

Malice, Exploitation, and Infection

An Overview of Computer Viruses and Malware

Bill Harrison
MU Department of Computer Science

Hi, I'm Bill, glad to meet you...

▶ **Ph.D 2001, UIUC**

- ▶ Thesis: Modular Compilers and Their Correctness Proofs
- ▶ Post-doc, Oregon Graduate Inst. (OGI/OHSU) '00-'03

▶ **Joined MU-CS faculty Fall 2003; Associate Professor since last May**

- ▶ NSF CAREER Award “Automated Synthesis of High-Assurance Security Kernels” in June 2008
- ▶ Director of High Assurance Security Kernel Lab
- ▶ Research interests: Computer Security, Programming Language Design, Formal Verification





Some History

- ▶ June 2008: University of Missouri designated as a *Center for Academic Excellence in Information Assurance* by the National Security Agency
 - ▶ Allows us to apply for scholarships, research funding, etc.
 - ▶ “Information Assurance” = Security
- ▶ February 2010: Information Security and Assurance Center (ISAC) founded in Engineering College
 - ▶ Encourage interdisciplinary research in IA at MU
 - ▶ Expand & enrich IA education at MU
 - ▶ Attract high quality students and faculty





A unique capability: CyberZou

- ▶ “CyberZou” is a special laboratory for research and education on Malware Analysis and Defense
 - ▶ Malware = **Malicious Software** : viruses, worms, etc.
 - ▶ Menagerie of known malware
 - ▶ I.e., a “Zou” as in Mizzou 😊
 - ▶ Students can learn anti-malware techniques
 - ▶ Coursework not typically found in academia
 - ▶ Isolated network to prevent accidental unpleasantness
- ▶ Opportunities for innovative interdisciplinary classes
 - ▶ NSF proposal to *Innovations in Engineering Education, Curriculum and Infrastructure* program
 - ▶ “...systems level thinking and the interaction of biology and engineering”



Just FYI*



INTELLIGENCE COMMUNITY **Virtual Career Fair**



U.S. citizenship is required.

Meet the Intelligence Community Online

The United States Intelligence Community (IC), an integrated network of agencies that work together to protect our nation's security, is seeking a culturally diverse, technologically savvy and skilled workforce for exciting careers in a number of fields. Join us at the IC Virtual Career Fair to explore career opportunities, chat with recruiters and subject matter experts, and learn how to apply for job openings.

Thursday, February 19, 2 p.m. – 8 p.m. ET

Registration opens January 15. **Go to [ICVirtualFair.com](https://www.icvirtualfair.com)**

Space is limited! To guarantee your entrance into this event, pre-registration is highly encouraged.

***US Citizens only**

More FYI

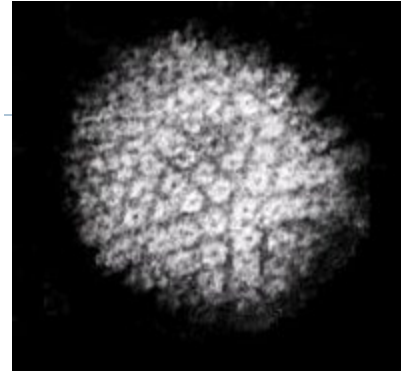


Career opportunities are available in a variety of fields, including:

- Clandestine Services/Intelligence Collection
- Computer Science/Computer Engineering
- Cybersecurity/Information Assurance
- Data Management Specialist
- Data Scientists
- Engineering - Aerospace, Aeronautical, Chemical, Electronics, Mechanical, Nuclear and Systems
- Graphic Design and Multimedia Specialist
- Foreign Languages - Language Analysts, Officers/Instructors, Contract Linguists
- Information Technology - Software/Application Developer, Network Engineer, Mobile Application Developer, User Experience Architect
- Intelligence Analysis and Production - Analytic Methodology, Maritime, and Technical Analysis (Geodetic Surveyor, Aeronautical, Bathymetry, Photogrammetry, Geodetic/Earth Science, Geodetic Orbit Scientist, Cartography and targeting)
- Mission Support - Accountant, Auditor, Budget Analyst, Data Analytics, Facilities, Financial Specialist, Human Resources, Logistics, Police Officer, Security Specialist, Training
- Science and Mathematics (Physics, Chemistry, Biology, Materials Science)
- Visual Info Specialist
- Watch/Operations Officer



What is a virus?



- ▶ A virus is a sub-microscopic particle (ranging in size from about 15–600 nm)
 - ▶ infects the cells of a biological organism
- ▶ Viruses can replicate themselves only by infecting a host cell. They therefore cannot reproduce on their own.
- ▶ Viruses consist of genetic material contained within a protective protein coat called a capsid.
 - ▶ They infect a wide variety of organisms: both eukaryotes (animals, plants, protists, and fungi) and prokaryotes (bacteria and archaea).





What is a computer virus?

- ▶ A virus (whether biological or computational) is self-replicating
 - ▶ i.e., it reproduces copies of itself within some host environment
 - ▶ alternatively, host cells or host OS or programs
- ▶ What is “self-replicating code”?
 - ▶ most of the programs we write strictly separate “program” from “data”
 - ▶ but this separation is artificial





Are Programs Data?

- ▶ There are a number of commonly known applications/languages that manipulate code
 - ▶ compilers: take an input program within a source language, analyze it, and translate it into a program in a target language
 - ▶ metaprogramming/staged languages
 - ▶ LISP/Scheme: include quasiquote (‘) constructs designed for writing programs that construct programs
 - ▶ MetaML, MetaOCaml, Template Haskell, Jumbo,...
 - ▶ Run-time code generation: Tempest, Dynamo,...





Programs ARE data at the machine level

Machine code bytes

```
B8 22 11 00 FF
01 CA
31 F6
53
8B 5C 24 04
8D 34 48
39 C3
72 EB
C3
```

Assembly language statements

```
foo:
movl $0xFF001122, %eax
addl %ecx, %edx
xorl %esi, %esi
pushl %ebx
movl 4(%esp), %ebx
leal (%eax,%ecx,2), %esi
cmpl %eax, %ebx
jnae foo
retl
```

Instruction stream

```
B8 22 11 00 FF 01 CA 31 F6 53 8B 5C 24
04 8D 34 48 39 C3 72 EB C3
```



Early Virus: Elk Cloner on Apple II, 1982



Every 50th use of the infected diskette would print out

```
Elk Cloner:  
    The program with a personality  
  
It will get on all your disks  
It will infiltrate your chips  
Yes it's Cloner!  
  
It will stick to you like glue  
It will modify RAM too  
Send in the Cloner!
```

original source code:

▶ <http://www.skrenta.com/cloner/clone-src.txt>



Malware: Unifying Nomenclature

- ▶ **Virus:** code that recursively replicates itself
 - ▶ possibly evolving functionality
 - ▶ infect host or system area
 - ▶ or, modify/transform existing applications
- ▶ **Worm:** viruses whose primary vector is network
 - ▶ usually standalone program
- ▶ **Logic Bomb:** programmed malfunction in legitimate application
 - ▶ E.g., self-deleting applications
 - ▶ E.g., Nokia “Mosquitos” game sent messages at premium rates when played





The Challenge of Virus Detection

- ▶ A virus is simply a program
 - ▶ i.e., a sequence of instructions located on, say, a disk
- ▶ A disk is really just a sequence of bytes
- ▶ Q: how do I tell if virus **p** is located on disk **d**?
- ▶ Virus detection is a form of string matching
 - ▶ e.g. Boyer-Moore string matching algorithm is roughly $O(m)$ where m is the number of text characters
 - ▶ i.e., number of bytes on disk
- ▶ Not practical to run Boyer-Moore
 - ▶ viral infections may use obfuscation techniques to hide





Heuristics for Virus detection

- ▶ File size: has a known application changed size?
- ▶ Changes in behavior
 - ▶ because infections rewrite (parts of) a host application, it can cause changes in behavior
 - ▶ e.g., the program mysteriously crashes
- ▶ Initial code changed in application
- ▶ All of these presuppose knowledge of “standard” applications
- ▶ Use “decoy” files



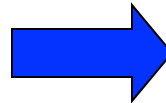
Decoy “Goat” Files



Viruses frequently use
“padded” areas to store
their code; e.g, 04lh or “A”

Changes in “goat” indicate
an infection

```
41 41 41 ... 41
41 41 41 ... 41
41 41 41 ... 41
41 41 41 ... 41
41 41 41 ... 41
41 41 41 ... 41
41 41 41 ... 41
```



```
41 41 41 ... 41
41 41 41 ... 41
41 41 41 ... 41
2E FD 16 ... E9
41 41 41 ... 41
41 41 41 ... 41
41 41 41 ... 41
```



The Virus Writer's POV

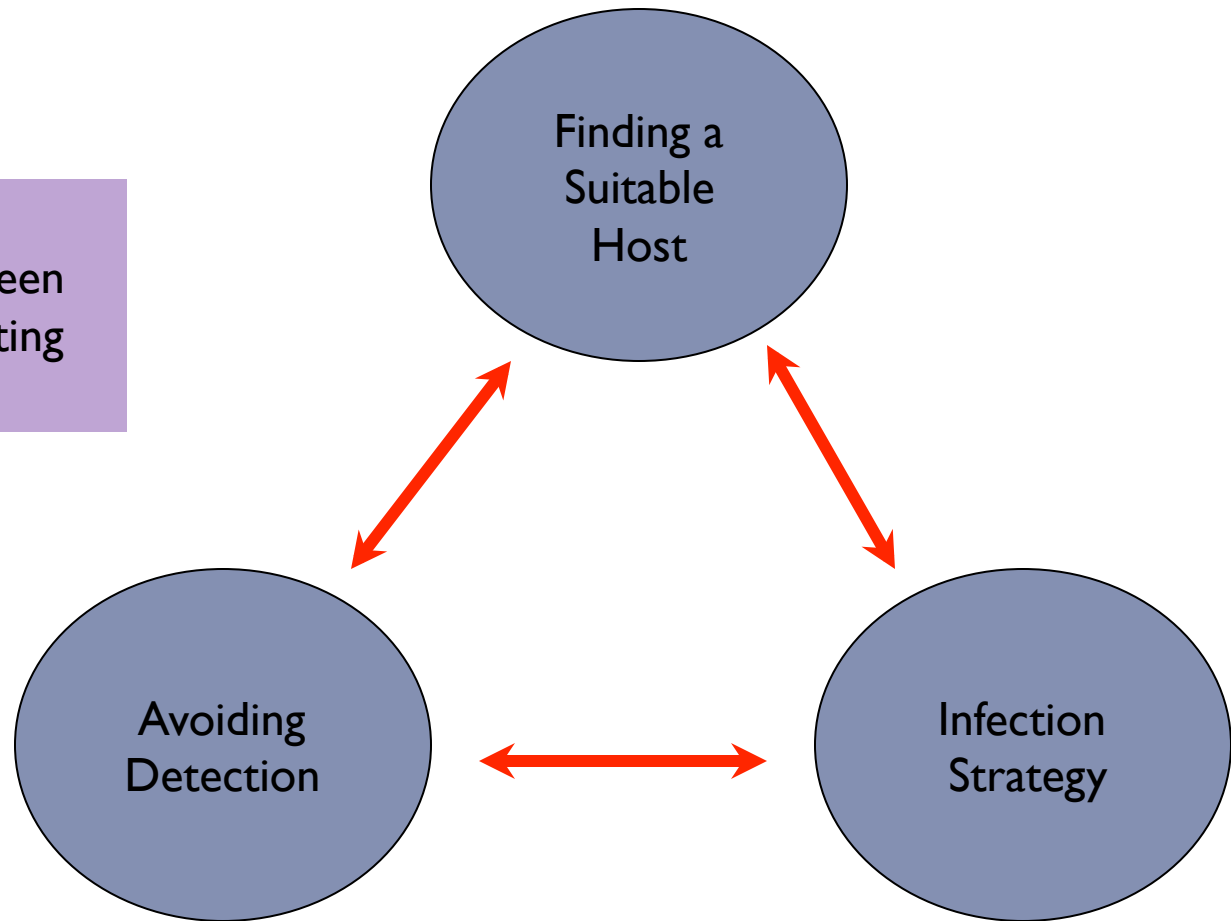
- ▶ The questions that must be answered for any virus are *where do I store my code? Once in place, how does my virus get control (i.e., executed)?*
- ▶ Obfuscation is important
 - ▶ *How do I evade virus scanners?*
 - ▶ one could certainly add a new file “joes-virus.exe”
 - ▶ ...but such an approach could be easily foiled
- ▶ Basic obfuscation idea:
 - ▶ locate a “host” application
 - ▶ automatically modify its code in some manner to include my virus code
- ▶ Challenge: *how do I pick a host app to infect?*



Design Trade-offs for Virus Writers

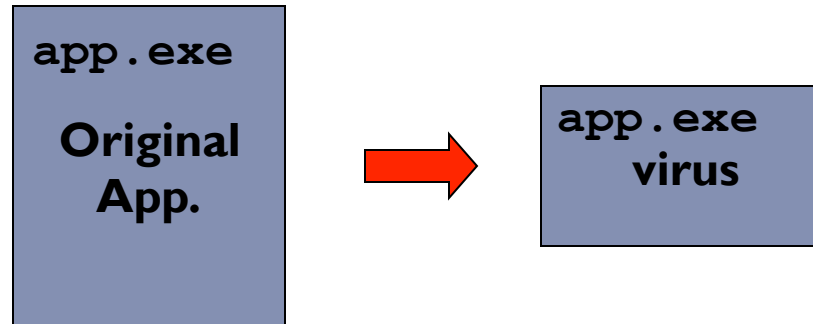


There is a
tension between
these competing
concerns





Overwriting Virus

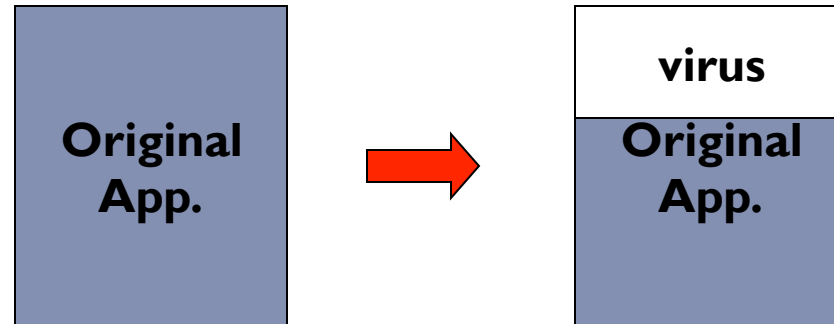


- ▶ Idea: Replace file contents of “`app.exe`” with my own code
- ▶ Pros: simple technique
- ▶ Cons: easily spotted by scanners
 - ▶ size of file almost certainly changes
 - ▶ if file name “`app.exe`” is hardwired in infection strategy, then it’s especially obvious





Overwriting Virus (redux)

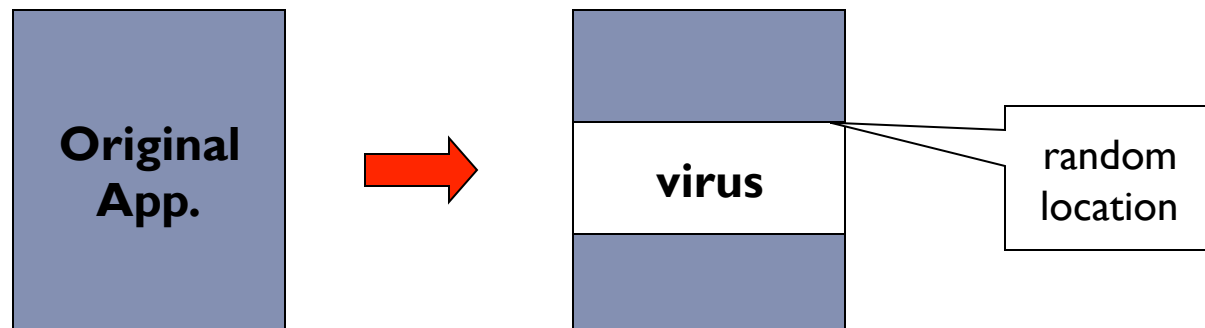


- ▶ Idea: Infect an application with more code
- ▶ Pros: simple technique slightly less obvious
- ▶ Cons: still easily spotted by scanners
 - ▶ virus detection becomes “string matching problem”
 - ▶ N.b., the entry point of the virus is always the same!





Overwriting Virus (re-redux)



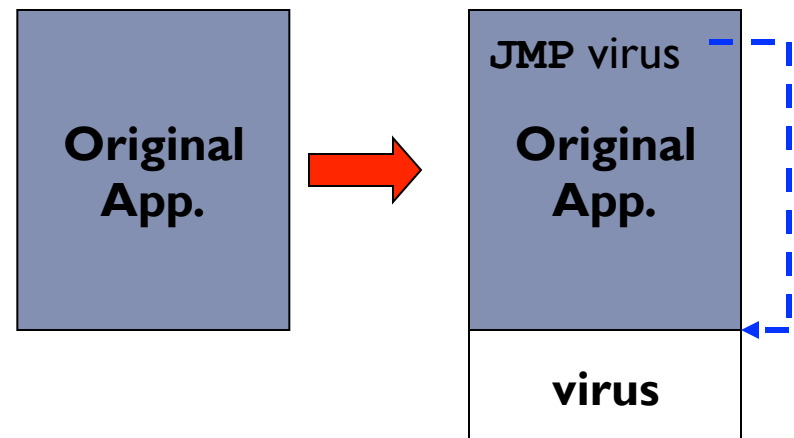
- ▶ Idea: put virus at random location within host application
 - ▶ Pros: less detectable
 - ▶ virus checkers look at “likely” virus locations
 - ▶ Cons:
 - ▶ more error prone (i.e., can the virus execute? Will executing the original app crash?)
 - ▶ Ex: Omud virus actually used this brute force approach
-





Appending Viruses

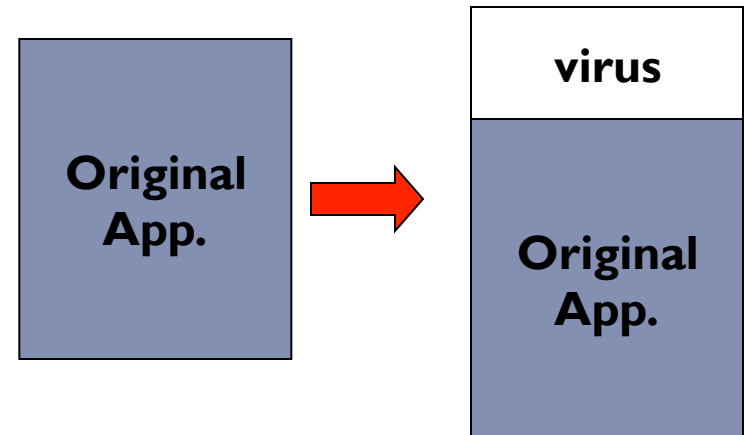
- ▶ Add virus code to the end of host application
- ▶ Then, overwrite first location in infected application with a “JMP” instruction to the code
- ▶ Pros: can save overwritten code to be less obvious; original functionality can be uncompromised
- ▶ Cons: file size changes





Prepending Viruses

- ▶ Add virus code to the beginning of host application
- ▶ Pros:
 - ▶ Simple
 - ▶ Virus can be written in a high-level language like C, for example
- ▶ Cons
 - ▶ can be detectable in similar manner to overwriting viruses

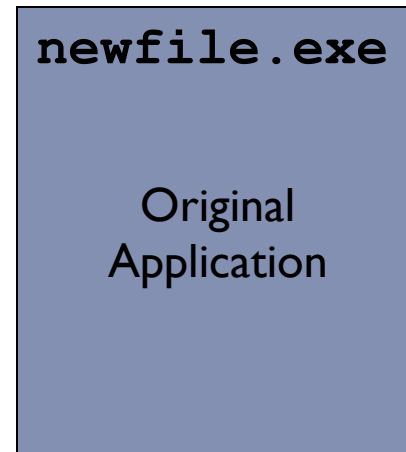
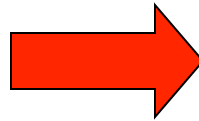
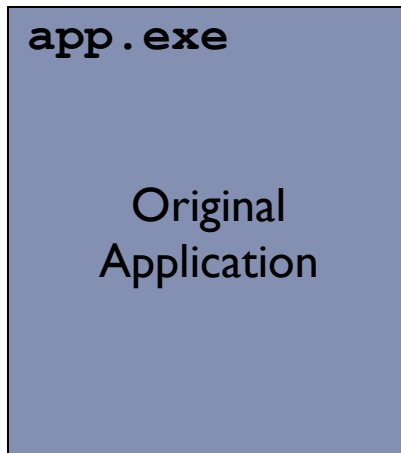




Prepending Viruses (redux)

app.exe

```
int main () {  
    do malicious stuff  
    system("newfile.exe");  
    return 0;  
}
```



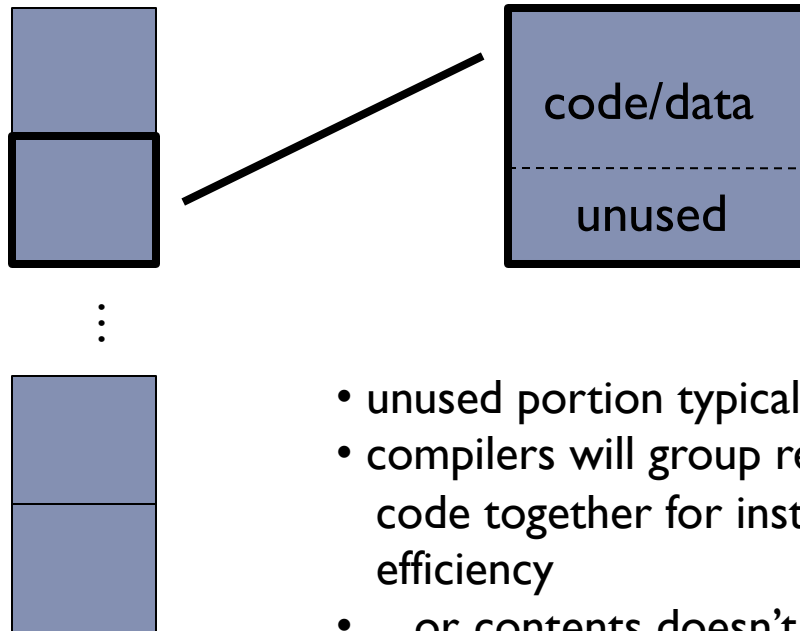
- HLL version less detectable
 - C prog passes args onto original
-





Where do cavities come from?

A file is represented on a disk as a collection of disk blocks

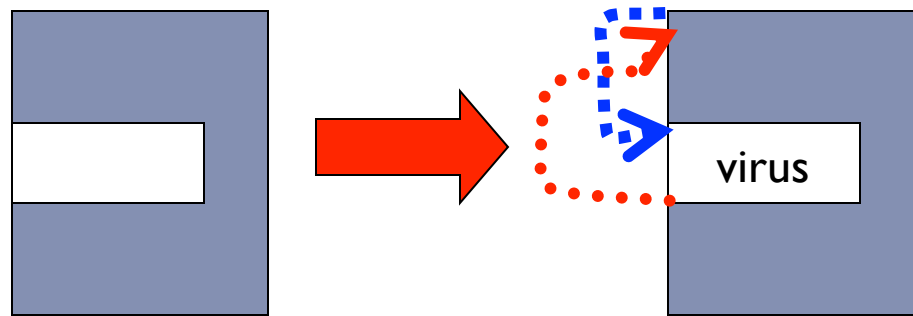


- unused portion typically zeroed out
- compilers will group related chunks of code together for instruction cache efficiency
- ...or, contents doesn't quite need all of the block





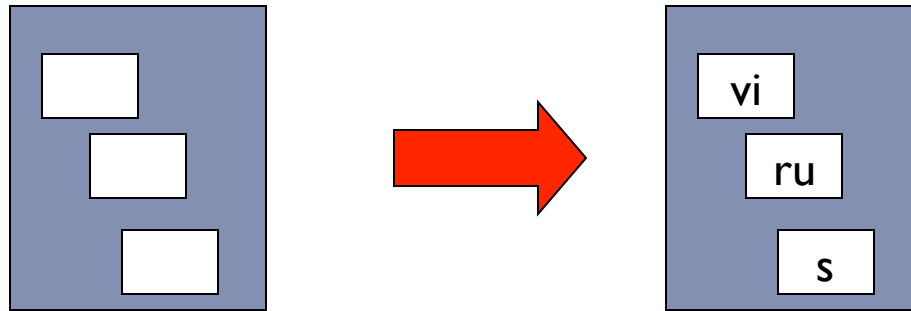
“Cavity” Viruses



- ▶ Find a probably unused portion in application
 - ▶ sequences of 0's, etc., created by compilers for instruction alignment
- ▶ write virus code within that “cavity”
 - ▶ ...with jumps to/from cavity code
- ▶ Big Pro: file size unchanged; original application functionality unchanged
- ▶ Con: have to find a big enough cavity



“Cavity” Viruses, redux

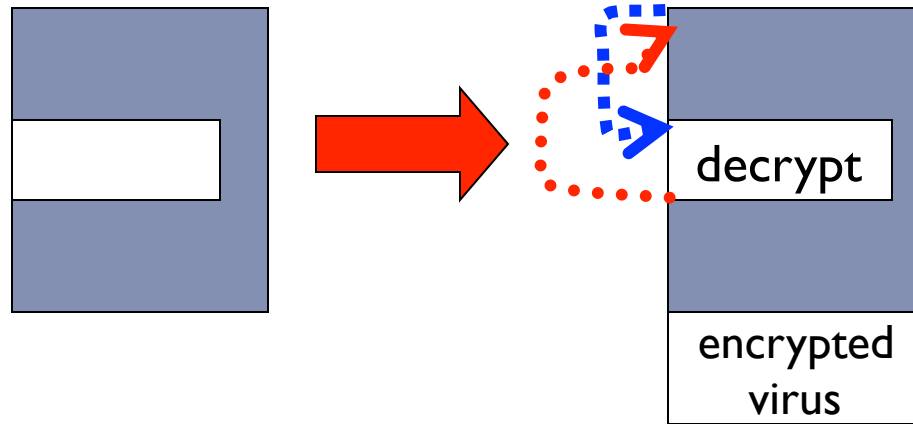


- ▶ **Fractionated cavity viruses** find multiple unused portions in host application
 - ▶ write virus code within those “cavities”
 - ▶ ...with jumps to/from cavity code
- ▶ Big Pro: file size unchanged; original application functionality unchanged
- ▶ Con: more intensive analysis \Rightarrow bigger virus





Decryptor viruses



- ▶ (Sort of) complement of the compressor viruses
- ▶ the virus code is encrypted and stored (e.g., appended)
- ▶ A decryptor program is injected into the host application using some infection technique
- ▶ the virus is decrypted and executed at run-time
- ▶ virus hidden by encryption, but may be difficult to maintain file size



Critical Concern for Virus Writer



obfuscate

One entry found for **obfuscate**.

Main Entry: **ob·fus·cate** 🗣️

Pronunciation: 'äb-f&-"skAt; äb-'f&s-"kAt, &b-

Function: *verb*

Inflected Form(s): **-cat·ed; -cat·ing**

Etymology: Late Latin *obfuscatus*, past participle of *obfuscare*, from Latin *ob-* in the way + *fuscus* dark brown -- more at **OB-**, **DUSK**

transitive verb

1 a : DARKEN b : to make obscure <*obfuscate* the issue>

2 : CONFUSE <*obfuscate* the reader>

intransitive verb : to be evasive, unclear, or confusing





Varieties of Obfuscation

- ▶ **File attributes:** make the infected file look as un-infected as possible
 - ▶ cavity viruses, appending viruses,...
 - ▶ rules of thumb: make virus code as small as possible, leave host application's behavior as “normal” as possible
- ▶ **Code Obfuscation:** hide/transform code within infected
 - ▶ splitting virus code up (e.g., multiple cavity infection)
 - ▶ encrypting the code
 - ▶ “polymorphism”
- ▶ **Entry-point Obfuscation (EPO):** don't put the virus code in the most obvious place to look
 - ▶ i.e., the entry point of a host application





“Polymorphic” Viruses

- ▶ **Polymorphic: many formed**
 - ▶ Greek: “poly” = “many”, “-morphic” = “-formed”
 - ▶ Term has a number of different connotations across Computer Science
- ▶ **Polymorphic virus = single virus with many different instances**
 - ▶ typical polymorphic virus replicates into different, equivalent forms
 - ▶ ...using padding with NOP's/garbage instructions and other more complicated strategies
- ▶ **Purpose: defeat the pattern recognition capabilities of virus scanners**



Ex: 1260 Virus



From the decryptor code of 1260

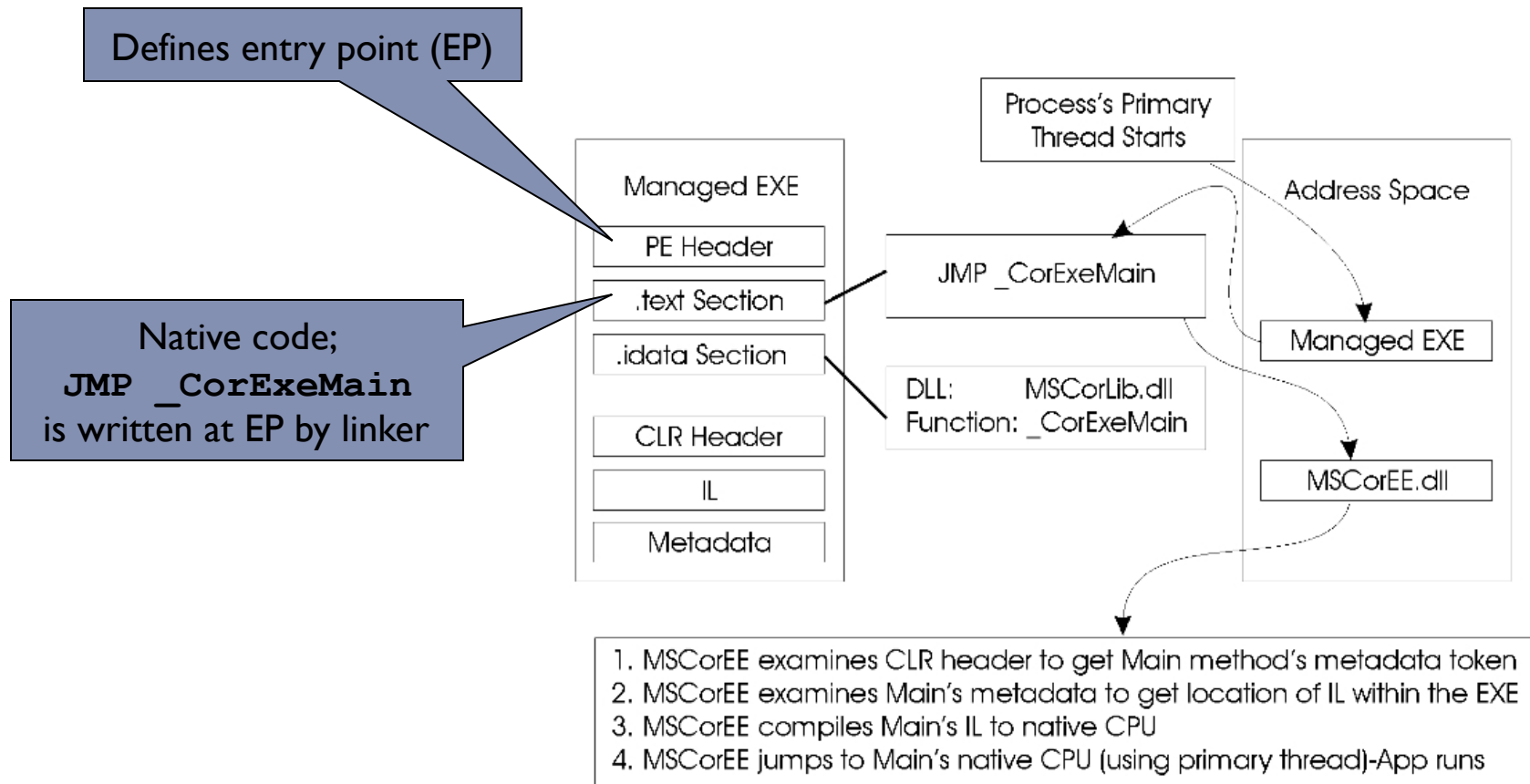
```
inc di  
nop  
clc  
inc ax
```

*one, both or neither
may occur within virus
body; have no function other
than making detection harder*

akin to evolutionary process of random variation
as a means of avoiding disease

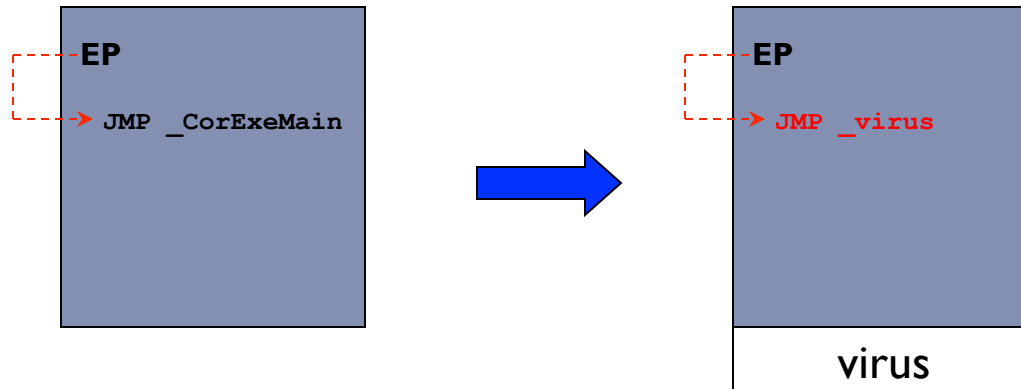


.Net Structure: Portable Executables (PE)





Obfuscated Tricky Jump



- ▶ The OTJ doesn't change the entry point in the host application's header
 - ▶ .Net style PE header
- ▶ Rather it changes the code referenced by the header
 - ▶ i.e., in the program (.txt) part
- ▶ Avoids detection because it (partially) hides the start of the application
- ▶ Ex: W32/Donut





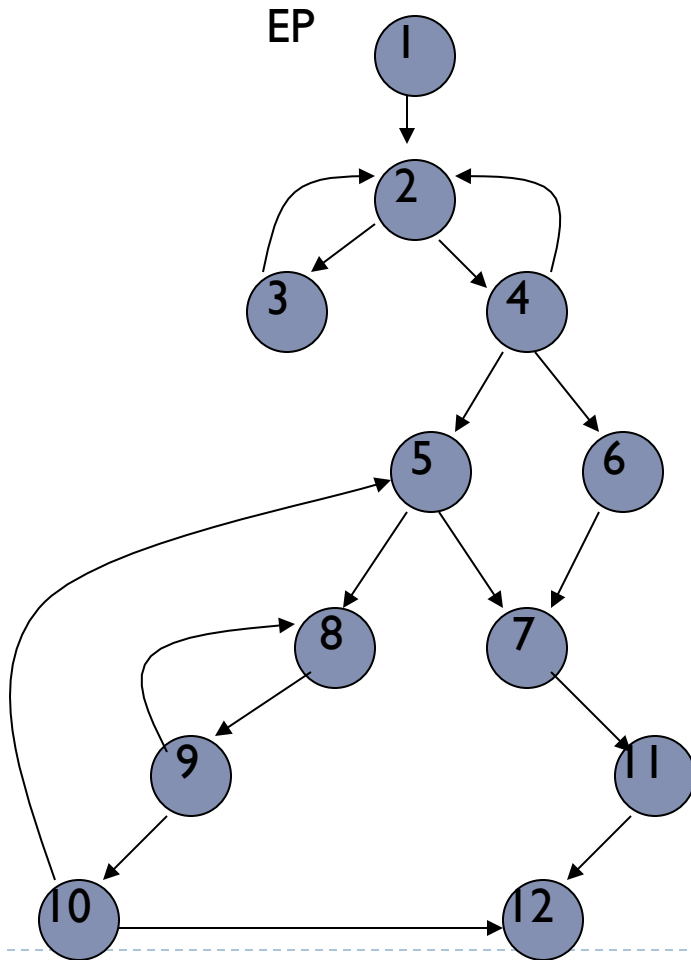
Entry Point Obscuring (EPO) Viruses

- ▶ One means of detecting a viral infection is to spot an unexpected change in/at the entry point of a potential host application
- ▶ EPO viruses “cover their tracks” by not changing the EP or the code at the entry point
 - ▶ This is in contrast to most of the infection techniques we’ve seen so far
- ▶ The idea is to insert the virus code “deeper” into the host application
- ▶ Pro: much trickier to detect
- ▶ Cons: infection is harder to create; more analysis of host may be required; not guaranteed that the virus will run every time (or, at all)





A control flow graph (CFG)



- Recall from compiler construction that code may be view as a directed graph
 - where each node is a “basic block” containing no jumps/calls/etc.



More background: basic block

- ▶ Basic block is a sequence of instructions
 - ▶ ending in control flow change (jump, call, etc.)
 - ▶ no previous control flow change among instr_i
 - ▶ no jumps from other blocks into middle of block
- ▶ Basically: code that is always executed from beginning to end within program

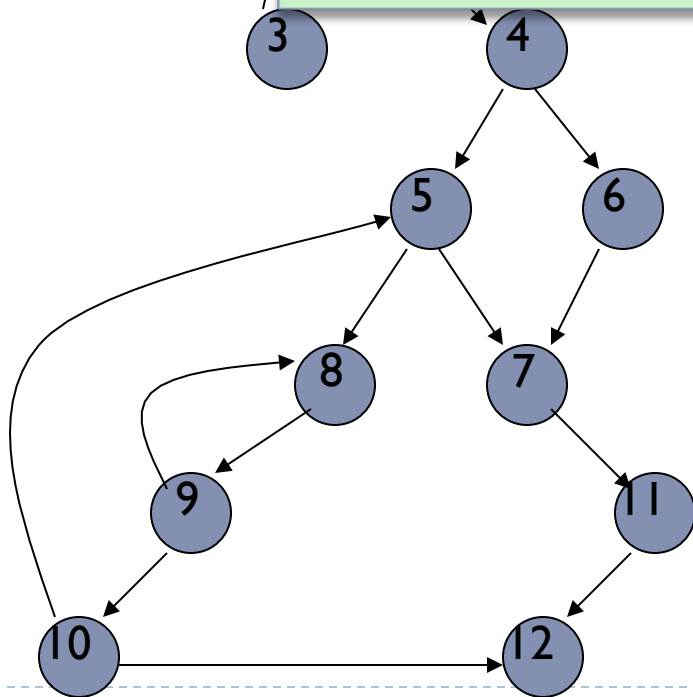
```
instr1;  
...  
instrn;  
JUMP/CALL
```



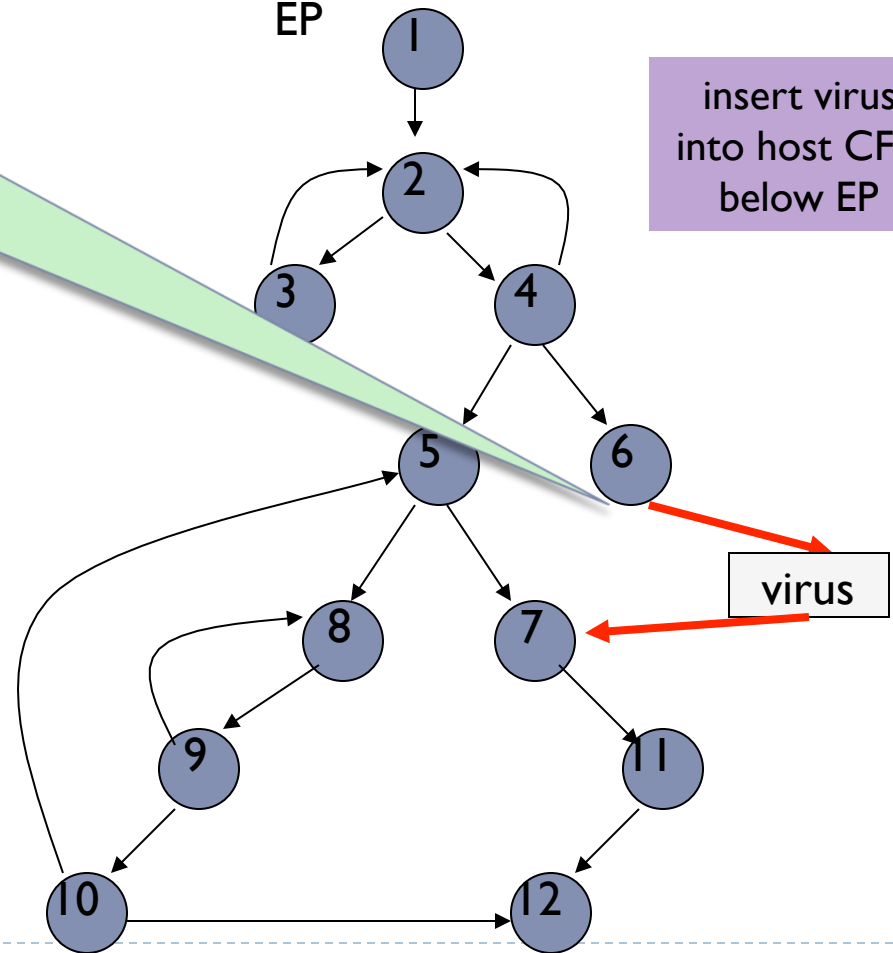
EPO Infection Strategy



The big question for the infection routine: how do I figure out where basic blocks start and end?

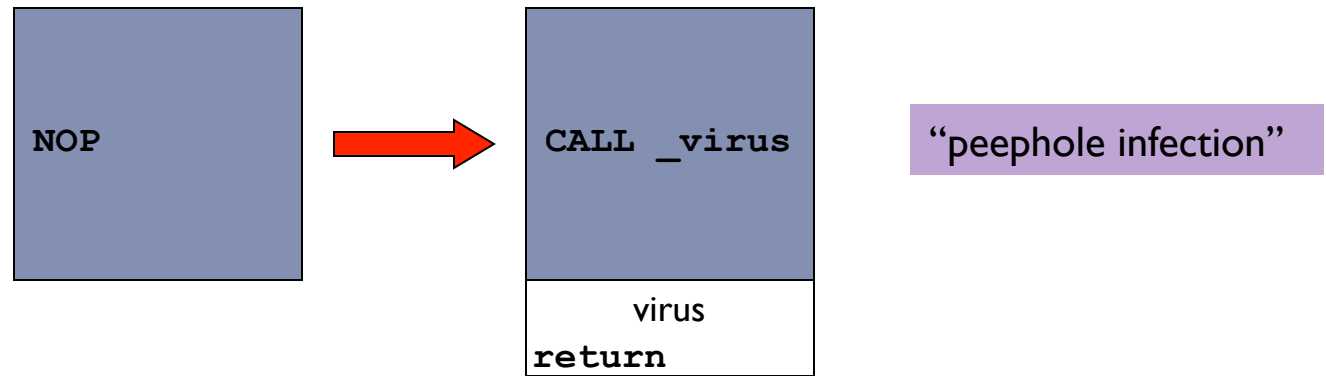


EP





EPO Simple Approach



- ▶ The analysis required to do arbitrary “virus insertions” is well-understood
 - ▶ however, it’s complex (recall your compiler course)
 - ▶ The challenge for the virus writer is how to make an EPO transformation...
 - ▶ **correctly**: so that the host doesn’t behave unexpectedly
 - ▶ **quickly**: so that the virus code can be as small and simple as possible
-





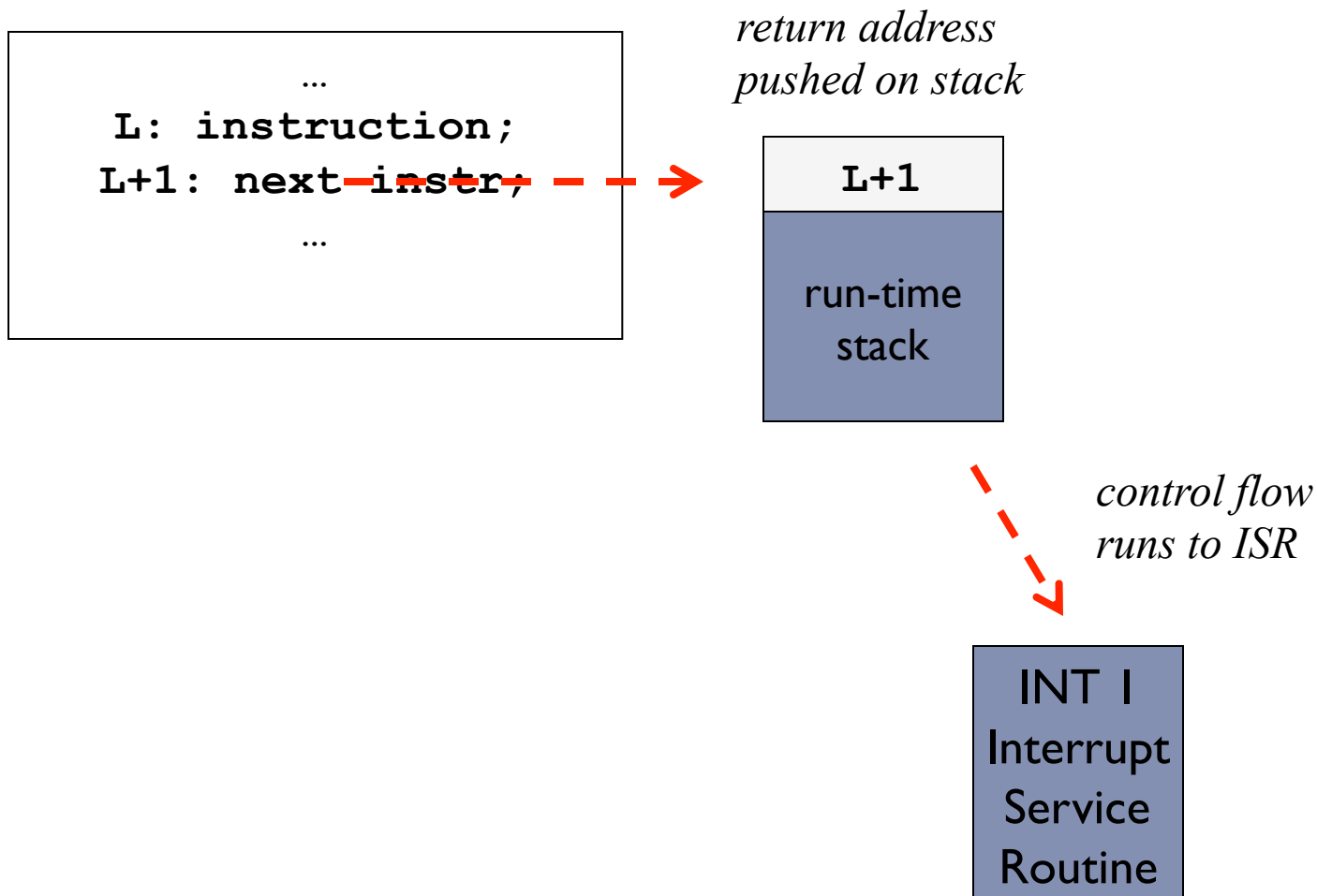
Background: DOS Trace Exception (INT 1)

- ▶ The single step exception occurs after every instruction if the trace bit in the flags register is equal to one.
- ▶ Debuggers will often set this flag so they can trace the execution of a program.
 - ▶ When this exception occurs, the return address on the stack is the address of the next instruction to execute.
 - ▶ The trap handler can decode this opcode and decide how to proceed.
- ▶ Debuggers that use the trace exception for single stepping often disassemble the next instruction using the return address on the stack as a pointer to that instruction's opcode bytes.



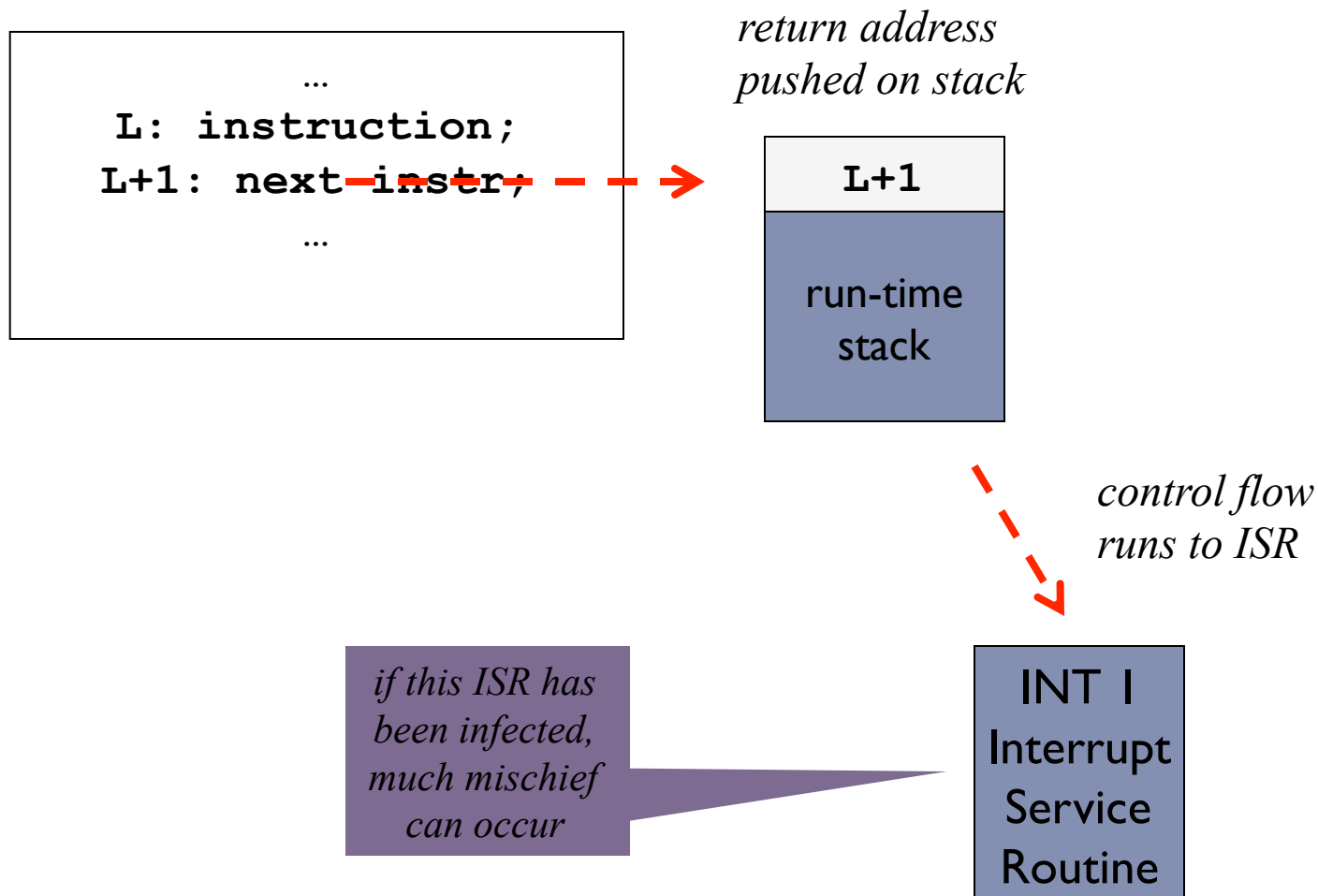


If the trace bit is set...





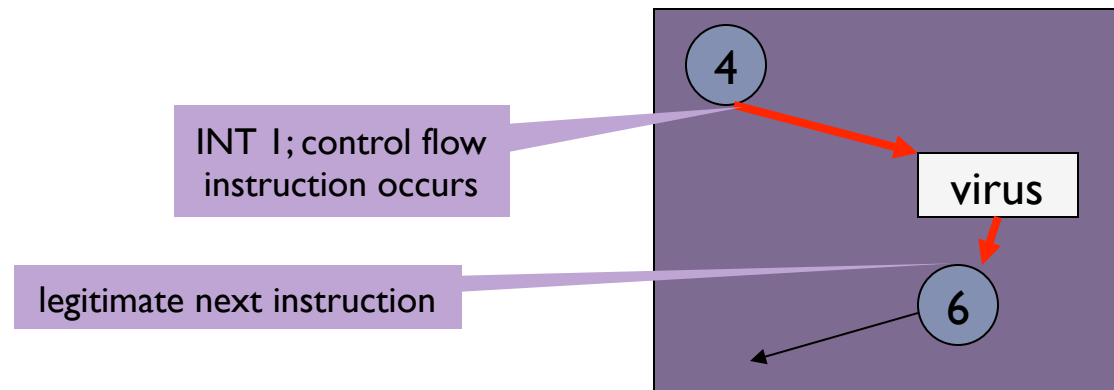
If the trace bit is set...



Ex: Nexiv_Der (“red vixen”) virus*



- ▶ Overwrite INT 1 (TRACE) interrupt service routine
- ▶ When new ISR is called, see if the next instruction is a control-flow instruction (e.g., “JMP”)
 - ▶ If it is, then it knows it is at the edge of a basic block
 - ▶ Call virus code instead
 - will return to legitimate “next instruction” when virus is done



* W32/Perenast (2003) uses standard Windows debugging API similarly



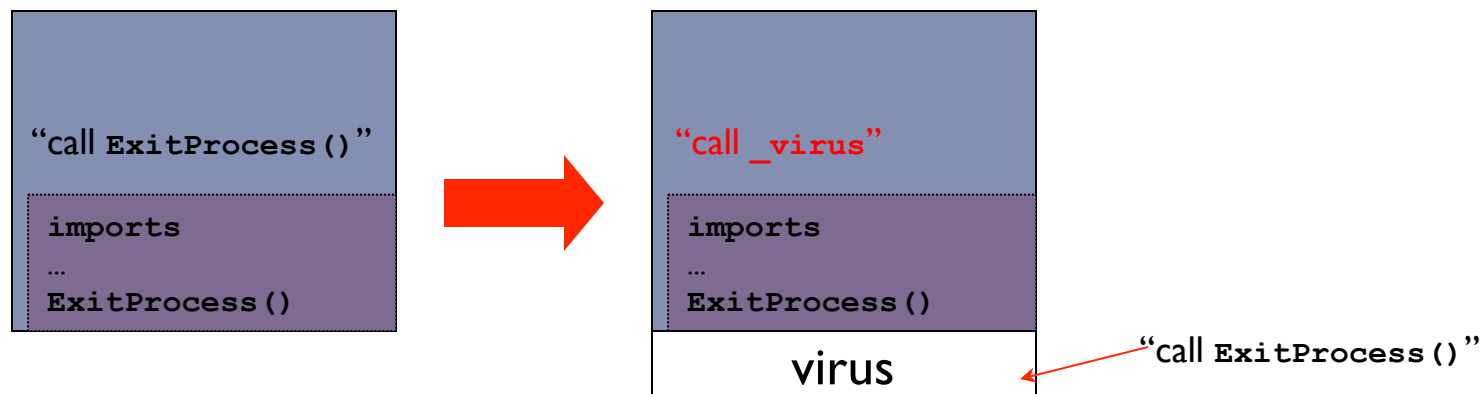
API Hooking

- ▶ Similar to Red Vixen in that a change in control flow is identified and exploited
 - ▶ API hooking identifies a call to an external library instead of overwriting a ISR
- ▶ In Windows PE format, there is an “import directory” that identifies all library calls
- ▶ API Hooking Virus inspects target application for calls referencing these imports
 - ▶ e.g., “`CALL DWORD PTR []`” where PTR refers to imports
 - ▶ Then, replace this call with call to virus code
 - ▶ ...when virus code finishes, continue with “`CALL DWORD PTR []`”





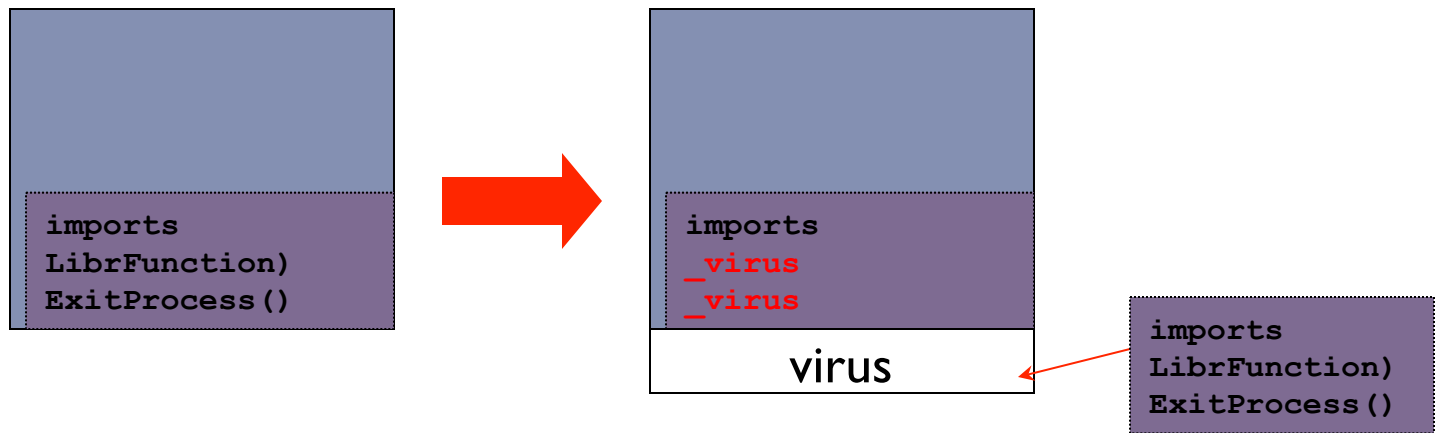
API Hooking redux



- ▶ Replaces call to library function with call to virus in host's code
- ▶ Obscures the entry point of the virus
- ▶ If `ExitProcess()` is used, virus will run more often
 - ▶ usually called upon exit of host applications
- ▶ "Function-call Hooking" attempts to replace call to user subroutine with call to virus
 - ▶ e.g., "CALL Foobar" with "CALL `_virus`"
 - ▶ slightly trickier than you'd think. Why?



Import Directory Corruption



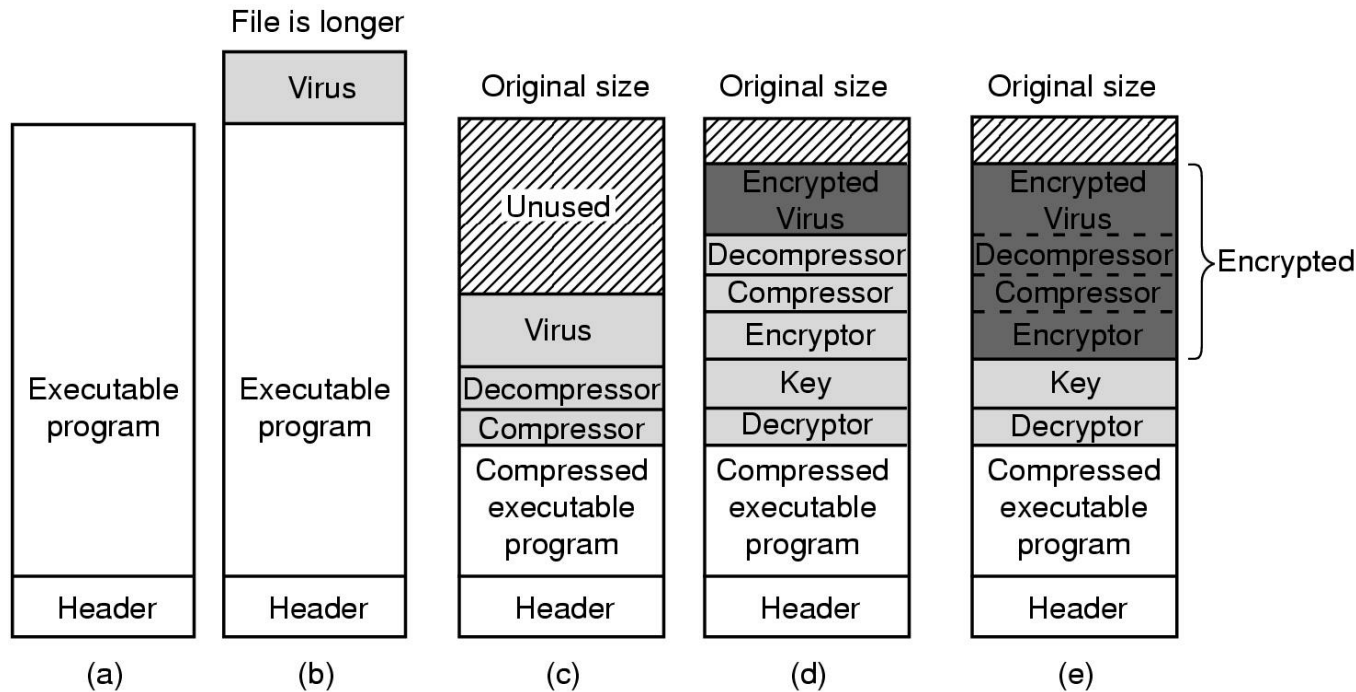
- ▶ Similar to API Hooking
- ▶ Replaces Import Directory Entries with virus entry point
 - ▶ Keeping a copy of original import directory



Polymorphic, Oligomorphic, Metamorphic Viruses



Viruses propagate via infection



- (a) A program
- (b) An infected program
- (c) A compressed infected program
- (d) An encrypted virus
- (e) A compressed virus with encrypted compression code



Virus Techniques

▶ Stealth viruses

- ▶ Infect OS so that infected files appear normal to user

▶ Macro viruses

- ▶ A **macro** is an executable program embedded in a word processing document (MS Word) or spreadsheet (Excel)
- ▶ When infected document is opened, virus copies itself into global macro file and makes itself **auto-executing** (e.g., gets invoked whenever any document is opened)

▶ Polymorphic viruses

- ▶ Viruses that mutate and/or encrypt parts of their code with a randomly generated key



Evolution of Polymorphic Viruses (1)

- ▶ **Why polymorphism?**
 - ▶ Anti-virus scanners detect viruses by looking for **signatures** (snippets of known virus code)
 - ▶ Virus writers constantly try to foil scanners
- ▶ **Encrypted viruses:** virus consists of a constant decryptor, followed by the encrypted virus body
 - ▶ Cascade (DOS), Mad (Win95), Zombie (Win95)
 - ▶ Relatively easy to detect because decryptor is constant
- ▶ **Oligomorphic viruses:** different versions of virus have different encryptions of the same body
 - ▶ Small number of decryptors (96 for Memorial viruses); to detect, must understand how they are generated



Evolution of Polymorphic Viruses (2)

- ▶ **Polymorphic viruses:** constantly create new random encryptions of the same virus body
 - ▶ Marburg (Win95), HPS (Win95), Coke (Win32)
 - ▶ Virus must contain a polymorphic engine for creating new keys and new encryptions of its body
 - ▶ Rather than use an explicit decryptor in each mutation, Crypto virus (Win32) decrypts its body by brute-force key search
- ▶ **Polymorphic viruses can be detected by emulation**
 - ▶ When analyzing an executable, scanner emulates CPU for a time.
 - ▶ Virus will eventually decrypt and try to execute its body, which will be recognized by scanner.
 - ▶ This only works because virus body is constant!

Anti-antivirus techniques



```
MOV A,R1
ADD B,R1
ADD C,R1
SUB #4,R1
MOV R1,X
```

(a)

```
MOV A,R1
NOP
ADD B,R1
NOP
ADD C,R1
NOP
SUB #4,R1
NOP
MOV R1,X
```

(b)

```
MOV A,R1
ADD #0,R1
ADD B,R1
OR R1,R1
ADD C,R1
SHL #0,R1
SUB #4,R1
JMP .+1
MOV R1,X
```

(c)

```
MOV A,R1
OR R1,R1
ADD B,R1
MOV R1,R5
ADD C,R1
SHL R1,0
SUB #4,R1
ADD R5,R5
MOV R1,X
MOV R5,Y
```

(d)

```
MOV A,R1
TST R1
ADD C,R1
MOV R1,R5
ADD B,R1
CMP R2,R5
SUB #4,R1
JMP .+1
MOV R1,X
MOV R5,Y
```

(e)

► Examples of a polymorphic virus

- All of these examples do the same thing

Antivirus software



- ▶ Integrity checkers

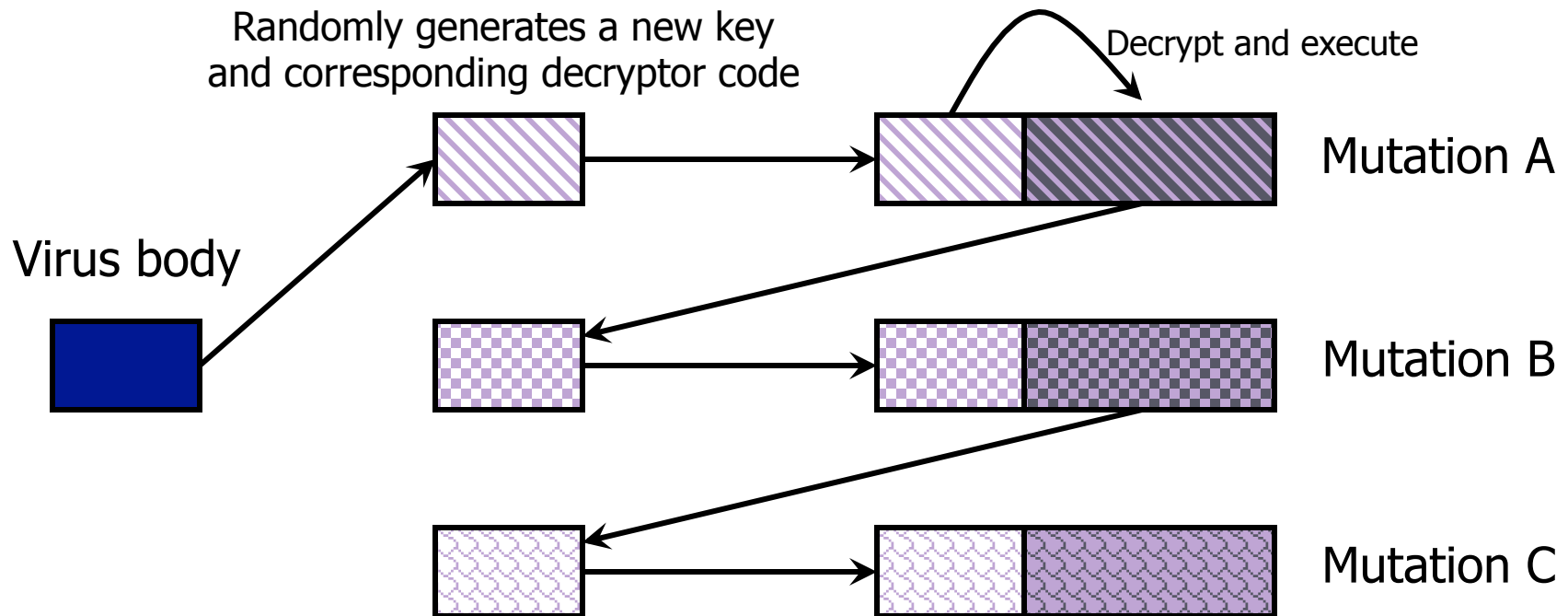
- ▶ use checksums on executable files
- ▶ hide checksums to prevent tampering?
- ▶ encrypt checksums and keep key private







- ▶ Behavioral checkers

- ▶ catch system calls and check for suspicious activity
 - ▶ i.e., record the pattern of system calls generated by an application
 - ▶ compare this with known virus calling patterns
- ▶ what does “normal” activity look like?



Virus Detection by Code Emulation



To detect an unknown mutation   of a known virus , emulate CPU execution of   until the current sequence of instruction opcodes matches the known sequence for virus body 



Metamorphic Viruses

- ▶ Obvious next step: **mutate the virus body**, too!
- ▶ Virus can carry its source code (which deliberately contains some useless junk) and recompile itself
 - ▶ Apparition virus (Win32)
 - ▶ Virus first looks for an installed compiler
 - ▶ Unix machines have C compilers installed by default
 - ▶ Virus changes junk in its source and recompiles itself
 - ▶ New binary mutation looks completely different!
- ▶ Many macro and script viruses evolve and mutate their code
 - ▶ Macros/scripts are usually interpreted, not compiled



Metamorphic Mutation Techniques

- ▶ Same code, different register names
 - ▶ Regswap (Win32)
- ▶ Same code, different subroutine order
 - ▶ BadBoy (DOS), Ghost (Win32)
 - ▶ If n subroutines, then $n!$ possible mutations
- ▶ Decrypt virus body instruction by instruction, push instructions on stack, insert and remove jumps, rebuild body on stack
 - ▶ Zmorph (Win95)
 - ▶ Can be detected by emulation because the rebuilt body has a constant instruction sequence

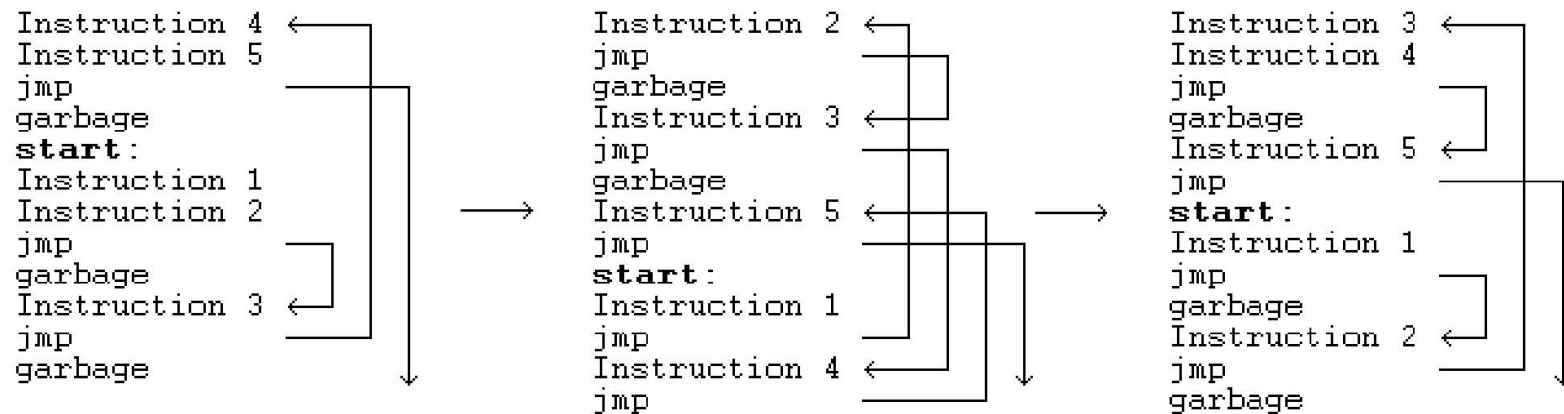


Real Permutating Engine (RPME)

- ▶ Introduced in Zperm virus (Win95) in 2000
- ▶ Available to all virus writers, employs entire bag of metamorphic and anti-emulation techniques
 - ▶ Instructions are reordered, branch conditions reversed
 - ▶ Jumps and NOPs inserted in random places
 - ▶ Garbage opcodes inserted in unreachable code areas
 - ▶ Instruction sequences replaced with other instructions that have the same effect, but different opcodes
 - ▶ Mutate `SUB EAX, EAX` into `XOR EAX, EAX` or
`PUSH EBP; MOV EBP, ESP` into `PUSH EBP; PUSH ESP; POP EBP`
- ▶ There is no constant, recognizable virus body!



Example of Zperm Mutation



► From Szor and Ferrie, “Hunting for Metamorphic”



Defeating Anti-Virus Emulators

- ▶ Recall: to detect polymorphic viruses, emulators execute suspect code for a little bit and look for opcode sequences of known virus bodies
- ▶ Some viruses use **random code block insertion** engines to defeat emulation
 - ▶ Routine inserts a code block containing millions of NOPs at the entry point prior to the main virus body
 - ▶ Emulator executes code for a while, does not see virus body and decides the code is benign... when main virus body is finally executed, virus propagates
 - ▶ Bistro (Win95) used this in combination with RPME



Putting It All Together: Zmist

- ▶ Zmist was designed in 2001 by Russian virus writer Z0mbie of “Total Zombification” fame
- ▶ New technique: **code integration**
 - ▶ Virus merges itself into the instruction flow of its host
 - ▶ “Islands” of code are integrated into random locations in the host program and linked by jumps
 - ▶ When/if virus code is run, it infects every available portable executable
 - ▶ Randomly inserted virus entry point may not be reached in a particular execution





How Hard Is It to Write a Virus?

- ▶ 498 matches for “virus creation tool” in Spyware Encyclopedia
 - ▶ Including dozens of poly- and metamorphic engines
- ▶ OverWriting Virus Construction Toolkit
 - ▶ "The perfect choice for beginners“
- ▶ Biological Warfare Virus Creation Kit
 - ▶ Note: all viruses will be detected by Norton Anti-Virus
- ▶ Vbs Worm Generator (for Visual Basic worms)
 - ▶ Used to create the Anna Kournikova worm
- ▶ Many others