OPENOFFICE SECURITY

Sami Rautiainen
F-Secure Corporation, PL 24, FIN-00181 Helsinki, Finland

Tel +358 9 2520 0700 • Fax +358 9 2520 5014 • Email Sami.Rautiainen@F-Secure.com

ABSTRACT

OpenOffice, the open office suite is available for Linux, Windows and Solaris, with the other platforms like MacOS X under development. It has reached the production version 1.0, being already the most popular office suite for Linux.

This paper discusses the security model of OpenOffice – the environment and restrictions that are provided for executable content such as macros and embedded objects. The paper also examines the features of the native macro language and the XML file format used by OpenOffice.

The OpenOffice feature set is similar to the feature set of Microsoft Office. It has word processing, spreadsheet and presentation applications all tied together within a common look and feel.

OpenOffice has a built-in macro language and a support for both JavaScript and Java applets. Looking at these features, the question arises: How secure is OpenOffice? Have OpenOffice developers taken into account the pitfalls shown by the history of the Microsoft Office or is OpenOffice the next victim of the abuse of macro viruses?

1 INTRODUCTION

OpenOffice is a feature-rich Office application suite available for no cost with source code. At the time of writing of the paper, it is available for *Windows*, *Linux*, *Sun Solaris* and as a beta version for MacOS X. This paper concentrates on version 1.0 of the product with some notes based on the changes in the current development version 1.1 beta 2.

The *OpenOffice* suite is based on the *Sun Microsystems StarOffice 5.2*, released in 2000. After the release of the *StarOffice*, *Sun* decided to transform proprietary *StarOffice* into open source project and *OpenOffice* was born. *StarOffice* has been based on the *OpenOffice* code base starting from version 6.0 [1].

The *OpenOffice* suite consists of three major applications: *Write* for the word processing, *Calc* for the spreadsheets, and *Impress* for the presentations and slide shows. In addition, there are two supporting applications: *Draw* is the vector graphics application and *Math* is the equation editor. These applications roughly resemble *Microsoft*'s *Office* suite with *Word*, *Excel* and *PowerPoint* respectively. While the database component, *Access*, has no substitute within *OpenOffice* yet, the applications of the suite can access several open source and commercial database engines either directly or over Open Database Connectivity (ODBC).

2 INSTALLATION

One important security aspect of any product, especially when the product is targeted to end user market, is the default installation. The default settings are used by most of the users, and it is extremely important to have secure default configuration in place – the users are not willing to change anything later.

Each version of *OpenOffice* has a different installation location by default, as the installation directory name contains the version number of the suite. For example, by default, the shared components of version 1.0.3 are installed on

- /usr/local/OpenOffice.org1.0.3 on *Linux*
- C:\Program Files\OpenOffice1.0.3 on *Windows*.

On *Windows*, the installation program takes care of the integration to the user's environment directly. The installation will assign the *OpenOffice* file extensions to appropriate applications via the registry and install the quick start applet into the system tray. The filename extensions that are assigned for *OpenOffice* suite by the installation program are:

Extension	Application
sxw	Writer document
stw	Writer template
sxc	Calc workbook
stc	Calc template
sxi	Impress presentation
sti	Impress template
sxd	Draw picture
std	Draw template
sxm	Math equation
sxg	Master document

On *Linux*, the installation program must be executed for each user separately when the integration takes place. The installer updates the user's .mailcap file, so that the mime types assigned for the *OpenOffice* applications will be assigned to the appropriate applications.

This way, any *Linux* mail client that has the mailcap(4)support will be able to open *OpenOffce* documents directly from email attachments. The mailcap system was first introduced in the *Metamail* package [2], and is nowadays the most common way to handle different

file types on *Linux*. Furthermore, the installer can integrate *OpenOffice* documents into the desktop environment so that files accessed via desktop or the default file browser of the desktop environment can be opened directly.

In both platforms, after installation, any *OpenOffice* document can be opened from the user interface. This includes those documents that have executable code in them. In *OpenOffice*'s case even the macros can be executed, if the user confirms the warning dialog.

3 THE SECURITY MODEL

The security model used by *OpenOffice* for its native macros is quite simple. The decision whenever a macro can be executed or not, is made upon a absolute path where the file is located. For example, with the default settings the current version has this set to the user subdirectory in the *OpenOffice* installation directory. This means that every document copied into that specific, trusted directory is allowed to execute macros without warnings. If a document that contains macros is not located in the trusted directory, it triggers a warning dialog. From this dialog the user can either deny or allow the execution – in the same way as with *Microsoft Office*. The default action is to deny execution.

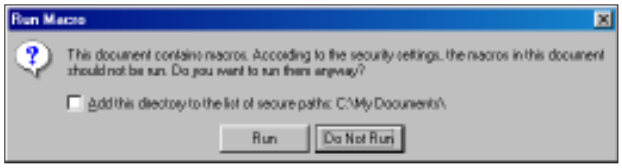

Figure 1: The macro confirmation dialog.

Additionally, the user is allowed to add the current working directory to the trusted path list. The ability to add the current working directory to the trusted directory list can be harmful, as it is easy to add for example a temporary directory to the trusted list by mistake.

Versions 1.0 and 1.0.1 of *OpenOffice* contained a security hole in the default setting for the trusted directory: the default trusted path was set into the user's home directory in the *Linux* version and to the whole C: drive in the *Windows* versions, *allowing* that practically all documents had the

Figure 2:* OpenOffice *security settings dialog.

macros enabled without warnings by default. Fortunately, this problem was corrected in version 1.0.2.

The OpenOffice default security settings are:

Setting	**1.0, 1.0.1**	**1.0.2, 1.0.3, 1.1 beta 2**
Run macro	Path settings	Path settings
Path list - *Linux*	~	User directory at the installation directory
Path list - *Windows*	C:\	User directory at the installation directory
Hyperlinks	With security check	With security check
Java	Enabled	Enabled
Java security checks	Enabled	Enabled
Java net access	Unrestricted	Unrestricted

The idea of trusted directories is easy to understand, and that probably has been the goal of the developers of the *OpenOffice* – keeping things simple. However, the security settings are perhaps oversimplified, providing only an all-or-nothing approach. It is unlikely that a user, who needs macros in the first place, needs them with the all applications of the suite. More typically, there is a need to use macros only in some applications – for example *Calc*. Therefore, it would be better if these settings could be customized on each application separately. That would allow more flexible configurations where, for example, macros in spreadsheets are enabled from a specific directory, but not for the rest of the suite.

3.1 Embedded Java, JavaScript & hyperlinks

OpenOffice also supports embedded JavaScript code, however, the JavaScript code is not executed from the document; the main idea is that one can create HTML pages that contain JavaScript with *OpenOffice*.

Additionally, *OpenOffice* contains support for Java. A document can contain a Java applet, that is by default executed whenever a document is opened – the Java support has completely different security settings than native macros. A Java virtual machine, by default, has similar restrictions in place that a typical browser has – there is no access to the local file system but only to the originating remote server.

A document can also contain hyperlinks to remote locations or local file system. There is no security check for links that point to the remote locations, such as a website. The links are passed to the default browser as they are. For local links the user has to confirm that the link will be followed before it is passed to the default file manager.

3.2 Embedded objects

The embedded objects, such as OLE objects, are executed or passed to appropriate application as they are. There are no security checks or warnings at all.

This is true for the embedded *OpenOffice* documents that have automatically executing macros as well – they are opened without any warnings whatsoever, regardless of the content of the embedded document. Fortunately, the macros are not executed from embedded *OpenOffice* documents.

As a matter of fact, with versions 1.0.2 and 1.0.3, if the user attempts to edit macros from embedded *OpenOffice* documents, the application will just crash. The problem causing the crash has been corrected within version 1.1 beta 2. Fortunately this version does not still execute macros from embedded documents.

The current implementation of *OpenOffice* allows that a document containing executable code can be executed by the user regardless of the security settings exposing the system open for attack. At the bare minimum, a warning should be shown to the user before an embedded object is activated and is passed to the host system or another application.

4 THE FILE FORMAT

The *OpenOffice* file format is a collection of Extensible Markup Language (XML) files with the associating binary data, stored into a single compressed archive in a zip format. In the documentation this file is referred as a package file [3].

The XML is a flexible text format. The basic semantics of the format are standardized by World Wide Web Consortium (W3C) and each application is required to define its own tags or elements used in the XML formatted document. This definition is referred to as a Document Type Definition (DTD) [4].

OpenOffice also supports several other file formats, like the legacy *StarOffice* OLE format, Rich Text Format (RTF) and *Microsoft Office* file formats. However, this paper will concentrate on the default XML file format.

A basic document package contains five different files, outlined below:

File name	Purpose
content.xml	Document content.
styles.xml	Styles (optional).
meta.xml	Document metadata.
settings.xml	Basic document settings.
META-INF/manifest.xml	*Manifest file:* contains information about the files in the package.

Each package always contains a manifest file. The purpose of this file is to list the files in the archive, their media types and, if the file is encrypted, the algorithm and other information needed to decrypt the file.

The basic package structure is the same within all *OpenOffice* applications.

4.1 Macros in the package file

Native macros are stored in the package file to their own directory – Basic. The directory contains a separate file for each macro module. For example, a document that has a single macro module, will have two files in addition to the five mentioned above:

File name	Purpose
Basic/Standard/Module1.xml	The macro code in plain text
Basic/Standard/script-lb.xml	The list of libraries and external objects used by the macro

The media type of the macro files in the manifest files is text/xml, which is used for other XML formatted files as well. However, the *OpenOffice* enforces the directory structure – macro programs are recognized and used by the application only if they are under the Basic subdirectory.

If the name of the subdirectory is changed, the current version of *OpenOffice* will ignore the macros, even if the information in the manifest file is changed accordingly. Therefore, the presence of macros can be reliably determined by checking the presence of the Basic subdirectory in the package file.

The macros themselves are stored in the XML files that contain a <script:module> element. This element contains the macro code in plain text format. The following example has been reformatted for clarity.

```
<?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=”Module1" script:language=”StarBasic”>
REM ***** BASIC *****
Sub Main
        msgbox &amp;quot;test macro&amp;quot;
End Sub</script:module>
```

By default no macros are executed by the application. Unlike *Microsoft*'s VBA, there are no magic subroutine names to be used – instead, *OpenOffice* requires that an event such as AutoOpen has to be defined in content.xml if the macro needs to be executed automatically. This can be done from the user interface, by assigning an event for the macro.

4.2 Embedded scripts

Embedded scripts are stored in content.xml in a plain text format, within <text:script> element. The attribute script:language specifies the scripting language:

```
<text:script script:language="JavaScript">
        alert(&quot;test script&quot;);
</text:script>
```

As mentioned above, the embedded scripts are not executed by *OpenOffice*. The purpose of this element is to store scripts when *OpenOffice* is used to edit files that will be exported to HTML.

4.3 Embedded objects

From the file format point of view, there are two different types of embedded objects: XML objects and binary objects. Native *OpenOffice* file objects, such as charts or spreadsheets are stored in XML format and other binary objects, such as OLE objects or graphics, are stored in the binary format.

Each embedded object is referenced in the content.xml by using a draw XML tag. For example, an embedded OLE object – in this case an *Excel* workbook – is referenced as follows:

```
<draw:object-ole draw:style-name="fr1"
 draw:name="Object1" text:anchor-type="paragraph"
 svg:width="5.001cm"
 svg:height="5.001cm" draw:z-index="0" xlink:href="#./
ObjBFFFE441"
  x link:type="simple" xlink:show="embed"
xlink:actuate="onLoad"/>
```

The two types of the objects are set with their own elements: <draw:object> is an XML object and <draw:object-ole> is an OLE object. Both elements have attributes that control the appearance of the object. Additionally, there is an attribute that specifies the physical location of the embedded object:

`xlink:href` Physical object name. With a binary object this attribute specifies the file name in the archive that contains the embedded OLE object. With XML objects, this is a subdirectory that contains all XML files related to the embedded object. The hash (#) in the beginning of the file name is a short for the current package.

The `xlink:href` attribute can also specify a linked external file. In that case the attribute contains the full URL.

In the above example, the object is stored in the root of the zip archive as ObjBFFFE411. Since this file is not altered by *OpenOffice* in any way, an anti-virus scanner that has archive scanning enabled will detect whether the embedded object is infected or not. The scanner actually does not have to care if the object is listed in the content.xml at all.

However, unfortunately the *OpenOffice* XML file format specification also allows that the embedded object is directly embedded into content.xml. The <draw:object-ole> can contain another element, <office:binary>, that holds the embedded binary object in base64 encoded format. Also, the XML object can be embedded directly into the <draw:object> element. In both cases the xlink:href attribute is not present. The direct embedding is not used by the current versions of the applications – the packages written by the suite have objects always separated to their own files.

This means that extracting and parsing the content.xml file from the *OpenOffice* package file is required to determine whether the document has any executable content. In this sense the *OpenOffice* file format has similar problems as the upcoming *Microsoft Office XP* with its XML file format [5]. In *OpenOffice*'s case the potential performance hit is even higher, due first to the compression used to store the package and second to the fact that there is no easy way to determine whether the content.xml contains embedded objects other than processing the entire file.

5 THE STARBASIC MACRO LANGUAGE

The macro language used by *OpenOffice* is called StarBasic. The language itself, like its name originates from *Sun Microsystems StarOffice*.

The StarBasic language has a rich set of functionality. Beside normal Basic operations, it can access all the functionality that *OpenOffice* provides via the OpenOffice Application Programming Interface (API). This will be discussed in greater detail below.

The macro language has all the common features of modern Basic – it closely resembles *Microsoft*'s Visual Basic for Applications (VBA) . Like VBA, StarBasic supports object-oriented programming.

5.1 StarBasic built-in functions

Beside a normal instruction set for the program control, StarBasic has a number of built-in run-time functions providing access for

- screen and file I/O
- and time handling
- error handling
- mathematical operations
- logical operations

The file I/O functions allow a macro to read and write any binary or text file to which the user has access. Additionally,

macro language provides access to the host system – the macro can fetch the environment variables via the Environ() function and execute any host executable via the Shell() function without any additional warning. The only requirement is that the user has privilege to run the command. This is enforced by the operating system and not *OpenOffice*. Therefore these functions alone are already obviously dangerous. The Shell() function combined with file I/O, a macro can first write a shell or batch script, or even a binary executable, and run it – allowing macro to perform any action in the context of the user.

5.2 Compatibility with Microsoft Office

OpenOffice has import and export file filters for *Microsoft Office* applications, providing compability between *OpenOffice* users and *Microsoft Office* users allowing exchange of the documents, for example, within an organization that have both suites in use.

One feature of the file filters of *OpenOffice* is that they are able to read the macros from documents created with *Microsoft Office*. These macros are automatically converted to StarBasic, but the code itself is disabled with Rem instructions. If the file is converted back to the *Microsoft* format, the macro is saved again to VBA's native format. Therefore, a *Microsoft Office* macro virus cannot be executed in *OpenOffice*, however, a file that is edited in the *OpenOffice* and then converted back to the *Microsoft* format, can carry a macro virus that will be activated again in *Microsoft Office*.

This behaviour is similar with *StarOffice 5.2* [6]. However, the conversion back to *Microsoft Office* format is behaving incorrectly in the current version of the *OpenOffice*. The macros carried via, for example, *OpenOffice Write*, will be effectively disabled. These macros will no execute at all unless they are edited or recompiled within the VBA editor of *Microsoft Office*. The reason for the behaviour is unknown and requires further research.

6 OPENOFFICE APPLICATION PROGRAMMING INTERFACE

The OpenOffice API is divided into several parts, or collections, categorized by the purpose. For example, the text editing functionality is in the sun.com.star.text collection. This collection contains several services, like sun.com.star.text.TextDocument, that provide means to a macro or external application to edit the document body.

At the time of writing of this paper there are more than 40 different collections, providing interfaces for different components of *OpenOffice*. This allows the development of feature-rich applications based on the *OpenOffice* suite. The API can be accessed from a StarBasic macro allowing development of effective cross-application and cross-platform programs within *OpenOffice*. For example, a macro that runs on the *OpenOffice Writer* can perform calculations on the *Math*, and return results to the document in the *Writer*. The possibilities of the API is not limited to the application of the suite, as the API contains interface to the host system as well, including basic email services and host system file I/O [7].

Beside the native macro language, the API is accessible via the Universal Network Objects (UNO) interface. The UNO implementation of the *OpenOffice* provides both local and remote access to the running instance, integration with several different programming languages such as C, C++, Java and Python. UNO also provides a bridge to OLE automation on *Windows*, allowing a control of the *OpenOffice* applications from Windows Scripting Host or Visual Basic for Applications (VBA). The remote access feature of the UNO implementation is disabled by default. It needs to be enabled explicitly with a command line parameter when any of the *OpenOffice* applications is started.

6.1 OpenOffice API from StarBasic

In the following, the API is discussed from the view of StarBasic macros. However, the same general rules apply to all interfaces provided by UNO.

As previously said, the API provides a macro to access to almost any aspect of the *OpenOffice* suite. Besides control of the documents, such as text editing or calculations in spreadsheets, there are some collections that have functionality whose availability in an API of an *Office* application suite is questionable, at least in the default configuration.

One of them is the com.sun.star.system collection, providing the host operating system integration services [8]. The services in this collection can be categorized in three different groups: the proxy settings, system shell execute and email interface.

The first group is proxy settings. These settings are used when a remote hyperlink is followed from a document. This interface allows a macro to determine if a proxy server is used to connect remote machines and if a proxy is enabled, which protocol is used for, and what is the network address of the proxy server.

The second group is actually a single service, SystemShellExecute, that effectively provides the same functionality as the StarBasic built-in function Shell() – it allows execution of an arbitrary system command, so this service is not important in the context of macro programs. However, due to the remote access ability of UNO, a remote program can execute local commands with user's privileges.

The third group, email interface, consists of two services:

SimpleSystemMail and SimpleCommandMail. These services allow a macro to send email, using the system or the user's email client. By default, the configuration is done on *Linux* so that the system mail command is used to perform the task of the sending email and on *Windows* the Mail Application Programming Interface (MAPI) service is used instead. These services, together with *OpenOffice*'s ability to use the system address book imported from the default email application, make *OpenOffice* potentially vulnerable to mass mailers.

There are no means to restrict the access to different parts of the *OpenOffice* API for macros or other programs. Macro programs that have no other need than to edit a spreadsheet have exactly the same features available from the API as a program that sends a letter to predefined set of recipients. There is a clear need for the configuration to be made more flexible, on an application-by-application basis with the possibility to fine-tune which parts of the API are enabled.

6.2 Access to macro modules from OpenOffice API

The API also allows the macro to access the current document – the document where the macro is loaded and executed. In many cases there is a valid need to modify the document itself, but the bad thing is that a macro can access, beside the document text, also the macro source code. This includes the currently running macro itself. As a matter of fact, the macro is not limited to its own document, it can also access other documents that are loaded as well as the global macro modules.

Global macros are shared between users in a single machine. In the default installation of *OpenOffice for Linux*, these macros are stored in files that cannot be written by the normal user. On *Windows*, the default installation of *OpenOffice* fails to set any restrictions on these files, allowing any user to add or modify macros that are executed in the context of another users of the same system.

The *OpenOffice* macro architecture resembles the global and local macro space in the *Microsoft Office*. It is technically possible to write a macro that installs a macro program to the global template space, provided that the user has the applicable privileges, as well as it is possible to write a macro that inserts a program to the local document that is being edited – making *OpenOffice* vulnerable for macro viruses similar found mostly from *Microsoft Office*.

7 CONCLUSIONS

The macro language and the API of *OpenOffice* is very powerful. Unfortunately this power can be abused for malicious purposes. The security settings in the default installation of *OpenOffice* much resemble older versions of the *Microsoft Office*, providing the user the possibility to run an unknown macro directly from a document after a simple warning dialog. As learned from the past, this is a measure that is not going to help much. Instead, the default settings should be more restricted: macros from non-trusted directories should be disabled completely by default, making it more difficult for the user to execute a macro in a document that originates from an unknown source.

The *OpenOffice* API is currently in a state where all features are available from all the interfaces, regardless of whether the interface is local or remote. If remote access is enabled, a user does not have control over what can or cannot be done – in the worst case the user does not even know that the remote service has been started. Also, the majority of the functionality is something that most of the end users will not ever need, for example the unrestricted access to the host system, macro modules, address books and email components. The user should have more control over when and how different components can be accessed. Better yet, the access in the default configuration to potentially dangerous interfaces must be disabled.

The majority of the file format of the *OpenOffice* document packages is well documented by the development team. Even better, *OpenOffice* enforces certain rules in the file format, such as location of the macro code in the archive, making it easier to determine whenever there are objects in the file that needs to be scanned or removed. The biggest problem within the file format seems to be the content.xml file that needs always to be decompressed and parsed in order to find embedded scripts or objects.

REFERENCES

[1] About Us: OpenOffice.org, OpenOffice.org, USA, 2003. http://www.openoffice.org/about.html.

[2] Metamail, Bell Communications Research, Inc., USA, 1991, http://www.bell-labs.com/project/wwexptools/packages.html.

[3] OpenOffice.org XML File Format Technical Reference Manual, Version 2, OpenOffice.org, USA, December 2002.

[4] Extensible Markup Language (XML) 1.0, Second Edition, World Wide Web Consortium (W3C), USA, October 2000, http://www.w3.org/TR/REC-xml.

[5] Szappanos, Gabor, 'XML Heaven', *Virus Bulletin*, February 2003.

[6] Kaminski, Jakub, 'Linux Malware – Has the Next Battlefield Been Decided?' *Proc. Int. Virus Bull. Conf., 2000*.

[7] OpenOffice.org Developer's Guide 1.0.2, OpenOffice.org, 2003, http://api.openoffice.org/DevelopersGuide/DevelopersGuide.html.

[8] OpenOffice.org API reference, OpenOffice.org, USA, 2003, http://api.openoffice.org/common/ref/com/sun/star/module-ix.html.