: doc1_dir/mimetype
   creating: doc1_dir/Configurations2/statusbar/
  inflating: doc1_dir/Configurations2/accelerator/current.xml
   creating: doc1_dir/Configurations2/floater/
   creating: doc1_dir/Configurations2/popupmenu/
   creating: doc1_dir/Configurations2/progressbar/
   creating: doc1_dir/Configurations2/menubar/
   creating: doc1_dir/Configurations2/toolbar/
   creating: doc1_dir/Configurations2/images/Bitmaps/
  inflating: doc1_dir/META-INF/macrosignatures.xml
  inflating: doc1_dir/META-INF/documentsignatures.xml
  inflating: doc1_dir/content.xml
  inflating: doc1_dir/Basic/Standard/Hello.xml
  inflating: doc1_dir/Basic/Standard/script-lb.xml
  inflating: doc1_dir/Basic/script-lc.xml
  inflating: doc1_dir/styles.xml
  inflating: doc1_dir/meta.xml
  inflating: doc1_dir/Thumbnails/thumbnail.png
  inflating: doc1_dir/settings.xml
  inflating: doc1_dir/META-INF/manifest.xml
```

A deeper analysis clearly shows that the two signatures files have not been created at the time:

```
ZZR:~/Research/Analysis/OpenOffice.org/work/Attaque_odf_190507/App_dsig_doc_macro/doc1_dir2/MET
A-INF lrv$ ls -l *
-rw-r--r--  1 lrv  lrv  5875 Jul 12 14:45 documentsignatures.xml
-rw-r--r--  1 lrv  lrv  5657 Jul 12 14:30 macrosignatures.xml
-rw-r--r--  1 lrv  lrv  2602 Jul 12 14:45 manifest.xml
```

All this constitutes a critical design weakness since the user must be aware of the fact that a specific signature process must applied to sign OpenOffice macros. **The main consequence is that most of the time it is possible to modify (infect) macros without violating the document's integrity.**

5.- Attacking ODF: a formalization

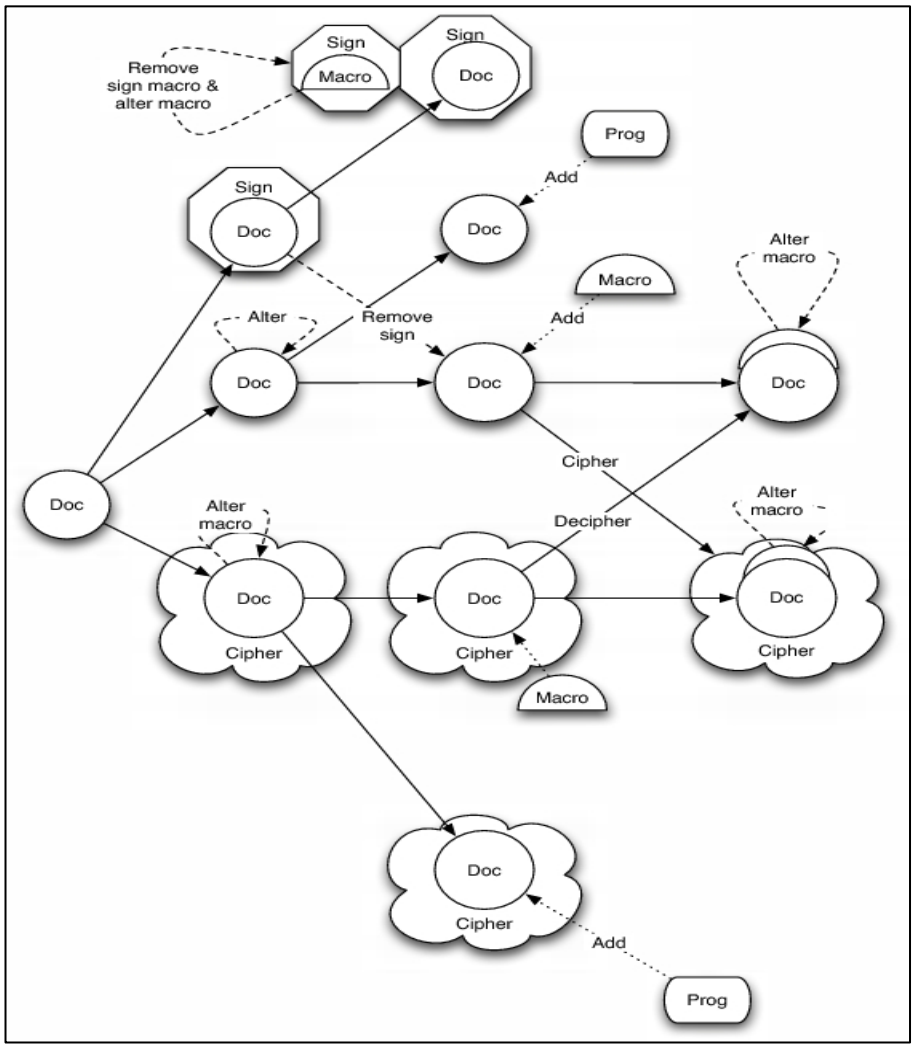
ODF is based on the XML technology. All the information is contained within XML tags, thus giving a semantic value to the information. All the XML tags are then organized within a tree that makes the extraction of the information far easier. These structures enable to formalize ODF in a very powerful way by means of automata and language theories. For sake of clarity we will not present our whole formalization work. The reader can refer to our technical report [8].

In order to identify all possible attacks that can be operated by malware against an OpenOffice document by exploiting integrity and/or signature management flaws, we have applied a formal model based on a graph-theoretic approach again. The graph is defined as follows:

- Each node represents the document status at time instant *t*, before any action is applied to it.
- Each arc models the different possible attacks that can be performed once a given action (command) has been applied.
- The node surrounding area defined a given features for the document.
- An action is applied to a node feature and thus defines which attack has been performed:
    - *Add*: add a property or a macro to the document,
    - *Modify*: modify a document property or component,

- *Cipher*: the document is encrypted by using OpenOffice built-in tools,
- *Decipher*: the encrypted document is deciphered by using OpenOffice built-in tools.

We thus obtain the attack graph depicted in Figure 3. It just describes things at the lowest level. It is possible to combine different document status to exhaustively describe all possible attack at every possible level.



6.- Security issues in OpenOffice integrity management

All our experiments [4, 7, 8] and real case analyses have proved that it is indeed possible to infect an OpenOffice document very easily. Moreover this can be performed without triggering any alert or even arleting the user of the presence of macros (use of trust macros) [4]. The most critical point lies in the fact that all these attacks are possible despite the fact that all security measures have been applied (encryption, signature...). Since the work presented in [4, 7], all these conceptual flaws still exist in the 2.2.x releases.

In the present section, we are going to present some of the most critical classes of attacks. All of the attacks presented in [7] for OpenOffice 2.0.x are still efficient for the 2.2.0 release. That is why we will just recall some of them and just add new ones with respect to the digital signature. They have been identified very recently.

All the attacks lie on the modification of the following files in ODF archives.

- *content.xml*,
- *META-INF/manifest.xml*,
- *Basic/script-lc.xml*,
- *Basic/<library_name>.xml or Standard.xml*
- *Basic/<library_name>/script-lb.xml,*
- *Basic/<library_name>/<macro_name>.xml.*

6.1- Modifying an encrypted document with macro

Let us consider an encrypted document whose macro is encrypted as well. We are going to show how to replace the encrypted macro with a malicious, unencrypted one. When the user opens and deciphers (by inputing his password), no alert will be issued but the malicious macro will be ran. Let us first give the structure of this document before any modification. Once again it is contained in the META-INF/*manifest.xml* file. Parts of the file related to the encryption are in red color except those referring to the macro encryption, which are in blue color.

```
<?xml version="1.0" encoding="UTF-8"?>
<manifest:manifest xmlns:manifest="urn:oasis:names:tc:opendocument:xmlns:manifest:1.0">
 <manifest:file-entry manifest:media-type="application/vnd.oasis.opendocument.text" manifest:full-path="/"/>
 <manifest:file-entry manifest:media-type="" manifest:full-path="Configurations2/statusbar/"/>
 <manifest:file-entry manifest:media-type="" manifest:full-path="Configurations2/accelerator/current.xml" manifest:size="0">
  <manifest:encryption-data manifest:checksum-type="SHA1/1K" manifest:checksum="aIk0hF8iBJyxRmiDLvoz1FATtrk=">
   <manifest:algorithm manifest:algorithm-name="Blowfish CFB" manifest:initialisation-vector="Aft5D8rS4Tc="/>
   <manifest:key-derivation manifest:key-derivation-name="PBKDF2" manifest:iteration-count="1024" manifest:salt="7+uA1gcyifrwus8NAJ4P0g=="/>
  </manifest:encryption-data>
 </manifest:file-entry>
 <manifest:file-entry manifest:media-type="" manifest:full-path="Configurations2/accelerator/"/>
 <manifest:file-entry manifest:media-type="" manifest:full-path="Configurations2/floater/"/>
 <manifest:file-entry manifest:media-type="" manifest:full-path="Configurations2/popupmenu/"/>
 <manifest:file-entry manifest:media-type="" manifest:full-path="Configurations2/progressbar/"/>
 <manifest:file-entry manifest:media-type="" manifest:full-path="Configurations2/menubar/"/>
 <manifest:file-entry manifest:media-type="" manifest:full-path="Configurations2/toolbar/"/>
 <manifest:file-entry manifest:media-type="" manifest:full-path="Configurations2/images/Bitmaps/"/>
 <manifest:file-entry manifest:media-type="" manifest:full-path="Configurations2/images/"/>
 <manifest:file-entry manifest:media-type="application/vnd.sun.xml.ui.configuration" manifest:full-path="Configurations2/"/>
 <manifest:file-entry manifest:media-type="text/xml" manifest:full-path="content.xml" manifest:size="2654">
  <manifest:encryption-data manifest:checksum-type="SHA1/1K" manifest:checksum="cLjeVcn0HUvWH6AksJUt9B+/m80=">
   <manifest:algorithm manifest:algorithm-name="Blowfish CFB" manifest:initialisation-vector="88UAHW9S7AA="/>
   <manifest:key-derivation manifest:key-derivation-name="PBKDF2" manifest:iteration-count="1024" manifest:salt="l4mR5N16lc15CM2i1+thdg=="/>
  </manifest:encryption-data>
 </manifest:file-entry>
 <manifest:file-entry manifest:media-type="text/xml" manifest:full-path="Basic/Standard/Mess_to_user.xml" manifest:size="347">
  <manifest:encryption-data manifest:checksum-type="SHA1/1K" manifest:checksum="Ak30lrpgYdX/q3qq4qjtJYfW3WQ=">
   <manifest:algorithm manifest:algorithm-name="Blowfish CFB" manifest:initialisation-vector="vu7rTd3OYWU="/>
   <manifest:key-derivation manifest:key-derivation-name="PBKDF2" manifest:iteration-count="1024" manifest:salt="KIeIhkKFlu0+C4eL1E7EwQ=="/>
  </manifest:encryption-data>
 </manifest:file-entry>
<manifest:file-entry manifest:media-type="text/xml" manifest:full-path="Basic/Standard/script-lb.xml" manifest:size="353">
  <manifest:encryption-data manifest:checksum-type="SHA1/1K"
```

```
manifest:checksum="4FmXs2oOBSk6bWqLsvUFMrnp/ik=">
  <manifest:algorithm manifest:algorithm-name="Blowfish CFB" manifest:initialisation-
vector="yki90zxcSVU="/>
  <manifest:key-derivation manifest:key-derivation-name="PBKDF2" manifest:iteration-count="1024"
manifest:salt="th0bGueB7lHhnzbeYGgvyA=="/>
 </manifest:encryption-data>
</manifest:file-entry>
<manifest:file-entry manifest:media-type="" manifest:full-path="Basic/Standard/"/>
<manifest:file-entry manifest:media-type="text/xml" manifest:full-path="Basic/script-lc.xml"
manifest:size="338">
 <manifest:encryption-data manifest:checksum-type="SHA1/1K"
manifest:checksum="EClic6byHiSVEsuYf5VZ85y2C5A=">
  <manifest:algorithm manifest:algorithm-name="Blowfish CFB" manifest:initialisation-
vector="fa/vxhT25c0="/>
  <manifest:key-derivation manifest:key-derivation-name="PBKDF2" manifest:iteration-count="1024"
manifest:salt="dmJtqGXRW+sO+o8vU/GbiQ=="/>
 </manifest:encryption-data>
</manifest:file-entry>
<manifest:file-entry manifest:media-type="" manifest:full-path="Basic/"/>
<manifest:file-entry manifest:media-type="text/xml" manifest:full-path="styles.xml"
manifest:size="8315">
 <manifest:encryption-data manifest:checksum-type="SHA1/1K"
manifest:checksum="qgwehDuLTFNDAo7TKMExjmID9tY=">
  <manifest:algorithm manifest:algorithm-name="Blowfish CFB" manifest:initialisation-
vector="8dWw8yHo5aU="/>
  <manifest:key-derivation manifest:key-derivation-name="PBKDF2" manifest:iteration-count="1024"
manifest:salt="KTXWF5oelquuWtzKsibnTg=="/>
 </manifest:encryption-data>
</manifest:file-entry>
<manifest:file-entry manifest:media-type="text/xml" manifest:full-path="meta.xml"/>
<manifest:file-entry manifest:media-type="" manifest:full-path="Thumbnails/thumbnail.png"
manifest:size="4252">
 <manifest:encryption-data manifest:checksum-type="SHA1/1K"
manifest:checksum="oJf7JAjmPn/7q76QPXSxjNdN8RM=">
  <manifest:algorithm manifest:algorithm-name="Blowfish CFB" manifest:initialisation-
vector="ezfIUx0E/2A="/>
  <manifest:key-derivation manifest:key-derivation-name="PBKDF2" manifest:iteration-count="1024"
manifest:salt="D5J8wBvv1c4YAQlOvek6EA=="/>
 </manifest:encryption-data>
</manifest:file-entry>
<manifest:file-entry manifest:media-type="" manifest:full-path="Thumbnails/"/>
<manifest:file-entry manifest:media-type="text/xml" manifest:full-path="settings.xml"
manifest:size="7477">
 <manifest:encryption-data manifest:checksum-type="SHA1/1K"
manifest:checksum="XBWgGb0E8QJocGNDRgAluLWQ0yI=">
  <manifest:algorithm manifest:algorithm-name="Blowfish CFB" manifest:initialisation-
vector="Rjtsrax4yr4="/>
  <manifest:key-derivation manifest:key-derivation-name="PBKDF2" manifest:iteration-count="1024"
manifest:salt="R2OqHLfNJBy9S6be6+/F9Q=="/>
 </manifest:encryption-data>
</manifest:file-entry>
</manifest:manifest>
```

Now let us modify the document in order to insert a malicious macro while bypassing the encryption.
Here follows the extract of the META-INF/*manifest.xml* file which refers to the component to be modified.

```
<manifest:file-entry manifest:media-type="text/xml" manifest:full-path="Basic/Standard/Mess_to_user.xml"
manifest:size="347">
  <manifest:encryption-data manifest:checksum-type="SHA1/1K"
manifest:checksum="Ak30lrpgYdX/q3qq4qjtJYfW3WQ=">
   <manifest:algorithm manifest:algorithm-name="Blowfish CFB" manifest:initialisation-
vector="vu7rTd3OYWU="/>
   <manifest:key-derivation manifest:key-derivation-name="PBKDF2" manifest:iteration-count="1024"
manifest:salt="KIeIhkKFlu0+C4eL1E7EwQ=="/>
  </manifest:encryption-data>
 </manifest:file-entry>
```

The content of the encrypted macro then here follows:

```
y~}I\_^K<97>ý÷\^E^YÐ"¨\¨¯^Q<99><9c>´^Ytgñû^Si;!<85>^Aý^T,<84>AN±ÌÙ£^EÏ^P<8d>òì\^egU<97>
^Se±(WÞ°^LrÒøk[x#ËËEE<92>\'EÙÙ\'elZ\^aÎzK\`A<95><8e>*×<91>^CÔÁS}ebä~<93>|M%^ä-
*ºÖIW^Kb{^S¬j5^U<98><99>.^Z÷³<98><8d>¯<99>@<91>Ifæ%õ<85>ö\^A\^A<82>ò¢<9c>L¾<8c>R
Ë  Ï§Î´ûB~øtrËGJ^?L,Cw½^T^X\`eÝ<8f>õ<96>   #l^NG<8e><85>;Æ<94>Ù:ñùj
õ^H¡<9e>^A¥}Ä'RVm.^Zñ<81>r}^@¯<85>^@¦ü»äÁ^\Ì^[^@<98>'þ0<8e>+G
 \`A<92>0Ë{õNg<89>³ ¥&Dðý
```

Let us now replace this encrypted macro with a malicious, unencrypted macro. In a first step, we have to modify the META-INF/*manifest.xml* file accordingly.

```
<manifest:file-entry manifest:media-type="text/xml" manifest:full-path="Basic/Standard/Mess_to_user.xml"
manifest:size="347"/>
```

We have just removed all references to encryption. Now we just have to replace the encrypted macros itself with the malicious one whose code here follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE script:module PUBLIC "-//OpenOffice.org//DTD OfficeDocument 1.0//EN" "module.dtd">
<script:module xmlns:script="http://openoffice.org/2000/script" script:name="mess_to_user"
script:language="StarBasic">REM  *****  BASIC  *****

Sub Main
msgbox(&quot; I am a malicious macro... hey hey, I have just infected your document...&quot;)
End Sub
</script:module>
```

No alert will be issued at the document opening and the malicious macro will operate. In case of a trust macro, the presence of the macro would even not be noticed to the user, thus increasing the infectious power of the macro.

6.2.- Modifying a signed document without macro.

Let us now consider a digitally signed document without any macro. The digital signature is applied according to the path sig1 or sig2 in the graph presented in Figure 1. In the present case, we are going to bypass the digital signature to insert a malicious macro. This is possible since in this situation, the document's content only is signed. For that purpose, let us just remove the information which relates to the signature, in the META-INF/*manifest.xml* file:

```
<manifest:file-entry manifest:media-type="" manifest:full-path="META-INF/documentsignatures.xml"/>
```

Then we remove the META-INF/*documentsignatures.xml* file in the archive. When the user opens the document no integrity violation alert is triggered. However, the bottom icon in OpenOffice GUI, which indicates that the document is signed has disappeared. This could alert the recipient user.
The second step of the infection consists in keeping the signature but to insert a malicious macro into the

document. Let us consider the following macro:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE script:module PUBLIC "-//OpenOffice.org//DTD OfficeDocument 1.0//EN" "
module.dtd">
<script:module xmlns:script="http://openoffice.org/2000/script" script:name="mes
s_to_user" script:language="StarBasic">REM  *****  BASIC  *****

Sub Main
msgbox(&quot;I am a malicious macro... hey hey, I have just infected your document...&quot
;)
End Sub
</script:module>
```

The last step consists in adding the information relating to the existence of this macro into the *META-INF/manifest.xml* file.

```
 <manifest:file-entry manifest:media-type="text/xml" manifest:full-
path="Basic/Standard/Mess_to_user.xml"/>
 <manifest:file-entry manifest:media-type="text/xml" manifest:full-path="Basic/Standard/script-lb.xml"/>
 <manifest:file-entry manifest:media-type="" manifest:full-path="Basic/Standard/"/>
 <manifest:file-entry manifest:media-type="text/xml" manifest:full-path="Basic/script-lc.xml"/>
 <manifest:file-entry manifest:media-type="" manifest:full-path="Basic/"/>
```

The signature's icon is still present in the OpenOffice GUI. No alert will be triggered, the user has been successfully fooled and the macro will operate.

6.3.- Modifying a signed document with signed or unsigned macro.

In this third class of attacks, let us consider a document with macro where both the document's content and the macro have been signed. In a similar way as presented before, the infection consist in bypassing the signature by modifying all the relevant information, that is to say, to remove the following data in the META-INF/*manifest.xml*:

<manifest:file-entry manifest:media-type="" manifest:full-path="META-INF/macrosignatures.xml"/>
Then we just have to remove the *macrosignatures.xml* file from the archive. Now we are back to the previous case of a signed document with unsigned macros. According to this case, we have to modify the data located between the <*script:module xmlns:script="http://openoffice.org/2000/script" script:name="hello" script:language="StarBasic"*> tag and the <*/script:module*> tag.

The original macro:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE script:module PUBLIC "-//OpenOffice.org//DTD OfficeDocument 1.0//EN" "module.dtd">
<script:module xmlns:script="http://openoffice.org/2000/script" script:name="hello"
script:language="StarBasic">REM  *****  BASIC  *****

Sub Main
MsgBox(&quot;Hello World&quot;)
End Sub
</script:module>
```

has been modified as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE script:module PUBLIC "-//OpenOffice.org//DTD OfficeDocument 1.0//EN" "module.dtd">
<script:module xmlns:script="http://openoffice.org/2000/script" script:name="hello"
script:language="StarBasic">REM  *****  BASIC  *****

Sub Main
MsgBox(&quot;Hello Mr User, your macro has just have been hacked :)&quot;)
End Sub
</script:module>
```

Once again, when the document is subsequently opened, no alert at all is triggered. It is thus very easy to infect a document while keeping the digital signature referring to the document's content only, thus fooling the user's feeling of security provided by digital signature.

6.4.- Modifying an encrypted and signed document with macro.

In this last case, the document is first signed (including the macro) and then encrypted. This should be considered as the most secure way to protect an OpenOffice document, thus preventing any document infection. It is unfortunately not the case. Let us show how to bypass both the encryption and the digital signature. In the META-INF/*manifest.xm*l file, let us first remove the following parts which relate to the macro encryption and signature:

```
<manifest:file-entry manifest:media-type="" manifest:full-path="META-INF/macrosignatures.xml"/>
.....
 <manifest:encryption-data manifest:checksum-type="SHA1/1K"
manifest:checksum="YcgygyDHQ1NUCAB80HA5Z4C24No=">
  <manifest:algorithm manifest:algorithm-name="Blowfish CFB" manifest:initialisation-
vector="QcCMCISZu+8="/>
  <manifest:key-derivation manifest:key-derivation-name="PBKDF2" manifest:iteration-count="1024"
manifest:salt="RDdsU3RxAIqYmBhfgviwug=="/>
 </manifest:encryption-data>
</manifest:file-entry>
.....
 <manifest:encryption-data manifest:checksum-type="SHA1/1K"
manifest:checksum="AVJqugo0F2xvU9KaiKcanc17mgE=">
  <manifest:algorithm manifest:algorithm-name="Blowfish CFB" manifest:initialisation-
vector="e/nOQd+LvKY="/>
  <manifest:key-derivation manifest:key-derivation-name="PBKDF2" manifest:iteration-count="1024"
manifest:salt="Z2YgG2pkbAekJZ4AVvaLyg=="/>
 </manifest:encryption-data>
</manifest:file-entry>
.....
 <manifest:encryption-data manifest:checksum-type="SHA1/1K"
manifest:checksum="EClic6byHiSVEsuYf5VZ85y2C5A=">
  <manifest:algorithm manifest:algorithm-name="Blowfish CFB" manifest:initialisation-
vector="flO0sxd0s4w="/>
  <manifest:key-derivation manifest:key-derivation-name="PBKDF2" manifest:iteration-count="1024"
manifest:salt="+F5uK/4ALZuR2Q1Kl1uD0Q=="/>
 </manifest:encryption-data>
</manifest:file-entry>
.....
```

Then, let us remove the META-INF/*macrosignatures.xml* file from the archive. Finally we replace the encrypted macros:

```
/<9a>Îd<9d>gÍäëmBðÿa}<93>^A8çèîQâtä^W÷ 4¬<81>ßSÿ3^Vþ^QE^E)0lo^UY<81>ªeÄ^P§'(søòú^C^[X^
TQÁÜ¸^LVö½^Uc²oÚÉÿ^P<9a>^_XÜ^QT"b^]nôó°^L1à^?Ê^Yð^KQÏ"0^TnC>IÑSx¡¹^Q<9a>Ø´^G^@¸t<91
>^^^[<95>íó^CÂ÷ï^Oû<97>^S®^KWV<92> <87>¯^U
´¦<81>òøÍ§ÿ<91>¾^C<9f>Z^\õ÷Ôúâ^_³|w\.*£^YÔ^K^@°ÀvÚ}và<82><8a>Ãô·^YkeÍâï^ö¦ÿÆ(<83>Íð
æÀ|X3¬=þþd@^R'ú^Q©ç<88>^Lc^H^E<8c>Ì^Nì<8d>ÜÞ
```

with a malicious (unencrypted) one:

```
 <?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE script:module PUBLIC "-//OpenOffice.org//DTD OfficeDocument 1.0//EN" "module.dtd">
<script:module xmlns:script="http://openoffice.org/2000/script" script:name="hello"
script:language="StarBasic">REM  *****  BASIC  *****

Sub Main
MsgBox(&quot;Hello Mr User, your macro has just have been hacked :)&quot;)
End Sub
</script:module>
```

At the document opening, strictly no alert is triggered despite the fact that the document's signature is still present. The malicious macro has been executed successfully.

A less brutal approach would consist in removing the encryption data relating to the Basic/Standard/<*macro_name*>*.xml* file only. But for that purpose, it is required to exactly know the structure of the library.

6.5.- Other classes of attacks

Many other attacks can be performed despite the use of encryption and/or digital signature:

- adding external files into the archive: theft of documents from an OpenOffice (malicious) document,
- using complex macro libraries: all the previous work extends to complete libraries. It is just possible to perform very complex malicious attacks.
- ....

7.- Conclusion

The in-depth analysis of OpenOffice security has exhibited design flaws that make viral attacks very easy and powerful, by fooling the user's feeling of security provided by both encryption and digital signature. These techniques are badly managed and thus can be simply bypassed by using a simple text editor.

All relevant data have been provided to the OpenOffice developers and we hope that they will very soon issue a new, secure release that will correct all these weaknesses. But OpenOffice's design philosophy must be deeply changed in order to better manage the integrity of OpenOffice documents – the most critical issue underlying all the OpenOffice weaknesses. The question is: is it possible to really offer security while being totally open? Since the attacker also has access to the whole security specifications, he has total control on the security mechanisms, unless they are not public or a secret parameter is used – a key – and efficiently managed. In this context, the odds are in the proprietary software's favours.

We strongly advise OpenOffice users to protect their document by external encryption and digital signature (e.g. PGP). At the present time, it is the only possible solution. On the AV software's side, antivirus should warn of the presence of any macro in OpenOffice document and of the fact that the document is signed but not the macros. They also could issue an alert whenever an encrypted document contains unencrypted macros. It is sure that AV products have to play an essential role in the context of OpenOffice security.

References

[1] W3C, Spécification signature W3C, http://www.w3.org/Signature/

[2] XML http://www.xml.com/pub/a/2001/08/08/xmldsig.html and http://www.w3.org/TR/xmldsig-core/

[3] Open Oasis, OpenDocument Specifications v1.1, http://www.oasis-open.org/specs/

[4] De Drézigué, D., Fizaine, J.-P. and Hansma, N. (2006), In-depth Analysis of the Viral Threats with OpenOffice.org Documents, *Journal in Computer Virology*, (2)-3, 2006.

[5] Rautiainen, S. (2003), OpenOffice Security. Virus Bulletin, september.

[6] Lagadec, P. (2006), OpenOffice/OpenDocument and MS Open XML Security. In PACSEC 2006 Conference, http://pacsec.jp/psj06archive.html

[7] Filiol, E. and Fizaine, J.-P. (2006), Le risque viral sous OpenOffice 2.0.x, MISC – Le journal de la sécurité informatique, vol. 27. http://www.miscmag.com

[8] Filiol, E and Fizaine, J.-P. (2007), Security Analysis of the ODF Security: a Formal Approach, ESAT Technical Report 2007-21.