J Comput Virol (2006) 2:187–210
DOI 10.1007/s11416-006-0020-2

ORIGINAL PAPER

In-depth analysis of the viral threats with OpenOffice.org documents

David de Drézigué · Jean-Paul Fizaine · Nils Hansma

Received: 10 April 2006 / Revised: 20 June 2006 / Accepted: 4 July 2006 / Published online: 1 August 2006

Abstract This paper presents an in-depth analysis of the OpenOffice suite (release 2.0.3) with respect to viral threats, independently of any software flaw or vulnerability. First we will identify, then analyse the different potential viral vectors of OpenOffice.org v2.0.3. Our examination applies to win32 and Unix-like platforms. For each identified vector, a detailed study will show to which extend the infection is possible. From then on we will define the solutions in order to maximize OpenOffice.org security in the production field as well as in the office tools at the administration level.

Keywords Malware · Macro virus · Document malware · Self-reproduction · Security Policy · Antiviral Policy

1 Introduction

For many years, the software sector of the office suite has been dominated by the supremacy of the Microsoft Office suite. This suite has succeeded in offering the end-user a powerful and ergonomic environment. Nevertheless, the security of the products making up this suite (*Word* text processing, *Excel* spreadsheet, presentation software such as *PowerPoint* …) has long been a subject liable to concerns. On top of issues related to information leaks [1], the risk linked to macro-virus has been a concern to the end-users [3] since the *Concept* macro-virus in 1995. The presence of macros in an office document is henceforth identified by the risk – at least potential – of macro-virus. Even though the antiviral offer enable to fight efficiently against the well-known macro-viruses or the identified techniques, it still remains pretty easy to conceive an undetected macro-virus by the current antivirus.

The development and the release of OpenOffice free software suite [11] for a few years seemed to inspire new hopes – and maybe hypes – as far as security linked to office documents was concerned.

The adoption of this suite by several countries and by foreign armies in particular (Singapore in 2006, the French Gendarmerie in 2005), some institutions, some universities… [12] led the media to see a sign of greater security into it, furthermore it costs almost nothing. The certainty of a greater security towards the viral risk has become an overspread feeling in the freeware community as much as in the mind of several end-users. The consequence is that the latters forget the security behaviour and reaction – and more particularly the caution against the presence of macros – when opening office documents. What should one think about the security of OpenOffice suite regarding macro-viruses? Myth or reality?

The OpenOffice suite, just like its commercial equivalent, contains an inserted development environment around several programming languages. The language OoBasic – the best known – has replaced the *Visual Basic for Applications* for macros writing. But other programming languages, more powerful than the plain OoBasic allow more sophisticated developments than

D. de Drézigué · J.-P. Fizaine (✉) · N. Hansma
Ecole Supérieure et d'Application des Transmissions,
Laboratoire de virologie et de cryptologie,
BP 18, 35998 Rennes Armées, France
e-mail: labo.virologie@esat.terre.defense.gouv.fr

D. de Drézigué · N. Hansma
Ecole des Systèmes de Combat et Armes Navals,
Centre d'Instruction Naval,
BP 500, 83800 Toulon Naval, France

the plain macro writing, no matter how complex they may be. These different kinds of execution contexts seem to argue in favour of the possible uses of macro-viruses for OpenOffice. Such a possibility has been previously evoked by Rautiainen [14] but this author considered only a few security aspects. Moreover, Rautiainen's paper dealt with the OpenOffice version 1.0.x. The OpenOffice release 2.0.x contains many more powerful capabilities as far as infectious threats are concerned.

This article introduces the real security analysis of the OpenOffice suite with regards to viral threat is concerned. Independently of any software vulnerability, the results show that not only can this suite not be considered immune to macro-viruses but above all, considering the current development condition of this suite, the later shows a higher rate of hazard than for her commercial competitor. In other words, any hazardous behaviour from the end-users may lead to serious outcomes over information security system. This security analysis has been recognized by the design and the testing of several operational[1] viral strains demonstrating that this risk is truly real and is giving cause for concern. Therefore any security policy should seriously consider the viral risk linked to this suite.

This article is partly based on [2] and is organized as follow. Sect. 2 will present the main identified vectors of potential infections for this suite, through a step by step analytic approach. Sect. 3 will deal with the operating of these infection vectors through malevolent codes. Finally, Sect. 4 will, on the one hand, ponder over the viral risk attached to the OpenOffice suite, and on the second hand, will introduce the algorithmic of some few viral stocks that have been developed and tested for validation purposes.

Disclaimer: *Developing proof-of-concepts is not a goal in itself. It remains the only scientific way to prove if a viral risk exists or not. Deficiency of antivirus may result in human casualties or even deaths, lost of jobs... when considering critical or very critical systems. Consequently any serious protection cannot rely only on the deployment of an antivirus, whatever may be its efficiency. No serious security policy of such critical systems could accept security enforcement without a proactive research including proof-of-concept developments. This study has been performed in the strict respect of the different existing laws. The source codes which have been developed will neither be disclosed nor published. Only reknowned and strictly identified IT security professionals are eligible for freely – and without any compensation of any kind – accessing these source code provided that they fill an application form to the Virology and Cryptology Laboratory of the Army Signals Academy.*

2 Identification of the potential infectious vectors: OpenOffice.org V2.0.x analysis

First we will analyze the file format in order to know its structure as well as its organization; which enable us to infer the way we will be able to manipulate it. Secondly, we will analyze the installation, the configuration so as to identify the potential propagation vectors for all the well known viral infection technologies [4]. We will define a potential propagation vector as follows:

- an automatic, scheduled or event-related execution point;
- use of a regular human task;
- execution transfer, or process creation;
- weak environment identification: configuration of the suitable application for the malevolent code execution.

2.1 The OpenDocument V1.0 file format

This file format is a standard introduced by Oasis-Open,[2] in 2005. It has recently adopted by ISO/IEC in May 2006, and called ISO/IEC DIS 26300, on the demand of the European Commission [8] with the agreement of SunMicrosystems and Oasis-Open. This is a file format based on the XML technology. It can be used with all kind of tools handling this technology. OpenDocument supports various other kind of document like database, diagrams, drawings, slideshows and spreadsheets. This file format is also supported by a number of various applications: AbiWord, Knomos, Koffice, OpenOffice, Scribus StartOffice and recently by Ibm Workspace but not with Office Microsoft. State of Massachusetts has first adopted this format, and recently Europe as standard exchange document format. The structure and organization of the file depend on the pattern specified by Oasis Open[3] [9]. We are presenting its main features that show specific relevance in the chosen context.

[1] These strains have been developed for the *Writer* component but are easily transposable to other software components of the suite.

[2] http://www.oasis-open.org

[3] A pattern is a document that describes the creation of an XML file format.

2.1.1 Analysis of a blank document

The "file" command on our blank reference file, points out that the file is a ZIP format archive. We notice that the archive is neither compressed nor protected by a password. This will not prevent an archive infection. The extraction of the archive is usually carried out with the help of any utility compatible with the ZIP format. Thus here follows the content of our blank reference file:

```
ZZR:~/research_projects/openoffice/openoffice-2.x/unzip_ref_file$
␣␣file␣reference_file.odt
reference_file.odt:␣Zip␣archive␣data,␣at␣least␣v2.0␣to␣extract
ZZR:~/research_projecy/openoffice/openoffice-2.x/unzip_ref_file$
␣␣unzip␣reference_file.odt
Archive:␣reference_file.odt
␣extracting:␣minetype
␣␣␣creating:␣Configuration2/
␣␣␣creating:␣Pictures/
␣␣inflating:␣_file.xml
␣␣inflating:␣styles.xml
␣extracting:␣meta.xml
␣␣inflating:␣setting.xml
␣␣inflating:␣META-INF/manifest.xml
```

```
ZZR:~/research_projects/openoffice/openoffice-2.x/unzip_ref_file$␣ls␣-l
total␣72
drwxr-xr-x␣␣␣2␣lrv␣␣lrv␣␣␣␣68␣Feb␣16␣15:46␣Configurations2
drwxr-xr-x␣␣␣3␣lrv␣␣lrv␣␣␣102␣Feb␣16␣16:51␣META-INF
drwxr-xr-x␣␣␣2␣lrv␣␣lrv␣␣␣␣68␣Feb␣16␣15:46␣Pictures
-rw-r--r--␣␣␣1␣lrv␣␣lrv␣␣2347␣Feb␣16␣15:46␣content.xml
-rw-r--r--␣␣␣1␣lrv␣␣lrv␣␣4427␣Feb␣16␣16:46␣reference_file.odt
-rw-r--r--␣␣␣1␣lrv␣␣lrv␣␣1047␣Feb␣16␣15:46␣meta.xml
-rw-r--r--␣␣␣1␣lrv␣␣lrv␣␣␣␣39␣Feb␣16␣15:46␣mimetype
-rw-r--r--␣␣␣1␣lrv␣␣lrv␣␣6607␣Feb␣16␣15:46␣setting.xml
-rw-r--r--␣␣␣1␣lrv␣␣lrv␣␣7623␣Feb␣16␣15:46␣styles.xml
ZZR:~/research_projects/openoffice/openoffice-2.x/unzip_ref_file$
```

The META-INF directory contains a *manifest* XML format file [9, Chap. 17]. The latest classifies the paths towards other XML files of the archive as well as information describing the archive as the encoding algorithm along with its parameters, the checksum passwords algorithm as well as the checksum value. The directories "*Configuration2*" and "*Pictures*" are empty. By lexicographic order:

- *content.xml*: this file is common to all types of OpenOffice.org documents. It can contain the following items: scripts, font settings, automatic style and document body.
- *meta.xml*: this file contains the document meta-information (author, date of the last action…),
- *styles.xml*: specifies the style used for the document,
- *setting.xml*: points to the configuration such as the program application window size or to printing information.

It is regarding the XML files format that the official literature [9, Chap. 2] provides a very useful understanding of an XML OpenOffice.org document structure.

2.1.2 Analysis of a user document

The reference test file is a simple document which contains styles, a rather important number of pages and its own text layout information. At this stage, neither macros, nor other objects (OLE objects, links, sound or video data...) or images are present yet. We consider again the previous analysis protocol. Let us compare this present user document with the blank reference one. We find again the same files and directories. Only the size of files *content.xml* and *setting.xml* have changed. Whereas the reference file has a size of 147,332 bytes, the user file has a size of 147,470 bytes. However the corresponding archives differ from one another in size of only five bytes. Moreover, the content of *setting.xml* file has changed. To make things clear, the content of the user document archive here follows:

```
ZZR:~/research_projects/openoffice/openoffice-2.x/experiment/ana_file_format$
  ls -l unzip_user_doc/
total 360
drwxr-xr-x   2 lrv  lrv      68 Feb 22 18:47 Configurations2
drwxr-xr-x   3 lrv  lrv     102 Mar  2 00:52 META-INF
drwxr-xr-x   2 lrv  lrv      68 Feb 22 18:47 Pictures
-rw-r--r--   1 lrv  lrv  147470 Feb 22 18:47 content.xml
-rw-r--r--   1 lrv  lrv      80 Feb 22 18:47 layout-cache
-rw-r--r--   1 lrv  lrv    1067 Feb 22 18:47 meta.xml
-rw-r--r--   1 lrv  lrv      39 Feb 22 18:47 mimetype
-rw-r--r--   1 lrv  lrv    7138 Feb 22 18:47 settings.xml
-rw-r--r--   1 lrv  lrv    9118 Feb 22 18:47 styles.xml
ZZR:~/research_projects/openoffice/openoffice-2.x/experiment/ana_file_format$
```

Let us now compare both archive sizes:

```
ZZR:~/research_projects/openoffice/openoffice-2.x/experiment/ana_file_format$
  ls -l
total 96
-rw-r--r--    1 lrv  lrv  22327 Feb 16 16:27 empty_doc.odt
drwxr-xr-x   11 lrv  lrv    374 Mar  2 00:50 unzip_empty_doc
drwxr_xr_x   11 lrv  lrv    374 Mar  2 00:52 unzip_user_doc
-rw-r--r--    1 lrv  lrv  22332 Feb 23 17:48 user_doc.odt
ZZR:~/research_projects/openoffice/openoffice-2.x/experiment/ana_file_format$
```

2.1.3 Comparison to a document with macros

Let us now insert a macro in the user document we have just considered. The name of this macro is *dicOOo* and belongs to the standard macros library (installation of a dictionary). We notice that a new directory has been created (output extract produced when unzipping an archive):

```
./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 une_macro.xml

./META-INF:
total 8
-rw-r--r--  1 lrv lrv  1465 Mar   2 00:38 manifest.xml
```

This new directory contains the whole organisation of macros into directories. The new file in this file sub-tree corresponds to the new macro itself. The *manifest.xml* file has been modified in order to declare the presence of any macro and to setup the document accordingly. The set of all macro pathes have been added.

```
<manifest:file-entry manifest:media-type="text/xml"
 manifest:full-path="Basic/Standard/une_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"/>
```

The code of any macro is located between the two following XML tags.

```
<script:module xmlns:script="http://openoffice.org/2000/script"
 script:name="a_macro" script:language="StarBasic">
 ......................
</script:module>
```

Here are located all the informations required to perform an infection of any macro. Let us notice that it is possible to change the macro language. An efficient attack will usefully consider this characteristic.

The size of *content.xml* and *manifest.xml* have increased whereas for the two relevant archives (with and without macro), a significant increase of size has been noticed (1,779 bytes). This corresponds to the additional meta-information which result from the macro creation.

In a macro-virus context, the size of such a virus is liable to have a rather large size ranging from a few bytes to hundreds of kilobytes. Its final size will greatly depend on its sophisticated level and of the script language that has been used to implement it. Let us suppose that the viral code has an average size of 10 Kb. Let us suppose in addition that the resulting archive (that of an infected file) has increased of only one Kb for every 500 added bytes of virus. If we consider, in a first approach, that the size linearly increases,[4] then we notice that consequently the archive's size has increased of 20 Kb. From an operational point of view, this size increase is rather limited, not to say negligible. This feature enables the design of very large-sized viral codes for OpenOffice.org whithout making the final size of the archive explode. This fact is worth considering as far as viral stealth features are considered, especially when infecting large-sized documents.

2.1.4 Storage structure of macros in libraries

Libraries of macros are stored in the *Basic* directory. A library is materialized by a sub-directory which has the same name as the library itself. This sub-directory contains the macros which are linked to this library. Moreover, a library entry is created in the *"script-lc.xml* file and macros themselves are listed in the *script-lb.xml* file.

During this in-depth study, we easily manage to add, erase or modify one or more libraries by means of simple command lines. When infecting macros, any efficient malware attack must modify the files *script-lc.xml* and *script-lb.xml*, in order to avoid error on document opening. By considering this, we successfully manage to << play >> with the general structure of OpenOffice.org documents, without prompting any alert. The probably most surprising issue comes from the fact that any absent macro (when deleted for instance) does not cause any error when opening the document. When editing macros, the deleted macro is listed but cannot be executed since it does not contain any code. OpenOffice considers only the content of the files *script-lc.xml* and *script-lb.xml* and does not make any consistency checking.

Here follows the corresponding dumps (normal library).

– Here is the *Basic* directory content of an OpenOffice.org document:

[4] The experiments we have conducted have shown that this property did hold most of the time.

```
total 8
drwxr-xr-x   6 lrv  lrv  204 Jun 21 18:48 Standard
-rw-r--r--   1 lrv  lrv  400 Jun 21 18:48 script-lc.xml
drwxr-xr-x   5 lrv  lrv  170 Jun 21 18:23 toto

./Standard:
total 24
-rw-r--r--   1 lrv  lrv  335 Jun 21 15:52 Module1.xml
-rw-r--r--   1 lrv  lrv  363 Jun 21 18:47 macros.xml
-rw-r--r--   1 lrv  lrv  348 Jun 21 15:52 script-lb.xml

./toto:
total 16
-rw-r--r--   1 lrv  lrv  335 Jun 21 18:21 Module1.xml
-rw-r--r--   1 lrv  lrv  344 Jun 21 18:22 script-lb.xml
```

– Here is listed below the *script-lc.xml* content (two macro libraries):

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE library:libraries PUBLIC "-//OpenOffice.org//DTD
 OfficeDocument 1.0//E N" "libraries.dtd">
<library:libraries xmlns:library="http://openoffice.org/2000/library"
 xmlns:xlin k="http://www.w3.org/1999/xlink">
 <library:library library:name="Standard" library:link="false"/>
 <library:library library:name="toto" library:link="false"/>
</library:libraries>
```

– Content of the *script-lb.xml* file (information about the content of the *Standard* library):

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE library:library PUBLIC "-//OpenOffice.org//DTD
 OfficeDocument 1.0//EN" "library.dtd">
<library:library xmlns:library="http://openoffice.org/2000/library"
 library:name ="Standard" library:readonly="false"
 library:passwordprotected="false">
 <library:element library:name="Module1"/>
</library:library>
```

Let us recall that we experiment the fact that it is possible to add a macro or even a whole library into an OpenOffice.org document without prompting any alert. A very interesting attack scenario then consists in calling or manipulating any newly added (by a malware) macro directly from another infected macro. Moreover, it is possible to load a library that OpenOffice cannot access directly [10, p. 764].

2.1.5 Password protection

Let us now suppose that the user has protected his document with a password (read and/or write mode). The relevant OpenOffice.org component will deny any access to the document unless the correct password is given upon request.

Comparison of the *content.xml* files (for protected and unprotected documents) shows an unquestionable difference. However, the archive content is neither encrypted nor protected by the password that has been used when saving the document. Only the *content.xml* file is encrypted.

Let us analyse now a protected document with a macro. In some frequent cases, we have noticed some worrying algorithmic problems, as far as password protection is concerned.[5] In these cases, the listing of the relevant archive is the same before and after the protection. Let us have a more particular look at the macro directory. We notice a very important fact: there is strictly no difference at all between the corresponding macro files (documents with and without macros). Consequently, we can deduce that, in these cases, despite the protection through passwords, the code of macros remains unprotected (unencrypted). This implies – and we will later confirm it with the proof-of-concepts codes we have developed – that infecting a password protected document is as easy as for unprotected ones. The integrity of some files in an archive is not taken into account, even by password protection. We manage to replace an encrypted macro with an unencrypted one. For that purpose, we modify the following data: META-INF/manifest.xml, Basic/Standard/<macro_name.xml> and Basic/Standard/script-lb.xml. The password is still required at document opening while the new macro is executed without any security warning – by using trusted macros (see Sect 4.2).

The analysis of the *META-INF/manifest.xml* enables to get some information about the OpenOffice encryption which is moreover described in full in [9].

– Only the following files have been encrypted: *content.xml, style.xml* and *setting.xml*.
– The encryption algorithm is Blowfish [17] in CFB mode. The encryption uses a different seed and initialisation vector (IV) for every different file to protect. Files are compressed before encryption. The initialisation vector is an arbitrary 8-bytes sequence which is base-64 encoded.
– Encryption key is built directly from the password which is provided by the user. The key setup algorithm is the PBKDF2 algorithm [16] (16-bytes salt, base-64 encoded, 1024 iterations). Finally, the derived key is prepended to the encrypted text.
– The document integrity is secured by means of the SHA-1 hash function [7] but there is no integrity checking for the macros which thus can be modified without prompting an alert to the user.

[5] We will not explicit these cases in order to limit their potential exploitation by attackers. We have contacted OpenOffice developers in order to correct these security problems and a joint work has been initiated. We have noticed a very great reactivity.

Here follows an extract of the *META-INF/manifest.xml* files which shows the main aspects of the encryption protocol.

```
<manifest:encryption-data manifest:checksum-type="SHA1/1K"
          manifest:checksum="lCu5udF22D2xpYcJI5iy6RvskTg=">
 .........
<manifest:algorithm manifest:algorithm-name="Blowfish CFB"
          manifest:initialisation-vector="+82Oz7CRbCY="/>
 .........
<manifest:key-derivation manifest:key-derivation-name="PBKDF2"
          manifest:iteration-count="1024"
          manifest:salt="lIIqEW7jBUoOUFmOosK43Q=="/>
 .........
<manifest:file-entry manifest:media-type="text/xml"
                     manifest:full-path="styles.xml"
                     manifest:size="9118">
 .........
<manifest:encryption-data manifest:checksum-type="SHA1/1K"
 manifest:checksum="HUCIFCYfy6Po8rDUGg1jPZuQozO=">
<manifest:algorithm manifest:algorithm-name="Blowfish CFB"
 manifest:initialisation-vector="d+BQMg92Ik0="/>
<manifest:key-derivation manifest:key-derivation-name="PBKDF2"
 manifest:iteration-count="1024"
 manifest:salt="zsx0ImRNhsrhnAEBEDL/ug=="/>
 .........
<manifest:encryption-data manifest:checksum-type="SHA1/1K"
 manifest:checksum="hOQee6IQot1Q5BajsNPsSFKE4dQ=">
<manifest:algorithm manifest:algorithm-name="Blowfish CFB"
 manifest:initialisation-vector="SPXn8aq79Wo="/>
<manifest:key-derivation manifest:key-derivation-name="PBKDF2"
 manifest:iteration-count="1024"
 manifest:salt="efpQcHnHAQBsTRDmxey3Cw=="/>
</manifest:encryption-data>
```

2.2 Analysis of the OpenOffice.org setup

In a first step, let us analyse in depth where and how different possible OpenOffice.org setup may take place. Then we will study the setup structures themselves for every possible operating system. This issue is essential in a malware context since any efficient malware aiming at infecting OpenOffice documents or the application itself will have to precisely locate critical files and resources. This information proved to be very useful in order to design our proof-of-concept codes (see Sect. 4.3). Two kinds of setup are to be considered:

- single user setup,
- muti-user setup.

In every case, the file naming remains the same, with a few exceptions. Only the name of the setup directories are different according to the kind of installation, the operating system in use and its configuration. As far as Unices systems are concerned, as an example, the name of the default directory which contains the users sub-directories may be changed. Such a change would put a check on viruses specially designed for this operating system. Consequently, such a virus has to include a search/identification routine to precisely locate the users directories. A perfect knowledge of the setup environment and configuration is definitively required. This fact still holds for Windows systems since the default setup considers the `C:\Program Files\OpenOffice.org 2.0` directory.

When analysing the OpenOffice.org setup under Unices systems, we have noticed that some other developments or administration environments were present as well: script shell, Java, Python, Perl. It is worth mentioning the presence of many (executable) programs. Every such "software environments" represents a potential danger that may be used as an execution entry point or execution facility for malicious codes. Since all these environments involve only the application layer, they are totally independent from the processor and the underlying operating system. This greatly contributes to increase the scope and portability of the risk attached to the OpenOffice.org suite. Let us now detail the different execution points we have identified.

2.2.1 Script shell

Script shell is rather widely used within OpenOffice.org as it is shown in the following listing:

```
ZZR:/Applications/Bureautique/OpenOffice.org 2.0.app/Contents/openoffice.org/ \
 program$ file * | grep shell
cde-open-url:                   Bourne shell script text executable
configimport:                   Bourne shell script text executable
gnome-open-url:                 Bourne shell script text executable
kde-open-url:                   Bourne shell script text executable
open-url:                       Bourne shell script text executable
python.sh:                      Bourne shell script text executable
sbase:                          Bourne shell script text executable
scalc:                          Bourne shell script text executable
sdraw:                          Bourne shell script text executable
senddoc:                        Bourne shell script text executable
setofficelang:                  Bourne shell script text executable
simpress:                       Bourne shell script text executable
smath:                          Bourne shell script text executable
soffice:                        Bourne shell script text executable
swriter:                        Bourne shell script text executable
unopkg:                         Bourne shell script text executable
viewdoc:                        Bourne shell script text executable
ZZR:/Applications/Bureautique/OpenOffice.org 2.0.app/Contents/openoffice.org/ \
 program$
```

Let us point out some important files:

- *Soffice*: here the script shell performs the following tasks:
  - OpenOffice.org absolute path resolution,
  - retrieval of the main application name,
  - verification of patches installation,
  - Mozilla program path resolution,
  - setting of the `LD_LIBRARY_PATH` environment variable according to the operating system in use,
  - update of the `LD_LIBRARY_PATH` environment variable, of the Java directories paths (use of the *javadlx* program),
  - sets the *Mozilla* path as the environment variable,
  - definition of a variable during the *pagein* program execution, which depends on the calling argument (*calc, writer, math, impress, draw*),

  - launching the OpenOffice.org main program *soffice.bin*, with the suitable calling arguments.
- *Sbase, Swriter, Scalc, Simpress, Sdraw, Smath*: these scripts launch the *Soffice* with the calling arguments. This causes the common part of OpenOffice.org as well as the relevant components (*Swriter, Scalc, Simpress, Sdraw, Smath*) to be launched.
- *python.sh*: this script is used to resolve the *Python* interpreter's path, in accordance with the operating system in use. Moreover, it launches this interpreter with the suitable arguments.

As far as the script shell is concerned, viruses written in interpreted language can be used. There exist many such malicious codes (see [4, Chap. 6] for more details). The most dangerous case refers to a virus that is run whithin a super-user session (particularly, in a Unix environment). A possible approach consists in using appender viruses. However, this infection technique increases the infected host's size. But in the OpenOffice.org context, infecting the *soffice* program remain an ideal solution. Since this latter program has a large size (6,485 bytes), a prepending infection will remain unnoticeable most of the time. Moreover, the *soffice* executable is executing whenever OpenOffice.org is launched.

2.2.2 VBscript

OpenOffice.org also uses this scripting language. This constitutes an additional risk which cannot be neglected. Consequently, VBScript malicious codes – this case is limited to Windows environment up to now – may take benefit of this.

2.2.3 Python

The *Python* environment in the OpenOffice.org setup directory, can be used through the interpreter *program/python-core-2.3.4/bin*, and the relevant source files and *Python* libraries (*program/python-core-2.3.4/lib*). The latter are dynamic ones[6]. Some *Python* scripts (*pythonloader.py,pythonscript.py, uno.py, unohelper.py*) as well as script examples in the */share/Scripting/python* (*Capitalise.py, HelloWorld.py, pythonSample/TableSample.py*) have been identified.

OpenOffice.org uses *Python* by calling the *python.sh* file, which is located in the OpenOffice.org setup file tree, with the code name to execute as an argument. Here follows the relevant part of the *python.sh* script:

```
# set search path for shared libraries
sd_platform= uname -s
case $sd_platform in
  SunOS)
    LD_LIBRARY_PATH="$sd_progsub":"$sd_prog":/usr/openwin/lib:/usr/dt/lib:
      $LD_LIBRARY_PATH
    export LD_LIBRARY_PATH
    ;;

  AIX)
    LIBPATH="$sd_progsub":"$sd_prog":$LIBPATH
    export LIBPATH
    ;;

  HP-UX)
    SHLIB_PATH="$sd_progsub":"$sd_prog":/usr/openwin/lib:$SHLIB_PATH
    export SHLIB_PATH
    ;;

  IRIX*)
    LD_LIBRARYN32_PATH=:"$sd_progsub":"$sd_prog":$LD_LIBRARYN32_PATH
    export LD_LIBRARYN32_PATH
    ;;

  Darwin*)
    DYLD_LIBRARY_PATH="$sd_progsub":"$sd_prog":$DYLD_LIBRARY_PATH
    export DYLD_LIBRARY_PATH
    ;;

  *)
    LD_LIBRARY_PATH="$sd_progsub":"$sd_prog":$LD_LIBRARY_PATH
    export LD_LIBRARY_PATH
    ;;
esac

PYTHONPATH="$sd_prog":"$sd_prog/python-core/lib":"$sd_prog/python-core/lib/ \
  lib-dynload":"$sd_prog/python-core/lib/lib-tk":"$PYTHONPATH"
export PYTHONPATH

PYTHONHOME="$sd_prog"/python-core
export PYTHONHOME

# set path so that other apps can be started from soffice just by name
PATH="$sd_prog":$PATH
export PATH
exec "$sd_prog/python-core/bin/python" "$@"
```

Here again, malicious codes that are implemented in *Python* programming language[7] may take benefit of the *Python* environment and divert it. There is still a lot of work to deeply and precisely understand how *Python* execution really works. Nonetheless, it is obvious that it can be used to make malicious code spread.

2.2.4 Perl

Perl scripts are located in the *share/config/webcast* directory as listed hereafter:

```
ZZR:/Applications/Bureautique/OpenOffice 2.0.app/Contents/Openoffice.org/share/\
  config/webcast$ ls -la *.pl
-r--r--r--   1 lrv lrv  1070 Dec 12 2002 common.pl
-r--r--r--   1 lrv lrv   554 Dec 12 2002 edit.pl
-r--r--r--   1 lrv lrv  1151 Dec 12 2002 editpic.pl
-r--r--r--   1 lrv lrv   342 Dec 12 2002 index.pl
-r--r--r--   1 lrv lrv   743 Dec 12 2002 poll.pl
-r--r--r--   1 lrv lrv   876 Dec 12 2002 savepic.pl
-r--r--r--   1 lrv lrv  1039 Dec 12 2002 show.pl
-r--r--r--   1 lrv lrv   600 Dec 12 2002 webcast.pl
ZZR:/Applications/Bureautique/OpenOffice 2.0.app/Contents/Openoffice.org/share/\
  config/webcast$ file *.pl
common.pl:  Perl5 module source text
edit.pl:    HTML document text
editpic.pl: perl script text executable
index.pl:   HTML document text
poll.pl:    perl script text executable
savepic.pl: perl script text executable
show.pl:    perl script text executable
webcast.pl: perl script text executable
ZZR:/Applications/Bureautique/OpenOffice 2.0.app/Contents/Openoffice.org/share/\
  config/webcast lrv$
```

[6] With Mac OS X (respectively with Unix, Win32), the extension in use is *.dylib* (resp. *.so* and *.dll*).

[7] Some *Python* malicious codes are known, like the *Biennale* virus.

Perl scripts are used to export OpenOffice.org documents in HTML format (*Help OpenOffice.org Webcast Export*). Here again, perl malicious codes are known to exist (e.g. *Intender, Nirvana, SSHWorm, Vich...*) and new malicious codes, intended to spread through OpenOffice.org can be designed very easily.

2.2.5 Asp

The *Asp* is the Microsoft scripting language. It is used mostly to build dynamic Internet webpage. The script file we have identified in the OpenOffice.org environment play the same role as the *Perl* that we have previously presented. The potential viral risk with regards to *Asp* language is not negligible at all.

2.2.6 Java

The first known case of Java virus – *StrangeBrew* – has been identified in August 1998 [18]. However, its infectious power was actually limited. Indeed, it was only able to infect Java applets and Java programs. More recently, the viral threats attached to the Java language have been extensively analysed and Java remains largely concerned with the viral risk, even for the second release of this programming language [15]. The Java environment, in OpenOffice.org, is present in three different ways:

- use of .JAR files, to install .CLASS files. A grand total of 54 such files has been identified within the OpenOffice.org environment:
  - nine are located in the `program/help` directory,
  - forty two are located in the `program/classes` directory,
  - and three can be found in the `share/Scripts` directory.
- through dynamic libraries which are used as an interface between either the Java virtual machine and the OpenOffice.org application (*sunjavaplugin.dylib*) or the Java virtual machine and UNO (*javaloader.uno.dylib, javavm.uno.dylib, libjava_uno.dylib* and *libjava_uno.jnilib*). All these libraries are located in the `program` sub-directory (in the OpenOffice.org setup directory).
- use of Java scripts.

2.2.7 Additional programs

Additional programs may also be successfully diverted or perverted by malicious codes. Here follows the listing of these programs:

```
ZZR:/Applications/Bureautique/OpenOffice 2.0.app/Contents/Openoffice.org/program$
  file * | grep "Mach-O executable"
configimport.bin:                Mach-O executable ppc
gnome-open-url.bin:              Mach-O executable ppc
javaldx:                         Mach-O executable ppc
msfontextract:                   Mach-O executable ppc
nsplugin:                        Mach-O executable ppc
pagein:                          Mach-O executable ppc
pkgchk.bin:                      Mach-O executable ppc
pluginapp.bin:                   Mach-O executable ppc
setofficelang.bin:               Mach-O executable ppc
soffice.bin:                     Mach-O executable ppc
spadmin.bin:                     Mach-O executable ppc
testtool.bin:                    Mach-O executable ppc
uno.bin:                         Mach-O executable ppc
unopkg.bin:                      Mach-O executable ppc
ZZR:/Applications/Bureautique/OpenOffice 2.0.app/Contents/Openoffice.org/program$
```

Although it is not OpenOffice.org-specific, it is worth mentioning that an infection could occur through sophisticated companion codes[8] [4,6]. This is particularly true if the setup has been misconfigured (e.g. file permissions under Unix, among many other cases). The most favourable target in this context is probably the *soffice.bin* executable. It is the main OpenOffice.org application, which is primarily executed by the user.

2.2.8 Quick start function

The OpenOffice.org quick start procedure is the same for MacOS X and Win32 platforms. The OpenOffice.org execution script is launched itself through a program which enables to quickly and directly use the suite.

The Mac OSX case Two cases are possible depending if the OpenOffice.org is active or not. Indeed, the OpenOffice.org application relies on the X11 layer for the interface management. The launch script runs the Apple X11 server[9] and then the OpenOffice.org activation script itself. Thus there exist a potentially huge number of attack scenarii, among which we can cite:

- use of companion infection techniques (non OpenOffice.org specific) which aims at infecting the start script;
- *droppplet* usurpation; the *droplet* is the Mac Os X application which is executed first by the user. Its role is to activate the *soffice.sh* script. When the latter is absent, the OpenOffice.org cannot be launched;
- X11 usurpation.

Cas Win32 There is no noticeable difference with Mac OSX.

2.2.9 Miscellanous interesting elements

The *share* sub-directory in the setup directory contains two sub-directories. They contain or are liable to

[8] We suppose that basic companion infectious techniques are still efficiently managed by most of the antivirus programs.

[9] This may be either X11 itself, *XDarwin* or *OroborOSX*.

contain some resources that can be diverted or perverted by malicious codes:

basic This directory contains standard macros for the application. Let us mention that we also find the same *basic* directory in the user '*user_path'/.openoffice.org2/user* directory. It contains itself the OpenOffice.org standard macros for the user's disposal.

psprint/driver It contains the Postscript files with respect to the printers use and/or management.

uno_packages This directory is liable to contain some useful information about the configuration as well as the UNO packages that have been put in addition to OpenOffice.org.

2.3 Analysis of the OpenOffice.org functionalities

The study of the OpenOffice.org functionalities enables to identify some execution transfer points, some task schedulings or some event-related execution point (which are similar to those identified for the VBA programming language in the Microsoft office suite). Moreover, both the open technical documents and the OpenOffice.org help have proven to be very useful in identifying all the execution points that we considered to perform computer infections.

2.3.1 Packages

A new functionality has been added to OpenOffice.org: packages. Packages are a powerful feature that enables to add UNO components[10], a new configuration, *OOoBasic* or *dialog*[11] libraries and packages archives in ZIP format (they contain one or more packages). However working with package requires to have super-user rights in a multi-user setting.

A management utility (*Package manager*) is also available within the application itself. This *Package manager* can be launched through the OpenOffice.org application directly from the *Tool* menu. In single user setting (*My packages*), the management directory is *$HOME/.openoffice.org2/user/uno_packages* whereas in a multi-user configuration (*Openoffice.org Package*), the management directory is *share/uno_packages*.

Packages are by nature liable to be a good entry point for malicious codes (initial infection), since they are in ZIP format.

2.3.2 Macros

Macros can be written in many languages as shown in Fig. 1. *OOoBasic, JavaScript* and *BeanShell* are the only ones which are supported internally. Higher level languages (like *Python* and *Java*) may be used through the API Project [10]. It is possible – and easy – to link a macro with an OpenOffice event (use the *Tools/Customize* menu). By default, there are at most 16 such events as shown in Fig. 2. Moreover, these events may be saved either at the document level only, or at the application level. It is also possible to create a (keyboard) shortcut key to directly launched an event-related macro (see Fig. 2).

The execution of the *DicOO* macro enables to install a dictionary in any available language, directly from the Internet. This implies the existence of resources within the macros themselves or within the OpenOffice application, in order to use the network layer. As far as the Internet is concerned, it is also possible to assign a macro to a hypertext link or to e-mail reception (see Fig. 3). The OpenOffice help also mentions the possibility to link a macro to an image, to a form control or to a *OOoBasic* dialog control. This is particularly interesting since an event-activated macro is a very good execution point. The existing possibilities, in terms of malicious execution are quite boundless. From this point of view, the OpenOffice.org is far less secure than its commercial counterpart (*Microsoft Office*).

2.3.3 Adding objects

Any OpenOffice document can include objects of various kinds: OLE objects, plug-ins, video, sound, Java applets, forum, graphics... Among these documents, we have to consider two different categories:

- passive or inactive documents; they are non executable documents. Nonetheless they may include executable data, which can be activated by *k*-ary malicious codes [4, Chap. 4] and [6, Chap. 3].
- active documents; contrary to inactive documents, they contain data that can be executed directly.

2.3.4 Modifying OpenOffice menus and message boxes

Another very worrying possibility for malware is the capability to modify the different entries in OpenOffice application menus or message boxes. This enables to manipulate and fool any user very efficiently. The number of such modifications are quite infinite. As an example, a virus could invert the button *Enable Macros* and *Disable Macros* in the OpenOffice security warning

[10] These are extra compiled modules. They can be of infectious nature, in a context of viral attacks.

[11] It is a UNO library, which belongs to the `[MODULES:com.sun.star.awt]` module and which enables the GUI manipulation, according to the "*Model-View-Controler*" model.

message box, while the respective functionalities remain unchanged. When choosing the *Disable Macros* button, the user in fact enables them.

To illustrate the menu modification with a more complex example, let us consider the example presented in Fig. 4. The relevant configuration file *GenericCommands.xcu* is located in the *share/registry/data/org/openoffice/Office/UI* directory. Here is the content corresponding to the left part of Fig. 4:

```
<node oor:name=".uno:MacroOrganizer?TabId:short=1" oor:op="replace">
  <prop oor:name="Label" oor:type="xs:string">
  <value xml:lang="en-US">Organize ~Dialogs...</value>
  </prop> </node>
<node oor:name=".uno:ScriptOrganizer" oor:op="replace">
  <prop oor:name="Label" oor:type="xs:string">
  <value xml:lang="en-US">Organize Macros</value>
</prop> </node>
<node oor:name=".uno:RunMacro" oor:op="replace">
 <prop oor:name="Label" oor:type="xs:string">
 <value xml:lang="en-US">Run Macro...</value>
</prop></node>
```

After modification (part right of Figure 4) we have:

```
<node oor:name=".uno:MacroOrganizer?TabId:short=1" oor:op="replace">
 <prop oor:name="Label" oor:type="xs:string">
 <value xml:lang="en-US">Organize Macro</value>
</prop></node>
<node oor:name=".uno:ScriptOrganizer" oor:op="replace">
 <prop oor:name="Label" oor:type="xs:string">
 <value xml:lang="en-US">Organize Dialog...</value>
</prop></node>
<node oor:name=".uno:RunMacro" oor:op="replace">
 <prop oor:name="Label" oor:type="xs:string">
 <value xml:lang="en-US">Run toto...</value>
</prop></node>
```

2.3.5 Miscellanous data

Some other data are to be considered since they are implied in mechanisms that may be used by malicious codes to spread. They are in fact particular directories, which are worth considering:

- the *modules/org/openoffice/office/Common* directory; it contains the configuration file of the different menus which are in any OpenOffice.org components;
- the *config/soffice.cfg/global/accelerator* directory; it contains the definition of shortcuts which are global to the application, by default, according to the setup language, a given event or action. It also defines language specific shortcuts, this for every different components (*config/soffice.cfg/modules/*);
- the *modules/org/openoffice/Setup* directory; it contains the name of any UNO services, depending on the component in use. For every such OpenOffice.org component, UNO services enables OpenOffice.org to extend its functionalities.

These three configuration files may be easily modified by a malicious code in order to setup its own execution. The main interest is to fool the user by changing menu or shortcut names in order to directly activate the selected event and thus make the malicious code activate.

Another point is worth mentioning. According to the OpenOffice SDK, it is possible to modify the user interface in order to add, for instance a new UNO component, by means of the *pkgchk* utility [10, p. 230]. When doing this, a malware must comply with the module archtitecture (as defined in [10, p. 193]). Moreover, it is possible to perform a "hot desactivation" of any OpenOffice command [10, p. 272]. Such a facility could inevitably be used by efficient OpenOffice malware.

2.4 Configuration analysis

The deep and exact knowledge of a given environment is essential. It enables a malicious code to optimally adapt to the target environment or an attack to optimally develop such a code. That is the reason why we are now going to analyse the OpenOffice.org configuration aspects.

Modifying the environnement will take place during the infection's initial step (*primo infection*). In the OpenOffice.org context, we have to consider the two possible main settings, whatever may be the underlying operating system: single user or multi-user setting. Using information presented in Sect. 2.3, the analysis of the setup directory allows to consider the following conjecture: the configuration is stored in the *openoffice_install_path/share/registry* directory. However, since OpenOffice.org may be set up in two different ways, this implies that in multi-user setup mode, the configuration itself must be stored in the user's directory.

2.4.1 Application configuration

As far as the OpenOffice application is concerned, the *Tools/Option* and *Tools/Customize* toolbar menus are involved. The main configuration points that are relevant for our study are:

- **Paths:** We can manage the whole set of paths used by the OpenOffice application. The *Basic*, *My Documents* and *User Configuration* are of particular interest. The relevant information may be used by any sophisticated malicious code which intends to use, modify and control the target environment in order to act or spread.
- **Security:** This point involves OpenOffice security options: macro security levels, application security levels and document sharing options. In particular, there exist a "trusted source" option which enables to define macros directories that can be trusted.

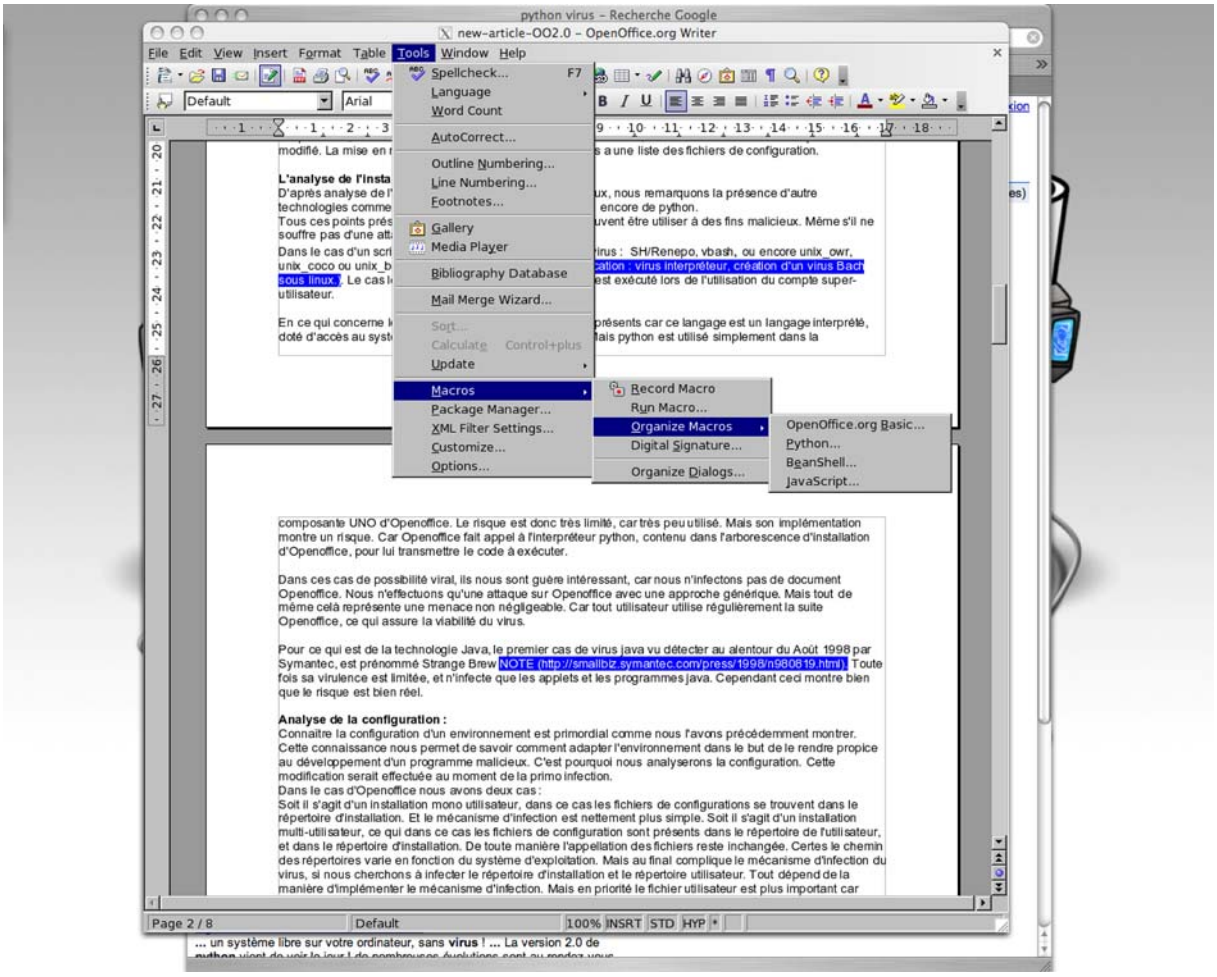

Fig. 1 Macro languages listing

– **Java:** Java execution environment may be modified as well. The relevant information are the virtual machine path and the class files directories. They can be used by any Java malicious codes [15].
– **Internet:** It is also possible to configure Internet connections management. In this particular case, we have to specify the parameters of: the proxy server, the e-mail manager, the Internet browsing and the Mozilla plug-in. These information may be modified according to the infection scenarii that we present in Sect. 4. They allow a free exchange towards or from the Internet. Thus, we can imagine a proxy server substitution. Data theft may then occur whenever OpenOffice connects to the Internet.

The default configuration files are (for every user):

```
C:\Program Files\OpenOffice.org 2.0\share\registry\data\org\openoffice
C:\Program Files\OpenOffice.org 2.0\share\registry\data\org\openoffice\
   Office\Common.xcu
C:\Program Files\OpenOffice.org 2.0\share\registry\data\org\openoffice\
   Office\Writer.xcu
C:\Documents and Settings\<Utilisateur>\Application Data\
   OpenOffice.org2\user
```

2.4.2 Single-user setting

Configuration files are located in the setup directory. Consequently, modifying the environment is trivial. Potential infection mechanisms are trivial as well.

2.4.3 Multi-user setting: looking for configuration files

Relevant information have been obtained by analysing the difference between files. This approach enables to identify all modifications. Here follow the main points of interest:

– configuration mechanisms are XML-based. This implies on one side that the OpenOffice.org is dealing with text data according to specifications in a XML-scheme file header (XCS extension; the XML file has itself an XCU extension). In another side, the XML file must be valid with respect to a XML scheme. The default schemes (for validation purposes or as data

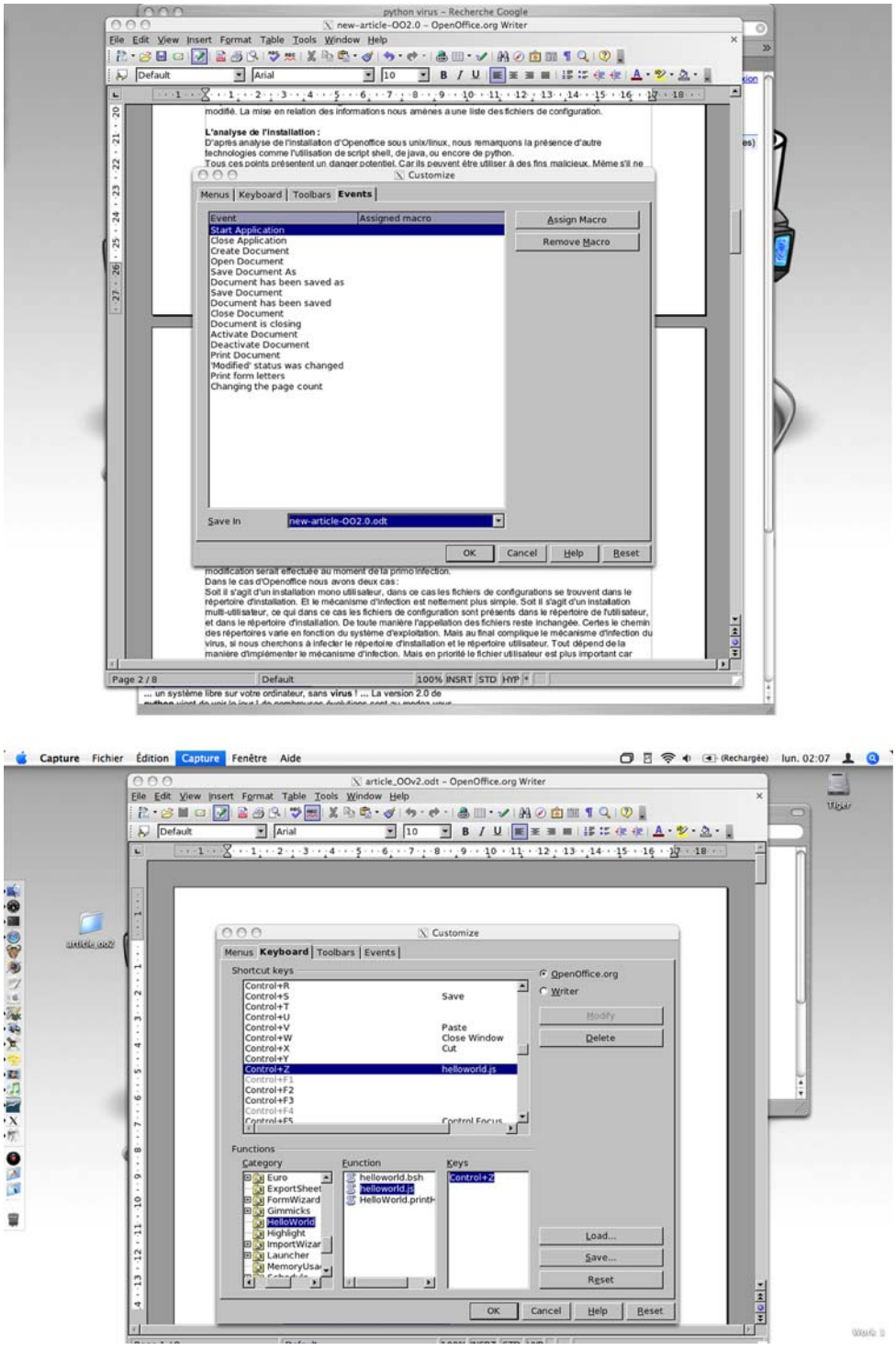

Fig. 2 OpenOffice.org events (left) and linking a macro with an event (right)

sources) are located in the *<open_office_install_path >/share/registry/scheme* while the configuration data themselves are located in the *<open_office_install _path>/share/registry/data* directory.

– Whenever the configuration is modified, changes are saved in the user's directory. This particularly involves the macro security parameters (*Common.xcu* file).

– The application security level is defined in the <openoffice_install_path>/share/registry/schema/org/openoffice/Office/Common.xcs file as far as the *Common.xcu* file is concerned. The different possible values of security level are summarized in Table 1. We will explain in Sect. 4.1 what trust in OpenOffice.org really is. The use of an invalid security level value makes the document unoperable. Security

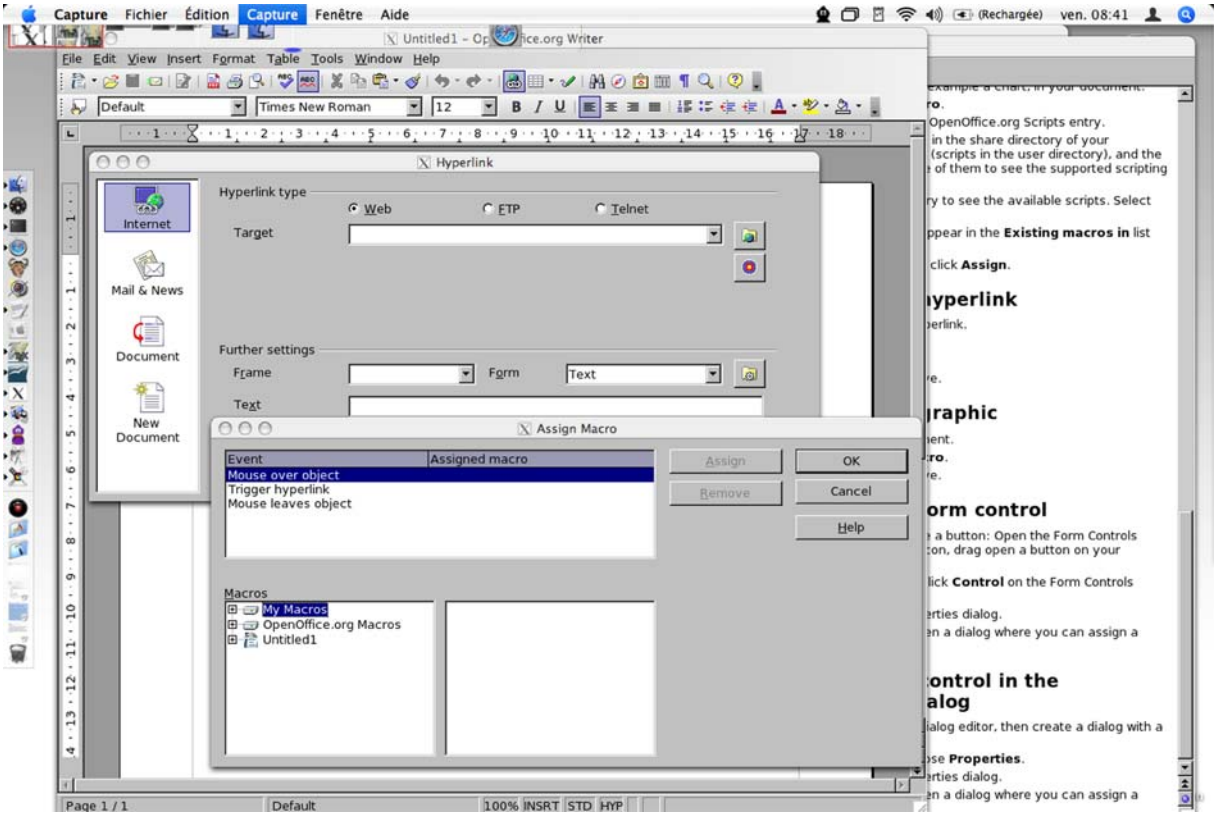

Fig. 3 Assigning a macro to a hypertext link

management is done in the following way: OpenOffice.org considers a user's directory to contain the user-specific configuration. Then, it takes default values contained in the setup directory and replace them with the user's one.

- Macro security parameters are also stored in the *<openoffice_install_path>/share/schema/org/openoffice/Office.Common.xcs* file. Let us mention the fact that modifying them is possible according to the installation directory permissions in force, only. Lastly, plug-in execution is activated by default (the relevant value is set to "TRUE").
- XML schemes that have been identified as critical are hereafter listed:
  - Common.xcs : scheme file of the data *Common.xcu* file,
  - Event.xcs :
  - Java.xcs :
  - Writer.xcs :

  As for the data XML files, the critical ones are:
  - Inet.xcu,
  - Setup.xcu,
  - Office/Common.xcu,
  - Office/ProtocolHandler.xcu,
  - Office/Scripting.xcu,
  - Office/Security.xcu,
  - Office/WebWizard.xcu,
  - Office/Writer.xcu,
  - Office/UI/.

  The study of all these files has shown that they are organized in "themes". Moreover, data files contains only a minimal amount of information. This can be explained by the fact that the *<open_office_install_path>/share/registry/schema/office* directory stores the application default parameters. An in-depth analysis of the installation directory, the following directories appear to be essential:
  - config/soffice.cfg/global,
  - config/soffice.cfg/module,
  - registery/data/org/openoffice.

  They indeed contain some other configuration parameters whose value are liable to be modified in order to enable malicious code execution.

Configuration files are located in the user's home directory as well as in the OpenOffice installation directory. The latter is shared by any user as far as configuration values are concerned. If some malicious code infects and modifies the user's setup, the spread and/or action of this code will be limited to the user's space. It is also

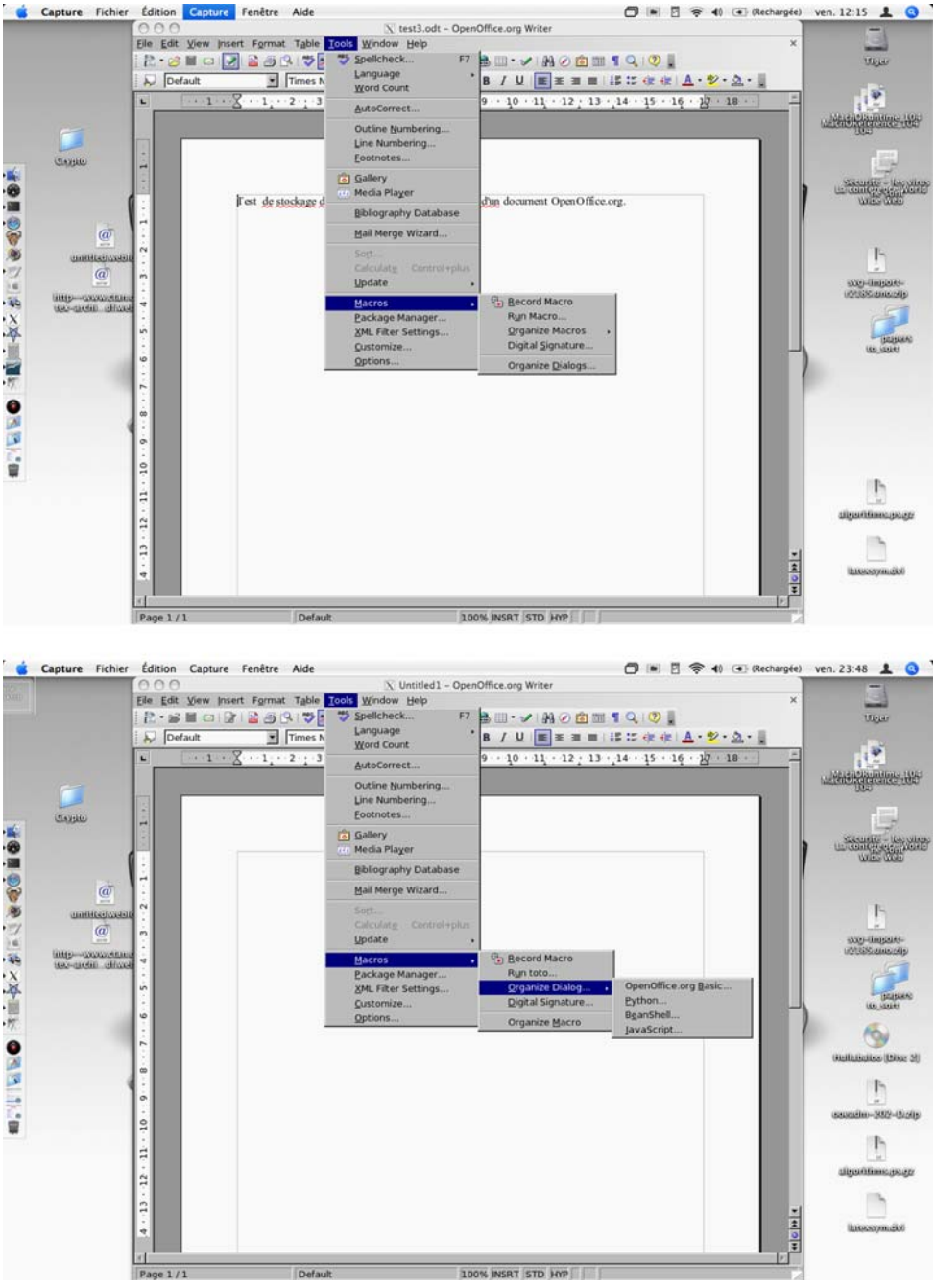

Fig. 4 Modification of OpenOffice menus (before at left; after at right)

Table 1 Macros security levels

Value	Security level	Comments
0	LOW	(minimum) Every macro is executed without prompting
1	MEDIUM	Prompt to execute macros that are not trusted macros
2	HIGH	Only trusted and "signed" macros are executed
3	HIGHEST	(maximum) Only macros in trusted directories are run

possible to infect every other user's space that is currently working on the system (logged-in). However, an efficient rights management (e.g. Unices systems) should prevent such a spread unless if the super-user is itself infected. On the contrary, if default installation files would be infected, every new user would be automatically infected or at least his environment could be more easily prone to infection.

In this section, we have presented the files which are essential as far as configuration is concerned. These files may be implied and manipulated during any infection process. Consequently, the viral risk attached to the OpenOffice.org application is very critical. A good solution could be to encrypt and sign any configuration files (use of public key cryptography). Thus any unlegitimate modification would become very difficult not to say impossible, provided that the key management and the surronding computer security are efficient.

3 Validation through proof-of-concept malicous codes

In the previous sections, we have identified the main files, mechanisms and events that could or must be involved in an infectious process. The number of possibilities that could be exploited by any attacker are infinite. In order to simply illustrate and operationally validate the previous in-depth study, we now are going to present proof-of-concept codes by considering event-related macros only, and how to exploit, pervert or divert them to make a malicious code work. Due to lack of space, other possibilities will not be exposed.

3.1 Use of macros for infection purposes

In the OpenOffice.org 2.0.x suite, a huge number of potential malicious uses of macros does exist. Most of them have no equivalent in the Microsoft suite. Scripting modules enable to consider new programming languages for writing efficient and powerful macros. Let us recall that OpenDocument features and document format allow to store several macros written in different languages in any document. Macros can be written in:

- Beanshell: Java light scripting language;
- JavaScript: object-oriented scripting language. Mostly used in webpage;
- Java: high-level object-oriented programming language which includes an integrated execution environment. Its syntax is very close to the C syntax and its most powerful feature lies in the fact that it can be ported to any operating system;
- Python: interpreted, multi-paradigm programming language. It enables an imperative structured programmation and an object-oriented, event-oriented approach.
- OOoBasic: denoted *StarBasic* as well. This programming language is very close to the Microsoft VBA but they are not compatible – up to now. Moreover, it is possible to invoke the OpenOffice.org API, directly from macro. This can be seen as an extension of the resource environment.

For every possible language (Beanshell, Javascript, Java, Python, OOobasic), we have considered and identified the resources that must be required to design a generic infectious algorithm. Only the *OOoBasic* will be presented in Sect. 4 in order to validate and fine tune the infection capabilities attached to OpenOffice.org. Of course, these proof-of-concepts are fully transposable to the other OpenOffice.org programming languages.

3.2 External programs

As we have previously pointed out in Sect. 2.2.7, the presence of external programs within an application is very interesting to consider. It represents a privileged environment for companion viruses (at least for sophisticated ones). As far as OpenOffice.org is concerned, the external programs which are involved by OpenOffice.org's use are listed hereafter:

- e-mail client,
- configimport.bin,
- gnome-open-url.bin,
- Perl and Python interpreters, as well as the Java virtual machine,
- javaldx,
- msfontextract,
- nsplugin,
- pagein,
- pkgchk.bin,
- pluginapp.bin,
- setoffice.bin and soffice.bin,
- spadmin.bin,
- testtool.bin,
- uno.bin and unopkg.bin.

Companion infection techniques are not OpenOffice-specific but it is very important to recall the existence and the feasability of such malicious codes even for OpenOffice. The most critical programs – in other words, the programs which are very frequently run by OpenOffice – are: soffice, pagein, the Java virtual machine and the Perl and Python interpreters.

3.3 Scripts

As previously seen, the OpenOffice.org application is launched from a script, which is itself run by an executable file. This execution chaining mechanism is particularly favourable to viral or infectious actions. But the most important aspect lies in the main component: a *sh* (Unices systems) or *VBScript* (Win32 systems). In this particular case, a prepending or appending infection technique is also possible, since OpenOffice.org does not perform any internal, integrity checking. Moreover, overwriting malicious codes could randomly replace existing, legitimate internal or external programs in order to fulfill their action. In this case again, the *soffice* script integrity is hurt and only an integrity checking tool (e.g. *Tripwire*) could prevent any such approach.

Let us now consider the infectious threats which are OpenOffice.org-specific by looking at the macro-related infectious capabilities.

4 In-depth analysis of the OpenOffice.org viral risk

As for the Microsoft Office suite, macros are supposed to be a potential help to the user in his every-day use. Now, languages of macro exhibit a major drawback due to their power: attackers can put malicious macros in any apparently innocent office document.

As far as the OpenOffice.org suite (release 2.0.x) is concerned, new potentialities have been developed and added for macros writing. As an example, the scripting module enables to use new additional programming languages. Such as the different data contained in an OpenOffice.org document, macro source codes is located in the ZIP structure of the document. Macros written in different programming languages may be stored at the same time in a document: BeanShell, Java, JavaScript, Python and OOoBasic.

Without loss of generality, our study of the viral threats with respect to the OpenOffice.org suite will consider only the OOoBasic language. However our approach and results can be fully transposed to other macro writing languages proposed in the OpenOffice.org suite. OOoBasic is a very modular, powerful, enhanced programming environment which enables to develop very sophisticated applications and macros. We can read, write, modify... any OpenOffice.org documents through calls of its (unique) API [13]. In this respect, OOoBasic is rather more efficient than Java language. Moreover, OOobasic applications can act at the operating system level and interact with it. To summarize, the OOoBasic language is fully comparable with the Microsoft Office Visual Basic for Applications, despite the fact that they are not compatible. A Microsoft Office macro virus is not functional with respect to OpenOffice.org and conversely – up to now.

4.1 How macros work

OpenOffice.org macros, as mentioned before, may be written in different programming languages. Every such language considers the same management structure. All macros are stored in "Libraries", each of which containing "modules" or macro scripts and if required "Dialog boxes" refering to the macros themselves.

The general structure of a macro is illustrated by pseudo-code which here follows:

Table 2 General structure of an OpenOffice Macro

```
REM Macro displaying a "Hello" message box
Sub DisplayHello
    Info = "Hello"
    MsgBox info
End Sub
```

There exists three different locations for libraries (see Fig. 5):

- libraries attached to the documents themselves;
- user's libraries located in "*My Macros*";
- OpenOffice.org standard libraries located in "*OpenOffice.org Macros*".

As far as security is concerned, the macros may be activated:

- either automatically if they are stored in a trusted location or if they come from a trusted source;
- or directly by the user according to the security level in force ("*low*", "*medium*", "*high*" and "*very high*").

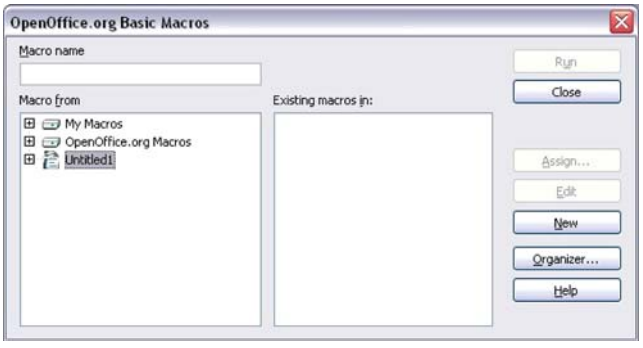

Fig. 5 Macros and libraries

Recorded trusted sources are used only from the "medium" security level. They apply only to macros created or imported by the user itself. In release 2.0.0, trust applies only to pointed directories, excluding the sub-directories contrary to release 2.0.1 which extends trust to sub-directories (as in releases 1.0 and 1.1). Moreover, trusted sources does not apply to OpenOffice.org standard macros. Lastly, any OpenOffice.org document may be numerically certified with one or more certificates. During the certification process, OpenOffice.org computes a digital signature for the document (*document signature*). In a similar way, any macro may be numerically signed as well. However, the document signature does not depend on the macros' signatures themselves, if any.

The "`C:\Program Files\OpenOffice.org2.0\share\basic\...`" directory contains the "*My Macros*" and "*OpenOffice.org Macros*" libraries. They are part of the above-mentioned trusted sources, by default. Once an OpenOffice.org application is opened, the macros contained in these libraries are automatically executed without alerting or prompting the user, this whatever may be the security level in force. This means that macros located in these directories – either by the user or by any malicious code – are *ipso facto* considered as trusted macros.

OpenOffice.org 2.0.x standard macros – which are trusted macros as well by default – are written in OOoBasic. They are located in the `<Office\-Path>\share\basic\...` directory. They are eleven in number:

- Depot,
- Euro,
- FormWizard,
- Gimmicks,
- ImportWizard,
- Launcher (contains FontOOo and DicOOo; see later),
- Schedule,
- ScriptBindingLibrary,
- Template,
- Tools,
- Tutorials.

Standard macros which are contained in these eleven groups are loaded at OpenOffice.org execution. They are fully comparable to the Microsoft Office suite *Auto-Exec* macro [4, Chap. 4]. Whereas the Microsoft Office considers a unique such macro, OpenOffice.org considers many. On that particular point, OpenOffice.org offers more opportunities to malware writer than Microsoft Office.

Without loss of generality, we will consider (see Sect. 4.3) the particular *DicOOo.xba* and *FontOOo.xba* macros (located in the `<Office\-Path>\share\basic\Launcher\...` directory) for our proof-of-concept. These two macros (see Fig. 6) are written in OOoBasic. Their role is to launch the installation of either the dictionary or additional character fonts respectively.

All previous facts being considered, a first, trivial infection scenario obviously arises which consists in replacing one or the other macro by a malicious one, or simply insert malicious code in any of them. This approach yet simple proved to be very efficient.

4.2 Event or service-activated macros

Any macro can be associated with one or more functional events. As a non exhaustive list of such events, we have:

- application start or closing,
- document creation, opening, activation/desactivation, saving (any kind) or closing,
- document printing,
- status modification,
- form letter printing,
- modification of the number of pages...

The case of event-activated macros is comparable to the case of Microsoft Office *auto-macros* [4, Chap. 4].

In addition to events, OpenOffice.org macros can call upon various services. Either of them can be diverted or hooked by malicious codes in many – nearly infinite – different ways (non exhaustive list):

- opening of an additional file,
- opening of a new empty OOoWriter document,
- opening of a new empty OOoCalc spreadsheet,
- call of macro by another macro,

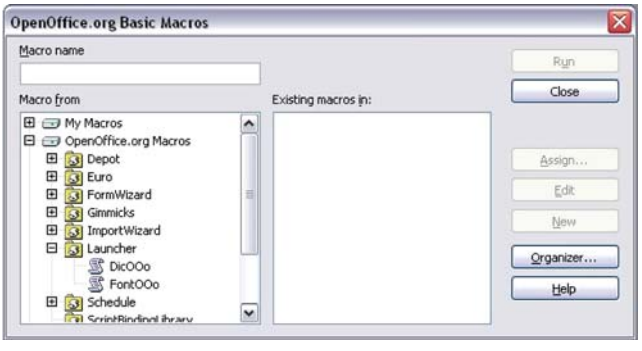

Fig. 6 *DicOOo.xba* and *FontOOo.xba* macros

– button-activation of a macro,
– shortcut-call of a macro (including directly from the toolbar),
– call of a macro through a hypertext link,
– macro call from a spreadsheet,
– macro call from another library,
– macro call during data validation,
– API call from other languages,
– external program call through OLE...

According to the security level in force, a security warning window may be displayed:

– The "*low*" security level enables any macro, whatever may be the document origin. It is the worst level since it is very dangerous as far as security is concerned. The user is absolutely not warned about the presence or not of macros.
– The "*medium*" security level warns the user and enables macros only if the current document is not located in any trusted directory. It is the default security level.
– The "*high*" security level authorizes the execution of macros only if the current document is located in a trusted directory. If not, macros are disactivated and the user is warned by OpenOffice.org.
– The "*highest*" security level takes into account only documents stored in trusted directories. Outside from these directories, macros are systematically disactivated, whether signed or not.

A very critical point in OpenOffice.org 2.0.x is the automatic execution of the macros which are contained in the "*My Macros*" and "*OpenOffice.org Macros*" directories since no security warning is displayed in any security level in force.

All these facts being considered, a number of proof-of-concept viruses have been designed and tested in order to practically and thoroughly evaluate the actual level of risk, as far as viral or computer infection is concerned. The source code of the proof-of-concept which have been developed will not be disclosed for obvious security and responsability issues[12]. Only the most significant algorithmic aspects of these codes will be detailed.

4.3 OpenOffice.org viral risk made concrete

We have designed a number of operational viral strains with respect to OpenOffice.org documents, for research and technical validation purposes. These strains have been successfully tested and they allow to claim that the viral hazard attached to OpenOffice.org is at least as high as that for the Microsoft Office suite, and even higher when considering some particular aspects. These viral codes have been designed and tested independently of the underlying operating system. This implies a total portability of these codes and hence a generalised risk. Lastly, we did not intend to develop "true" viruses with sophisticated and dangerous functionalities. Since the goal was to prove, through a scientific approach, that viral hazard actually exists for OpenOffice.org, only basic yet efficient codes have been developed without any payload. But the reader must be aware that writing dangerous codes for OpenOffice.org components is possible.

From an algorithmic point of view, the different viral strains include the general viral function hereafter listed (see [4, Chap. 4]):

1. primo-infection function: it performs the initial infection step of the OpenOffice.org environment. During our tests, this has been realised through a simple office document email attachment. Alternatively, a simple exchange of office documents can be considered;
2. search for target to infect function;
3. overinfection control function. The code must infect an already infected target;
4. infection function;
5. stealth and virulence control functions.

It is worth noticing that no other anti-antiviral functions have been included (polymorphism or armouring [5]) since these functionalities are not specific to OpenOffice. However, OOobasic or any other programming language in OpenOffice.org are powerful enough to implement very sophisticated polymorphic or armoured functionalities. As far as stealth is concerned, we have developed and tested such anti-antiviral capabilities but only by considering characteristics that are specific to the OpenOffice environment and/or capabilities.

4.3.1 Diverting or perverting events

For this first viral strain, denoted `OOv_s1` – the "*document opening*" and "*document closing*" have been diverted (or hooked) by means of macros associated to these events. This approach is quite comparable to using *AutoOpen* or *AutoClose* macros under Microsoft Office [4, Chap. 4].

The main steps performed by the `OOv_s1` virus here follow:

[12] The source codes are freely available upon written request only for strictly identified IT security professionals. The request must be sent to the corresponding author.

1. An e-mail with attachment (OpenOffice document infected by `OOv_s1`) is sent to the victim.
2. The opening of the attached document activates the malicious macro which is associated to the corresponding event. The primo-infection step then occurs in a first phase.
3. The malicious macro puts an additional malicious code $\mathcal{C}$ on the hard disk.
4. The malicious code $\mathcal{C}$ is launched when the current document is closed (the macro associated to the "document closing" event is diverted).

4.3.2 The `OOv_s1_f`*: a stealth variant of* `OOv_s1`

This second proof-of-concept code considers some particular OpenOffice characteristics in order to hide its presence and action. The stealth properties are achieved by hiding the malicious code inside the *DicOOo* or *FontOOo* macros. Thus, the malicious code is launched during the installation of *DicOOo* or *FontOOo*, by the user. There exist many other possibilities.

The main steps performed by the *OOv_s1_f* virus here follow:

1. An e-mail with attachment (OpenOffice document infected by `OOv_s1_f`) is sent to the victim.
2. The first time the document is opened, the macro associated to this event is executed. The primo-infection takes place in this initial step.
3. The malicious macro infects the *DicOOo* macro with an additional malicious code $\mathcal{C}$.
4. At the *DicOOo* setup, the malicious code $\mathcal{C}$ is launched.

This technique is similar to that used by many existing Miscrosoft macro viruses which use macros with legitimate Office components command names (*usurping macros*; see [4, Chap. 4] for details). The *FontOOo* macro can be used instead.

4.3.3 Using malicous documents templates

A third viral proof-of-concept code, denoted `OOv_s2` perform its action through an infected template. To illustrate the algorithmic capabilities of the OOobasic language, we consider two viral codes which act in cooperation one with another (this approach can be generalized to k-ary codes; see [6, Chap. 3] for the concept of k-ary codes). The purpose of this approach is to obtain an increased efficiency as well as sophisticated stealth properties. Indeed, each subcode contains only a part of the whole virus.

The goal here was to enable the viral spread from an already infected OpenOffice environment into any other non infected OpenOffice document and thus towards any non infected environments when considering the exchange of office documents. The `OOv_s2` perfectly corresponds to any existing Miscrosoft macro-virus, since the `OOv_s1` and `OOv_s1_f` only considered a non self-reproducing (*epeian* infection; for example a logical bomb or a Trojan horse).

The very first step of this attack consists in first modifying the current OpenOffice configuration. That is the reason why it is required to use one or more infected templates to efficiently perform the attack. By modifying the configuration files – through a two-level approach – it is thus possible to involve far more sophisticated infectious macros that will be activated whenever any document is opened. In particular, more complex infection mechanisms may be considered than simply diverting/perverting event-related macros 4.3.1.

The main working steps of *OOv_s2* here follow (see Fig. 7):

1. Two e-mail attachments P_1 and P_2 (sent together or in two different successive e-mails)[13] are sent to the victim, in such a way that P_1 is always open before P_2.
2. The first infection step occurs (attachment P_1): a template file which contains the malicious macros is setup.
3. The second infection step occurs (attachment P_2): malicious event-activated macros are put in trusted locations. Consequently, these macros will *de facto* be considered as "trusted macros". They will be thus transparently executed (the user is not warned of their presence). Many various actions can then be performed. We have implemeted and tested the following ones:
   - recording of a new module (new library) containing macros, in the "My Macros" directory. Whenever a user loads and opens a document, this library is automatically loaded as well without any security level checking;
   - a macro M_1 replaces the function attached to the F5 key (any other function may be replaced in such a way);
   - a macro M_2 calls for the templates previously installed. Any new document or any other document handled by the user will be automatically

[13] Of course, this scenario may be played with a single infected document. The corresponding malicious code is simply slightly more complex.

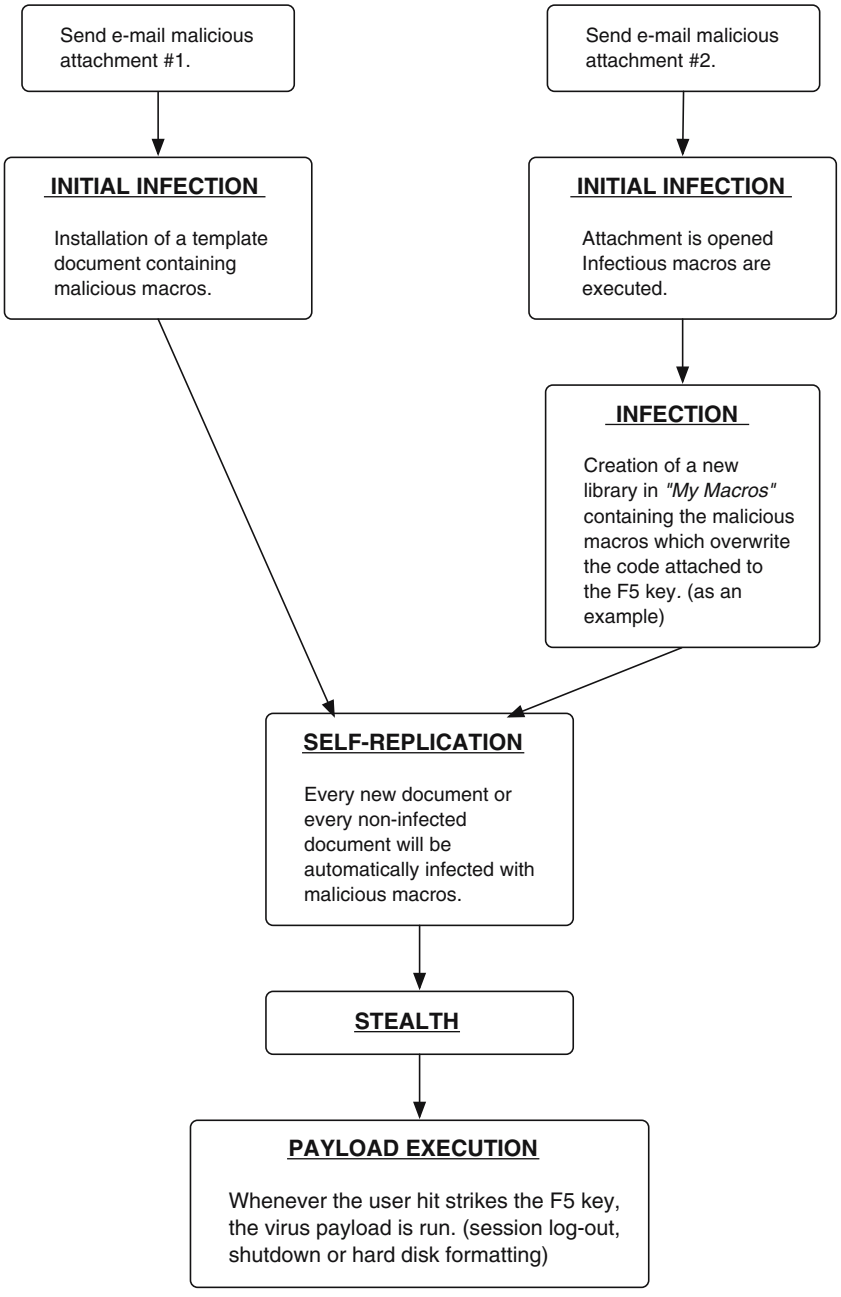

Fig. 7 Organisation Chart for the *OOv_s2* viral strain

infected, thus making the virus spread. It is worth noticing that a OOo macro seemingly in Microsoft VBA may be used inside a WORD document to fool the user.

4. Each time the user presses the F5 key, the payload is activated (session logout, computer shutdown, data destruction, formatting...).

Stealth features have been implemented by means of command lines hidden in legitimate pre-existing macros. The critical point is to thoroughly manage the potential function errors that may betray the presence of a virus (test of exit codes).

4.3.4 A stealthier version of OOv_s2*: the* FinalTouch *OpenOffice virus*

This last proof-of-concept is probably the most efficient and powerful one that we have developed. It illustrates and summarizes many of the capabilities that are likely to be used or diverted by any efficient OpenOffice

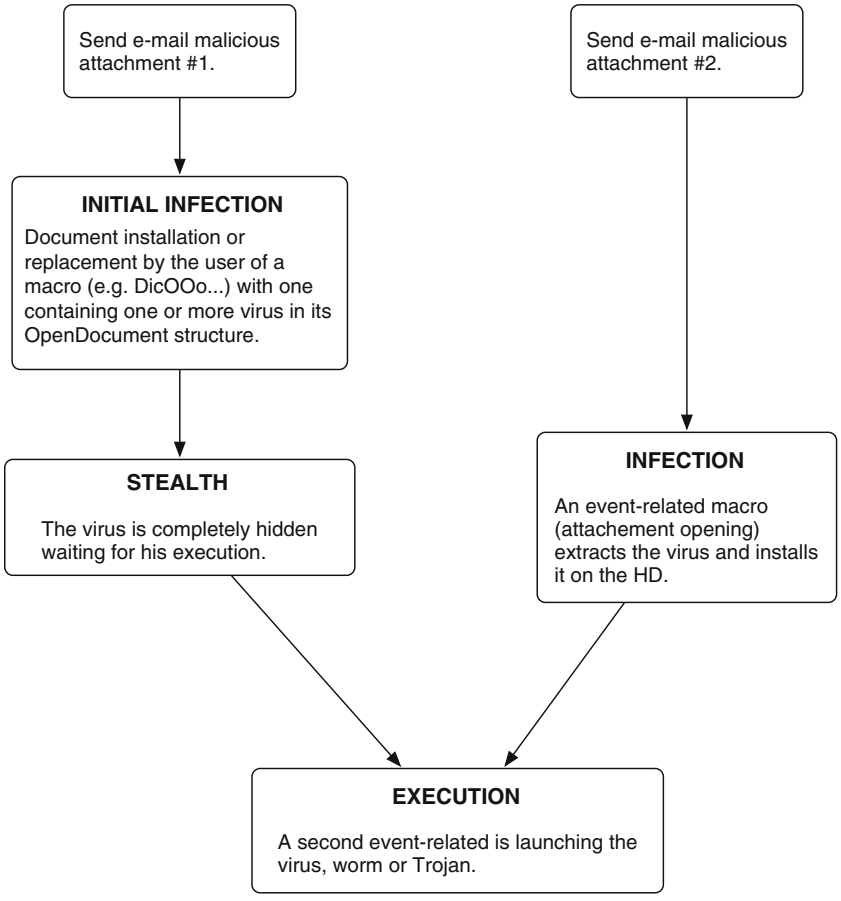

Fig. 8 Organisation Chart for the *FinalTouch*virus

malware. The main working steps of the *FinalTouch* virus are (see Fig. 8):

1. A first e-mail attachement is sent. The attachment contains a hidden (stealth) executable (e.g. *virus.cmd*). This executable is totally invisible and undetectable unless we manually unzip the attachement archive. This is performed by manipulating the OpenDocument structure organization as previously exposed in this paper. In our experiment, the e-mail tells the attachement is a new version of the *DicOOo* utility. Many other scenarii (using or not e-mail) are possible. The user has to replace the old *DicOOo* file by the new one.
2. A second e-mail attachement is sent. It contains an infectious macro. Whenever the document is opened, the infectious macro extracts the *virus.cmd* executable which is already present in the computer. This executable is run. The infection has now spread.

Of course, this proof-of-concept scenario is operationally rather simple. One may object that only a few users will perform step 1 and that the level of threat of such a virus is strongly limited. It is far from being obvious. Many OpenOffice users are totally unaware of the malware threats. Let us recall that this proof-of-concept only aims at validating OpenOffice malware hazard, at the technical level. At the operational level (real-life attack), the use of social engineering or of some other technical tricks are likely to greatly empower attackers to spread virus like the *FinalTouch* virus.

Let us mention the essential characteristics of the *FinalTouch* virus.

- *Self-reproduction*. The virus is hidden in a frequently used macro (e.g. *DicOOo* or *FontOOo*) which moreover provides a lot of improvements compared to standard macros (multilingual environment, more available fonts...). This point greatly helps to make the virus spread. But many other possibilities exist.
- *High level of stealth*. The OpenDocument format enable to very efficiently hide executables of any kind in a file. In particular, the size of the infected file will always be far less important than the size of any Microsoft Office document. Moreover, the size of the virus itself is very limited. Only 20 command lines are

required to both extract infectious macros and run the executable. Those command lines may be written in such a way that they may appear harmless or innocuous even for a cautious or suspicious administrator who would decide to analyze every single command lines in existing macros. It is also possible to make detection even far more difficult and fool any analyst by considering alternate extension (e.g. *.XBA).
- The execution of the *virus.cmd* file may be delayed, for instance in order to make sure that many other files have been first infected.
- *High portability*. An infected file can contains executables in different format (e.g. for different operating systems).

4.3.5 Miscellanous sophisticated variants

Many other infection techniques have been identified and some of them have been successfully tested. It would be boring to describe them all. Consequently, we limit ourselves in summarising the main significant techniques. They are listed hereafter.

- The OfficeOpen.org configuration file can be modified in order to automatically activate malicious macros each time a document is opened.
- The infection may concern either the operating system level at the OpenOffice.org level only. This can be performed by placing malicious macros in trusted locations. A document level infection is possible as well by using association or chaining of macros and events. In this latter case, different documents may be involved according to the more or less complex scenarii we intend to play. A huge number of actions can thus be performed (non exhaustive list):
  - Save the viral code in either "OfficeOpen.org Macros" or "My Macros". Whenever a user load and open a document, the "OfficeOpen.org Macros" is automatically loaded and activated.
  - A malicious macro may replace a legitimate standard macro (e.g. C:\Program_Files\OpenOffice.org 2.0\share\basic\Euro). It corresponds to the case of usurping Microsoft Office viral macros.
  - We can use a OOo macro that mimics a Word VBA macro and embedded in a WORD document.
  - We can also use various events such as: additonal document opening, macro chaining, printing functions, keyboard shortcuts, use of hypertext links, cross-call between OpenOffice.org components (OOoWriter ↔ OOoCalc), cross-call of macros located in different libraries...
  - The use of OLE links and external programs/applications has been proven very powerful for designing infectious processes.

Lastly, in order to improve stealth capabilities in addition to the internal, specific OfficeOpen.org resources, we can use the function "Document as E-mail..." or "Document as MS-Doc Attachement". It has thus been possible to imagine an E-mail worm. In this latter case, the worm is able to collect E-mails on the hard disk by means of the system access files functions or commands.

5 Conclusion and future work

In this paper, we have presented an in-depth analysis of the OpenOffice.org security with respect to the computer viral threats. We have identified the different functionalities and components that could be used and perverted in order to concretely express this risk. Lastly, for validation purposes, several proof-of-concept malicious codes have been developed and successfully tested, in operational conditions. They allow us to claim while proving at the same time, that the OpenOffice suite may be affected by malicious codes as it commercial challenger is. The portability of the OpenOffice suite (Unices, MacOsX and Windows environments) greatly increases the level of risk since we have observed a striking similarity between the different operating system ports which is worth noticing. Any malicious code developed for OpenOffice will thus have a maximal impact. It is important to keep in mind that the viral risk attached to OpenOffice is independent of any implementation vulnerabilities. From the implementation and development point of view, OpenOffice is a remarkable software. We only focused on its intrinsic viral algorithmic capabilities.

This study has proved that up to now the viral threat with respect to the OpenOffice.org suite is more important than that with respect to Microsoft Office and thus for many reasons:

- the ZIP format for the archive represents a significant risk since it enables the entry of malicious codes. The control of such archives is likely to be difficult. The management of archive integrity is quite impossible to perform with open *Message Integrity Codes* (MIC). Any attacker can manipulate an archive and recompute any MIC in place. Consequently, efficient password protection or digital signature should be considered;
- Self-integrity of OpenOffice components should be used (see Sect 2.3.4);

– there exist more execution points that can potentially be used, diverted or hooked in OpenOffice than in Microsoft Office. The desire for ergonomics seems to have been prevalent over security during the development;
– the security has been insufficiently taken into account except for very marginal aspects. The secure management of macros is very difficult not to say complex to perform. It is beyond a simple user's awareness and capabilities. Future developments of the OpenOffice.org project should make the security a priority. Moreover this security must be ergonomically viable. In particular, the concept of trusted macros is very exagerated at the present time and should be completely removed. Digital signature is sufficient in itself. Any macro should always cause a warning security alert.

When considering the power of OpenOffice macros, the OpenDocument format interoperability as well as the ZIP format together, the conclusion is that the general security of OpenOffice is insufficient. This suite is up to now still vulnerable to many potential malware attacks. It is now obvious to us that the OpenDocument format is not secure with regards to the malware threat unless efficient integrity checking is used. It is possible with open tools like MICs? Digital signature seems to be the best answer.

In order to anticipate forthcoming OpenOffice malware, the first generic and efficient approach is to update antiviral engine in order they check the consistency of the structure files with respect to the OpenDocument format. This will greatly help to reduce the malware risk.

The first consequence of this study is that any security policy must take into account this new viral threat. Since its management is far more complex, it has to be managed directly at the administrator level. User must be constantly sensitized to the viral threat attached to the OpenOffice.org components. However, the main interest of OpenOffice lies in the fact that the system is completely open. Such an in-depth study would not have been possible with proprietary software. Moreover, the high reactivity and professionalism of OpenOffice team is the best hope to finally get a secure product very soon. We already have contact with OpenOffice developers to help them in correcting the weaknesses that have been identified. It is worth mentioning that such a reactivity and the opportunity to take part to the project is invaluable. Lastly, let us recall that document malware remains a constant risk for any software, both free or proprietary.

The analysis which has been presented in this paper has to be carried on in order to evaluate the threats with respect to some OpenOffice.org components that have not been extensively considered yet, due to the lack of time: UNO and the API project. Moreover the specific threat with respect to the *Python* programming language must be deeply analysed. Initial results have demonstrated the tremendous potential of this environment for OpenOffice malware design.

Acknowledgements We would like to express our gratitude to the lieutenant-colonel Filiol, our advisor, for his constant support and help during this study. We also would like to thank second lieutenant Rachida H'midouche for her valuable help in translating this article and in correcting some of the typos. Lastly many thanks to the anonymous referees who helped us very much in improving this paper.

References

1. Chambet, P., Detoisien, E., Filiol, E.: La fuite d'informations dans les documents propriétaires. Journal de la sécurité informatique MISC, numéro 7 (2003)
2. De Drézigué, D., Hansma, N.: Etude de faisabilité de macrovirus sous OpenOffice. Mémoire de stage mastère spécialisé "Réseaux et Télécommunications Militaires", Ecole Supérieure et d'Application des Transmissions (2006)
3. Filiol, E.: Le virus Concept. Journal de la sécurité informatique MISC, numéro 4 (2002)
4. Filiol, E.: Computer viruses: from theory to applications. IRIS International series, Springer, Berlin Heidelberg Newyork ISBN 2-287-23939-1 (2005)
5. Filiol, E.: Strong cryptography armoured computer viruses forbidding code analysis: the BRADLEY virus. In: Proceedings of the 14th EICAR Conference, pp. 201–217 (2005)
6. Filiol, E.: Techniques virales avancées. Collection IRIS, Springer, Berlin Heidelberg New York (in press) (2006)
7. FIPS 180-1: Secure Hash Standard, Federal Information Processing Standards Publication 180-1, U.S. Department of Commerce/N.I.S.T., National Technical Information Service, Springfield (1995)
8. ISO: ISO and IEC approve OpenDocument OASIS standard for data interoperability of office applications, http://www.iso.org/iso/en/commcentre/pressreleases/2006/Ref1004.html. See also http://ec.europa.eu/idabc/en/document/3439/5585 and http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=odf-adoption for more details (2006)
9. Oasis Standards: Open document format for office applications, OpenDocument v1.0, http://www.oasis-open.org/committees/download.php/12572/OpenDocument-v1.0-os.pdf (2005)
10. The OpenOffice.org Software Development Kit, http://www.openoffice.org/dev_docs/source/sdk
11. OpenOffice Suite Official Website, http://www.openoffice.org.
12. Major OpenOffice.org Deployments, http://wiki.services.openoffice.org/wiki/Major_OpenOffice.org_Deployments
13. Marcelly, B., Godard, L.: Programmation OpenOffice.org: Macros OOoBasic et API, Eyrolles, ISBN 2-212-11439-7 (2004)

14. Rautiainen, S.: OpenOffice security. In: Virus bulletin conference, september 2003 (2003)
15. Reynaud-Plantey, D.: New viral threats of Java viruses. J. Comput. Virol. 1(1-2), pp. 32–43 (2005)
16. Kasliski, B.: RFC 2898-PKCS#5: Password-Based Cryptography Specification Version 2.0, Network Working Group, http://www.faqs.org/rfcs/rfc2898.html (2000)
17. Schneier, B: Description of a new variable-length key, 64-bit Block Cipher (Blowfish). In: Anderson, R. (ed.) Fast Software Encryption, Cambridge Security Workshop, LNCS 809, pp. 191–204, Springer, Berlin Heidelberg New York (1994)
18. http://smallbiz.symantec.com/press/1998/n980819.html
19. www.w3.org/XML/Schema