Self-Protection Strategies

Tunneling, Armored, and Retro Viruses

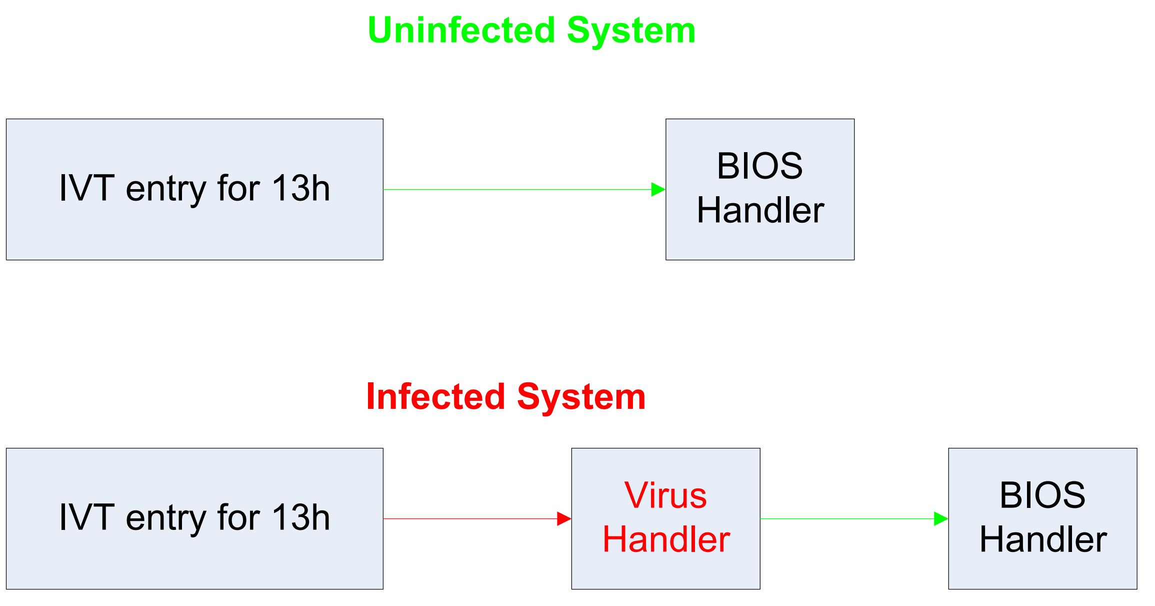
CS4400/7440

Anti-anti-virus Techniques

- Virus writers have devised numerous methods of resisting anti-virus software and making life difficult for anti-virus researchers
- We will examine four categories of virus self-protection in coming weeks:
  - tunneling,
  - armor,
  - retroviruses, and
  - encrypted viruses of several types
- Reading Assignment: Chapter 6 of Szor.

Tunneling Viruses

- Recall the DOS IVT (interrupt vector table) and the technique of interrupt hooking:



Background: Chaining Interrupt Handlers

- Interrupts contain address pointing to interrupt vector
  - Interrupt vector contains addresses of interrupt handlers.
- If more devices than elements in interrupt vector, then chain:
  - List of handlers for given address traversed to determine the appropriate one.

vector number	description
0	divide error
1	debug exception
2	null interrupt
3	breakpoint
4	INTO-detected overflow
5	bound range exception
6	invalid opcode
7	device not available
8	double fault
9	coprocessor segment overrun (reserved)
10	invalid task state segment
11	segment not present
12	stack fault
13	general protection
14	page fault
15	(Intel reserved, do not use)
16	floating-point error
17	alignment check
18	machine check
19–31	(Intel reserved, do not use)
32–255	maskable interrupts

Pentium Processor Event-Vector Table

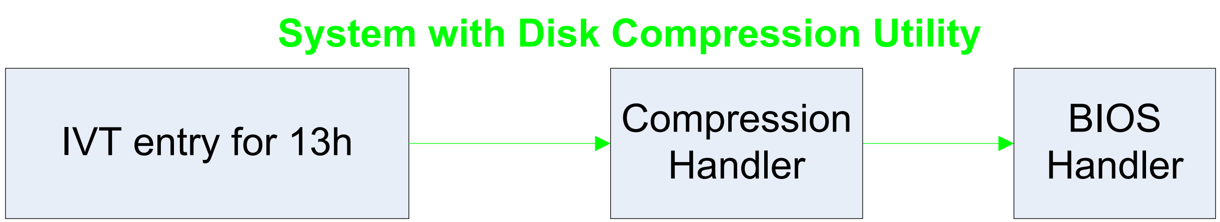
Hooking an Interrupt

1. Get location/length of IDT using Intel **`sidt`** instr.
   - SIDT (Store Interrupt Descriptor Table) stores contents IDTR (Interrupt Descriptor Table Register) register,
     - which is a selector that points into the Interrupt Descriptor Table.
2. Each descriptor is 8 bytes: Index into the Table by 8n bytes to change interrupt n
3. This descriptor contains the address of the Ring0 code to run for interrupt n

- This address is changed to point to hooking code
  - Additional work to chain

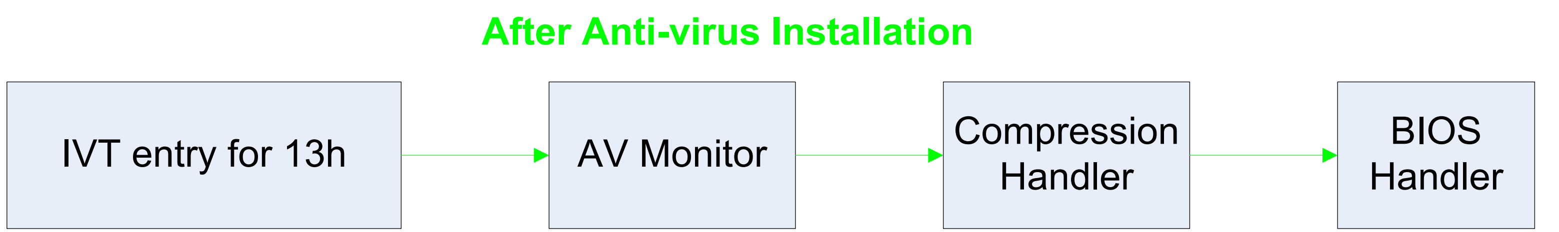
Interrupt Hooking

- Interrupt hooking IS a legitimate technique,
  - e.g. a disk compression utility might need to intercept disk accesses to compress and decompress on the fly:



Anti-virus Interrupt Monitors

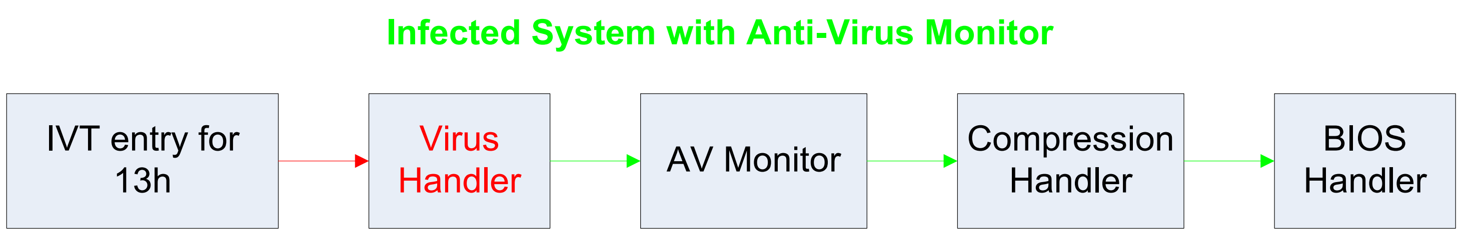
- When an anti-virus program executes at boot-up time, it installs a monitor that lengthens the call chain even more:



- The AV monitor checks to see if it is first on the call chain.
  - If so, calls the saved address for the next item on the chain (in this case, the compression handler).

Detecting the Interrupt Hooking Virus

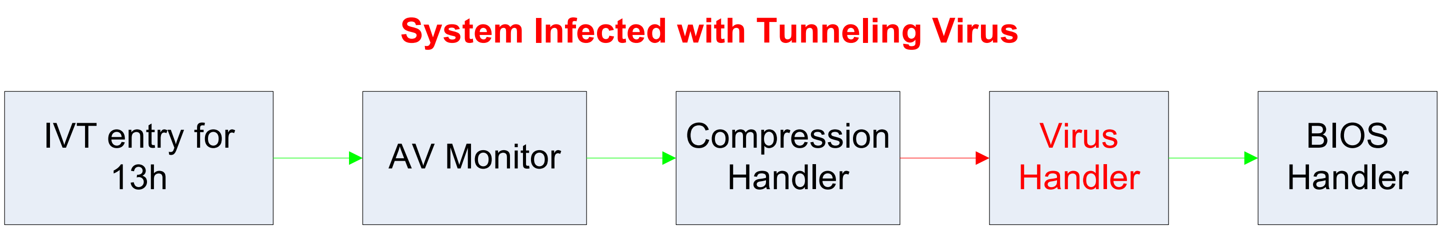
- However, if a virus has hooked the interrupt, then the anti-virus monitor code detects that *it is not being called directly* from the IVT:



  - The AV monitor now begins virus disinfection.

Tunneling Viruses

- A tunneling virus defeats the anti-virus monitor by following the interrupt call chain until it finds the end, installing itself there instead of at the beginning:



- The AV monitor now finds itself pointed to directly from the IVT and finds nothing to disinfect.

Tunneling Methods

- The process of following the interrupt call chain is called *tunneling*, because the virus is trying to locate itself in the system in a place that is beneath the vision of the anti-virus software
- How can a virus follow the call chain?
  - Emulation (sophisticated and costly)
  - Stepping through instructions in debug mode
  - In DOS, scanning all of memory to find the code that calls the BIOS handler, which must be the end of the chain

Defeating Tunneling Viruses

- The AV monitor
  - can scan in both directions and record the call chain for later checking
  - scan for virus code patterns throughout all the handlers in the call chain,
    - in case the virus had already tunneled down the chain before the AV software was installed
  - removes the virus handler when it is detected

Interrupt Wars

- An interrupt hooking virus usually has a memory-resident file infector component in addition to the interrupt handler; the handler calls the infector
- The memory-resident component can detect that the handler has been removed, and can re-install it at the end of the call chain
- The AV monitor will detect the new virus handler and remove it again; this *interrupt war,* carried on while interrupts are being processed, can make a system unstable
- Solution: find and remove the memory-resident code immediately before removing the handler

Armored Viruses

- An armored virus makes it difficult for anti-virus professionals to detect and analyze its functions
- Anti-virus professionals use a variety of detection and analysis tools:
  - Disassemblers
  - Debuggers
  - Emulators
  - Heuristic analyzers
  - Goat files
- Armored viruses try to make each of these tools ineffective or more difficult to use

Armored Viruses

- Armored virus techniques fall naturally into five categories, corresponding to the five tools they are designed to combat:
  - Anti-disassembly
  - Anti-debugging
  - Anti-emulation
  - Anti-heuristics
  - Anti-goat

Anti-Disassembly

- The broadest category of techniques that make disassembly difficult are the virus code encryption techniques, which we will study separately for several weeks starting next week. Other techniques:
  - Encrypted data
  - Code obfuscation
  - Using checksums
  - Compressed code
- We will examine each of these briefly

Encrypted Data

- The virus encrypts its data and decrypts it as it is used
- The encryption and decryption code is clearly visible, so it is straightforward to figure out
- BUT, when viewing the code in a disassembler, the data is garbled
- Labor-intensive: The anti-virus software engineer is slowed down by the need to emulate code, write a decryption utility program and paste data into it, etc.

Encrypted Data Example

- The Fix2001 worm attacked Windows 95 systems in 2001
- The worm sent stolen accounts and passwords by email back to a free email address (e.g. hotmail.com) obtained with a false identity
- The worm author did not want the email address to be readable to a disassembler
- The address was in a constant data section that was encrypted
- Stepping through a debugger to watch the data be decrypted slows down the analysis

Code Obfuscation

- We saw a DOS example two weeks ago that used a jump into the middle of a previous instruction
- Some obfuscation merely injects no-ops, do-nothings (e.g. `add eax,0`)
  - Regular expression matching can filter these out
  - Analysis is not slowed much by these instructions
- It is slower to analyze code with roundabout computations, computed jump addresses rather than direct jumps, etc.

Obfuscated Computation

- Example from Szor text, p. 223:
- Straightforward code to write 256 bytes into a file:

```
mov   cx, 100h    ; 100h = 256 bytes to write
mov   ah, 40h     ; 40h = DOS function number
int   21h         ; Invoke DOS handler
```

- Convoluted code to do the same thing:

```
mov   cx,003Fh    ; cx = 003fh
inc   cx          ; cx = 0040h
xchg  ch, cl      ; swap ch, cl (cx = 4000h)
xchg  ax, cx      ; swap ax, cx (ax = 4000h)
mov   cx, 0100h   ; cx = 100h
int   21h         ; Invoke DOS handler
```

Anti-Disassembly Checksums

- Straightforward code to match an imported function prototype, from the exported functions list in DLL,
  - to decide which system functions to infect,
  - might loop through the DLL function names list and
  - compare each function name to a constant string, e.g. (in C pseudocode),

```
for (each prototype in DLL export table)
 if (0 == strcmp(name,"GetFileHandle(int)"))
     infect(current export table address);
endfor
```

- Easy to read in the disassembled code;
  - good disassembler can even search and find the string "GetFileHandle" if the anti-virus researcher already suspects that is the function being infected

Checksums cont'd.

- Instead, the virus could compute a checksum over the ASCII bytes of the two strings, store one as a constant, and compare the checksums for equality:

```
int ConstantName = 0x89f7e5b2; /* Computed by virus writer */
for (each prototype in DLL export table)
    int foo = checksum(name);
    if (foo == ConstantName)
        infect(current export table address);
endfor
```

- This code no longer reveals the API name to a reader
- Labor Intensive: Anti-virus researcher must now step through the checksum computation to figure out what is going on
  - i.e., impedes the analysis
- Similar idea to encrypting data

Anti-Disassembly Compression

- A virus can be stored using a compression algorithm, and decompressed during execution by a decompression code at the beginning of the virus
- As with encrypted data, the compression algorithm is exposed, but examination of disassembled code is greatly slowed down
- Anti-virus researcher might need to emulate the code, or step through it in a debugger

Armored Viruses

- Armored virus techniques fall naturally into five categories, corresponding to the five tools they are designed to combat:
  - Anti-disassembly
  - Anti-debugging
  - Anti-emulation
  - Anti-heuristics
  - Anti-goat

Anti-Debugging

- We have seen that anti-disassembly techniques might drive an anti-virus researcher to step through virus code in a debugger
- The next step in the escalating war between the virus and anti-virus communities is the **development of virus code that resists being executed in a debugger**

Anti-Debugging Techniques

- Interrupts 1 and 3 are used often in x86 debugging
  - `INT 1` places the CPU in single-step mode
  - `INT 3` is inserted into the code by the debugger to set a breakpoint
- First anti-debugging technique: Hook these two interrupts
  - Anti-virus code must be used to unhook the interrupts before debugging can proceed
- Next anti-debugging technique: use a checksum to defeat `INT 3` breakpoints

Anti-Debugging Techniques

- If the virus code computes a checksum over a critical range of its code, stores that checksum in a constant, and then recomputes the checksum as it runs, it can detect the change when the `INT 3` instruction is injected into that code range, and just abort
  - The virus now runs successfully without a debugger, but aborts when breakpoints are inserted
- Note that a code emulator, as opposed to a debugger, does not insert breakpoints and is not defeated by this technique

Anti-Debugging Techniques

- Next technique: Detecting changes in the state of the stack during single-step mode
- A single-step debugger places a state record on the stack, and updates it after each step, to record the current `IP` (instruction pointer) value and the contents of the `FLAGS` register
- The virus code can examine the stack and see changing values where they would not change during normal (non-debugger) execution, and abort

Detecting Stack Changes

- In single-step debug mode, the debugger, after each instruction, saves state change information in a record that is placed just beyond the top of the stack.
- This location will be trampled by any user program instructions that change the stack, but the old info is not needed after an instruction is executed.

Detecting Stack Changes

- Without single-step debug state changes, a location on the stack will remain unchanged until an instruction changes it, but will be changed by the debugger after every instruction during single-step debug:

```
mov bp,sp        ; bp gets current stack pointer
push ax
pop ax           ; old pushed value still at [bp-2]
                 ;  which is beyond current stack
cmp word ptr [bp-2],ax ; equal if no debugger
jne DEBUG        ; debugger detected! Go abort!
```

Anti-Debugging Techniques

- If the virus encrypts part of its code, it can use the stack pointer in the decryption routine
  - When the debugger uses the stack to save state, it will change the stack pointer and cause the decryption to fail
  - Virus now executes only without a debugger
- Disabling the keyboard interrupt during virus execution prevents the AV researcher from using a debugger
- Other techniques are listed in Szor, 6.2.7

Detecting Debuggers

- On Win32 operating systems, an API is available: IsDebuggerPresent()
  - Virus can just abort if TRUE
- Many debuggers set registry keys when they are active
- Viruses can scan memory for debugger code

Armored Viruses

- Armored virus techniques fall naturally into five categories, corresponding to the five tools they are designed to combat:
  - Anti-disassembly
  - Anti-debugging
  - Anti-emulation
  - Anti-heuristics
  - Anti-goat

Anti-Emulation Techniques

- em·u·la·tor ***noun***
  - **1 :** one that emulates; imitator
  - **2:** hardware or software that permits programs written for one computer to be run on another computer
- Emulators approximate the behavior of their target
  - Why is this?
  - Emulation runs much more slowly than actual machine
- Anti-Emulation Armoring:
  - Floating point code
  - Take advantage of the approximation – be exceptional
  - Time/logic bombs
  - Dynamic code length

Anti-Emulation: Floating Point Code

- If armored viruses have driven AV researchers away from disassemblers and debuggers,
  - next step for virus writers is to impair use of code emulators
- Early emulators only kept track of the integer CPU registers and memory,
  - viruses were not floating-point code
- Viruses then began to deliberately use the floating point coprocessor registers and instructions
  - More recently, MMX and SSE vector graphics instructions are appearing in viruses
  - Modern emulators responded by emulating all registers and instructions
- Also, can use undocumented x86 instructions to similar effect!

Anti-Emulation: Exceptions

- The emulation environment is not always able to predict whether the next instruction will cause an exception (why?)
  - Viruses have been written to exploit this problem by putting part of their code in exception handlers, and then causing very subtle exceptions to occur
  - If part of the virus decompressor or decryptor code is in an exception handler, the emulator will fail to decrypt or decompress and emulation will try to execute garbage bytes

Anti-Emulation: Time and Logic Bombs

- Arrange for virus to execute only…
  - at certain times of day (time bomb),
  - or only under random conditions (logic bomb) that might be controlled by random number generation
- If the time/condition not met, the virus just transfers control to the infected host program and does no damage
- When executed in an emulator, the virus will probably be dormant and cannot be analyzed by the emulator
  - AV analysis forced to use a debugger

Anti-Emulation: Dynamic Code Length

- Code emulation is very slow (interpretation of an entire simulated machine environment)
  - Not a problem for most viruses, as most they start up pretty quickly and are written in tight code
- Anti-emulation Armoring: However, use huge loops of do-nothing instructions with a few real virus instructions thrown in & a high loop count
  - Emulation becomes too lengthy
  - Can also be done by encoding brute-force decryptors into the virus, which run billions of instructions before successfully decrypting it

Armored Viruses

- Armored virus techniques fall naturally into five categories, corresponding to the five tools they are designed to combat:
  - Anti-disassembly
  - Anti-debugging
  - Anti-emulation
  - Anti-heuristics
  - Anti-goat

Anti-heuristic Viruses

- Scanning a PE file for virus code patterns is not always as simple as using regular expressions,
  - especially when detecting new viruses or new variants
  - E.g., metamorphic viruses
- Search heuristics have been developed that can find suspicious code without having exact patterns from a pattern database
  - Static heuristics are used in scanners to analyze executable files
  - Dynamic heuristics analyze code running under emulation
- In addition to anti-emulation techniques directed against the dynamic heuristics, virus writers learned to write viruses in such a way as to evade the static scanners

Anti-heuristic Techniques

- It is common for virus code to be appended, or placed in the last section in the PE file
  - Scanners are programmed to flag PE files in which the entry point is directed to the last section of the file
  - This can cause a false positive for self-extracting archives, in which the extractor code is often at the end, so other heuristics must be used in combination with this one
- Virus writers responded by adding some of their own data sections after their code section, so the entry point was not the last section any more,
  - e.g., the Resure virus
  - Peter Szor, "Attacks on Win 32 – Part 2" for more details

Anti-heuristic Techniques

- Another way to append virus code without having the entry point be in the last section of the PE file
  - place the beginning of the virus in the slack area at the end of the host program code section,
  - …with a jump to the appended virus code at the end
- Emulators have been designed to detect jumping from one PE file section to another

Anti-heuristic Techniques

- The EPO (entry-point obscuring) techniques that we studied previously are also anti-heuristic techniques, as they make it hard for a scanner to detect that control passes to a virus
  - Import Address Table (IAT) replacement
  - Call hijacking
  - Replacing an arbitrary call with a jump into the virus
- Modern scanners cannot just look at entry points and the top and bottom of PE files
  - Scanning is getting more expensive as a result

Armored Viruses

- Armored virus techniques fall naturally into five categories, corresponding to the five tools they are designed to combat:
  - Anti-disassembly
  - Anti-debugging
  - Anti-emulation
  - Anti-heuristics
  - Anti-goat

Anti-goat Viruses

- Anti-virus software often uses *goat files*, a.k.a sacrificial goats, which are dummy files whose infection will signal the presence of a virus
  - Short, simple files, of known content
  - Easy to find a virus within them
  - Scattered around the disk in various file types that are prone to infection, e.g. *.exe, *.vbs, *.com
  - Also in various sizes, as viruses often infect only files with a certain minimum size
- Detecting which goat files are infected, and which are not, helps identify the virus

Anti-goat Techniques

- An anti-goat virus will try to detect goat files and avoid infecting them
- Files are examined for goat file characteristics:
  - Lots of no-ops and do-nothing instructions
  - Clusters of files with sequential numbers in their names, e.g. abcd0001.vbs, abcd0002.vbs, etc.
- As with all armored virus techniques, the point is not to be the mythical "undetectable virus", but just to slow down detection and analysis

Retroviruses

- In nature, a retrovirus replicates in a different manner than normal viruses and is thus partly immune to many antiviral drugs
  - The HIV retrovirus attacks the immune system
- Computer retroviruses directly attack anti-virus software in an effort to make themselves immune
- Szor, section 6.3: "A *retrovirus* is a computer virus that specifically tries to bypass or hinder the operation of an antivirus, personal firewall, or other security programs."

Vulnerability to Retroviruses

- Most of us log on to our personal computers with administrative capabilities
  - More convenient than having to change your login every time you perform an administrative task
  - Everyone was an administrator under DOS
- This gives a retrovirus the same administrative capabilities, meaning that it can kill anti-virus processes, remove anti-virus files, etc., just as you could
- Paves the way for other viruses to work freely
  - Might be the only function of a given retrovirus

Retroviruses: Direct Attack on Security Software

- With admin capabilities, a retrovirus can kill the processes that it recognizes as being AV or firewall software, behavior blockers, etc.
  - A *behavior blocker* is a background process, from the OS or from an AV package, that prevents certain suspicious behaviors, such as changing the interrupt chain or doing a disk write to an existing executable file
- Sometimes, settings in firewalls and AV monitors can be changed to bypass them without killing them (stealthier than killing)
- Can also delete AV or firewall files from disk

Retroviruses: Attack on Security Software Files

- Many AV programs maintain an *integrity checking database* full of checksums and file sizes for various system files
- Retroviruses can attack this database:
  - Delete the files (not too stealthy)
  - Modify the files so that checksums and sizes must be recomputed for infected files; this hides the infection by storing the new sizes and checksums
  - Replace the database with a modified database that prevents virus detection, causes virus misidentification, and even launches viruses (i.e. database is now a Trojan horse)

Example: Altering Integrity Database Entries

- The IDEA.6155 virus was designed to infect *.COM and *.EXE files while escaping detection in the integrity database:
  - The integrity database checksum record “thisfile.exe 23f7e65b” would be altered to “lhisfile.exe 23f7e65b” after infection of thisfile.exe
    - notice the change to the first character from “t” to “l”
  - AV integrity checker concludes, on its next scan, that thisfile.exe must be a new file, so it computes a checksum of the infected file and adds a new record to the integrity database: “thisfile.exe 269b7fc2”
  - Infected file now seems to have integrity
  - Full scan will be required, searching for virus patterns, etc., to find the problem; full scans are not done often because of time constraints

Retroviruses: Indirect Attacks on AV Programs

- Older versions of AV programs sometimes placed an integrity check record at the end of a validated file, with encrypted checksums
  - Files with a mark did not need to be scanned again
  - Cut down on scanning time by only scanning modified or newly created files
- The Tequila virus removed this record from files it infected, so that infection would not cause an integrity check failure
- Simpler non-cryptographic checksums, such as CRC, can be defeated by appending a few bytes to an infected file that cause its new CRC checksum to match the old one; Hybris worm used this technique on PE files, which had to be restored (could not be repaired)

Retroviruses: Deterring AV Use

- A retrovirus can attack analysis tools used by anti-virus researchers
- A retrovirus can remain dormant until it detects AV software, then start damaging the system
- What would you do if the following message appeared on your screen:

WARNING! Infected System!

Virus will do no harm unless you install anti-virus software. Then it will destroy your files!

Assignment

- Read Szor, Chapter 7 through section 7.5 (virus encryption techniques) before the next lecture