Dangers of IT Monocultures

Ken Birman

Modern mindset…

- Suppose some large enterprise like a bank adopts a highly standard platform configuration
  - "Microsoft Windows Vista on everything"
  - Identical patch levels, fixed list of possible configurations and applications
  - End users aren't permitted to customize at all
    - Want to run the RealPlayer add-on?
    - Just convince the central IT organization
    - Then they'll push an update… *to everyone, with the identical add-on configured in the identical way*
- Argument: offers many benefits….

2

Combine with consolidation?

- Could even take the next step
  - Virtualize everything
  - Run end-user images efficiently on a data center
  - Exploit tricks like copy-on-write page sharing to gain further efficiencies
  - Could still let users "check out" their own images for mobile use, "check back in" when they get back to the wired network
- Claim: could save huge amounts of
  - Money (replace fancy PCs with thin client platform)
  - Electric power and cooling (efficiencies of scale, sharing)
  - Manpower: fewer configurations, more standard hence can automate many tasks

3

How do these claims hold up?

- Claim 1: Will save money on fancy PCs
  - Reality? Probably not…
  - In large quantities, all PCs cost pretty much the same
  - Anyhow, trends suggest that even thin PC would have a fancy multicore processor and a fair amount of local storage capacity
  - Basically, like it or not, you'll own fancy PCs!
- Issue this raises
  - Will applications "use" all that edge computing power?

4

Ways to use client cycles

- Voice interfaces, artificial intelligence tools
  - Already common on 3G telephones like i-phone, Blackberry Storm
  - Talk to your machine… Auto-completion tools for things you type… Powerful design tools for people who do things like CAD-CAM
  - Emerging collaboration / conferencing / brainstorming technologies
- Conclusion?
  - Consolidation might not be such a big win

5

Efficiencies of power

- Clearly this is a win for the kind of applications that Amazon EC2 might host
  - But many of those are actually web data centers for entire companies
  - Notice distinction between consolidation aimed at the client computing platforms and consolidation aimed at data centers with huge numbers of servers
- Conclusion?
  - Could pay off, but depends very much on the kinds of applications you use (and share)

6

Management efficiencies

- Two big benefits here
  - With fewer and less quirky configurations, the local computing repair guy can solve problems faster
  - Can also automate all sorts of tasks under assumption that we have huge numbers of machines running a small set of configurations and applications
- Conclusion?
  - Standardization, consolidation could be a big win by reducing the human cost of management of a system
  - Still would need to deal with hardware failures, network configuration issues, malware infections, etc

7

Robustness of IT monocultures

- Suppose we have a choice
  - Highly diverse, individualized, configurations (like today)
  - Extremely uniform configurations ("monoculture")
- Which is likely to be more robust?


diverse


monoculture

8

Understanding "threats"

- How does malware propagate?
  - We've discussed the version you download and install voluntarily (like Facebook Beacon)
  - Many web sites are infected with malware
- What about viruses?
  - These typically exploit flaws in the O/S, network and applications
  - Many such flaws are known

9

Famous malware examples

- Robert Morris: Internet Worm
  - Supposedly was written for fun, to create an innocent new "life form" that would live in the network
  - Infected machines mostly through a bug in SendMail, guessing passwords, and by just copying itself within groups of mutually trusting machines
  - Slight flaw: Morris worried that sys admin might mark uninfected machines as infected to block spread. To work around this, his code ignored "already infected" marker 1 time in 7…
- Rapidly spread… infected about 6,000 machines at peak (but at the time this was 10% of the Internet)

10

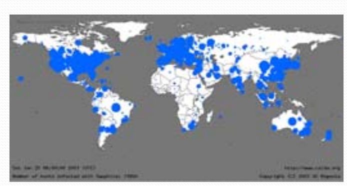
Famous malware examples

- Code Red: Computer worm/virus released 1/13/2001
  - Exploited a buffer overflow bug
  - Within six days infected roughly 400,000 machines
  - Purpose? Printed HELLO! Welcome to http://www.worm.com! Hacked By Chinese!
- Believed that an Philippine hacker was behind it

11

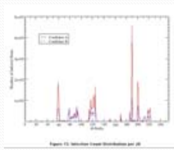
Famous malware examples

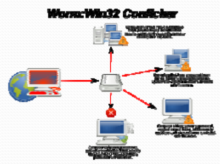
- SQL Slammer
  - Infected Microsoft database systems
  - Issue was a buffer overflow in Microsoft SQL Server
  - Entire worm was 376 bytes in length!
  - Infected 90% of vulnerable machines (about 22,000) in at most 10 minutes
- Image: 30 minutes into Slammer episode… Size of circle is *log* of number of infected hosts…



Famous malware examples

- Conficker C or Downadup: Today's most serious threat
  - Spread by buffer overflow, guessing passwords
  - Also exploited a weakness in USB "mount" software in Windows O/S and attached itself to some standard Windows services, like svchost.exe
  - Believed to have infected at least 1.5 million PCs

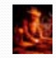




13

Worm/virus goals

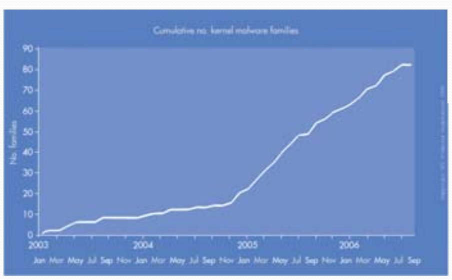
- Confiker believed to be building a "bot net"
  - Like Cloud Computing but for people sending spam
- But other recent attacks have
  - Just shown off ("Kilroy was here!")
  - Mined for corporate, military or other kinds of secrets
  - Damaged your system



- Bottom line: many bad guys with many goals

14

Trends?



15

Many worms "scan" network

- Pick random IP addresses and send infection as a packet
  - If target is susceptible, merely receiving the packet can leave it infected
  - As more and more machines are infected, the coverage of the Internet IP space rapidly grows
- Theoretically, a virus could infect the entire Internet in less than 1 second this way
- Read: "How to Own the Internet in Your Spare Time" for more info. (Staniford; Security 2002, 149-161)

16

Back to consolidation

- Will consolidation make the situation worse, or better?
  - Knee-jerk answer: worse, because if massive numbers of machines run identical software, once something breaks in, it will spread like lightening!
  - But is this really true?

17

How malware gets in the door

- All sorts of "vectors"
  - Buffer overrun exploits
  - Password guessing
  - Unprotected copying among mutually trusting systems
  - USB automount/autoplay interface
  - Application-level flaws
- Mixture of attacks against the O/S and against applications
  - Windows under attack… because there are more Windows PCs
    - Linux is *at least* as vulnerable
  - (*Why do criminals rob banks? Because that's where the money is!*)

18

NSA comment

- Suppose a town had a rash of robberies
  - Investigator discovers that nobody locks front doors
  - His recommendation? "Just lock the front doors"
- Would this eliminate the crime wave?
  - What about the back door?
  - How about windows?
  - The second floor?
  - What if the criminals cut a hole in the wall?

19

Fighting the last battle

- There is a tendency to focus on fixing the hole that was exploited by the last big malware system

But this only helps if you somehow think it was the last big vulnerability

- If your system is riddled with holes, sure you should defend against known threats, but don't delude yourself into thinking this will solve the problem!

20

How consolidation could help

- Realistically, many malware systems exploit configuration bugs
  - Such as passwords that have the default value
  - A common issue for installed software packages
- At least consolidated systems can be configured in a professional, competent manner...

21

How consolidation could help

- Better managed systems are also better patched
  - Central administration team should know which machines have which applications on them
  - And can push security and other patches aggressively
  - If a system isn't patched, don't let it run
- So this will help defend against known threats

22

What about "unknown bugs"

- For example, buffer overrun bugs
  - Like it or not, much of the millions of lines of installed base was written in languages like C
  - With open source movement, anyone leafing through has a chance to find a new "exploit"
  - Even things like printf/scanf are at risk!
- Could try and automatically detect/fix
  - Issue here is that there are so many places to look at
  - Experience with automated bug finders is pretty mixed (best success stories: checking device drivers)

23

Dealing with unknown bugs

- One idea: *synthetic diversity*
  - Suppose that each time the O/S boots, it *randomizes* the numbering of system calls
  - And each time we create a thread, we shift the stack by some random number of bytes
  - We can also pad malloc objects with random extra space
  - And can introduce randomness in the thread scheduler
- Tricks like these can defeat buffer overrun attacks
  - Attacker can't guess buffer address in memory, and can't guess what system calls to execute to load malware!

24

Windows Vista

- Windows automatically uses many of these tools
  - System itself was checked with automated tools to discover many kinds of buffer overrun risks
  - Drivers are automatically checked and tested
  - Employs stack randomization and address space randomization (but not system call renumbering)
- Experience is very impressive!
  - Very few attacks on Windows Vista itself in past year
  - But applications remain very vunerable

25

PaX

- Linux kernel patch
- Goal: prevent execution of arbitrary code in an existing process's memory space
- Enable executable/non-executable memory pages
- Any section not marked as executable in ELF binary is non-executable by default
  - Stack, heap, anonymous memory regions
- Access control in mmap(), mprotect() prevents unsafe changes to protection state at runtime
- Randomize address space

slide 26

Non-Executable Pages in PaX

- In x86, pages cannot be directly marked as non-executable
- PaX marks each page as "non-present" or "supervisor level access"
  - This raises a page fault on every access
- Page fault handler determines if the page fault occurred on a data access or instruction fetch
  - Instruction fetch: log and terminate process
  - Data access: unprotect temporarily and continue

slide 27

mprotect() in PaX

- mprotect() is a Linux kernel routine for specifying desired protections for memory pages
- PaX modifies mprotect() to prevent the following attacks:
  - Creation of executable anonymous memory mappings
  - Creation of executable and writable file mappings
  - Making executable, read-only file mapping writable
    - Except when relocating the binary
  - Conversion of non-executable mapping to executable

slide 28

Access Control in PaX mprotect()

- In standard Linux kernel, each memory mapping is associated with permission bits
  - VM_WRITE, VM_EXEC, VM_MAYWRITE, VM_MAYEXEC
    - Stored in the vm_flags field of the vma kernel data structure
    - 16 possible write/execute states for each memory page
- PaX makes sure that the same page cannot be writable AND executable at the same time
  - Ensures that the page is in one of only 4 "good" states
    - VM_MAYWRITE, VM_MAYEXEC, VM_WRITE | VM_MAYWRITE, VM_EXEC | VM_MAYEXEC
  - Also need to ensure that attacker cannot make a region executable when mapping it using mmap()

slide 29

PaX ASLR

- User address space consists of three areas
- Base of each area shifted by a random "delta"
  - Executable: 16-bit random shift (on x86)
    - Program code, uninitialized data, initialized data
  - Mapped: 16-bit random shift
    - Heap, dynamic libraries, thread stacks, shared memory
  - Stack: 24-bit random shift
    - Main user stack
- Only 16 bits of randomness are used to determine the random shift

slide 30

PaX RANDUSTACK

- Responsible for randomizing userspace stack
- Userspace stack is created by the kernel upon each execve() system call
  - Allocates appropriate number of pages
  - Maps pages to process's virtual address space
    - Userspace stack is usually mapped at 0xBFFFFFFF, but PaX chooses a random base address
- PaX randomizes not only the address at which the stack is mapped, but also the range of allocated kernel memory

slide 31

PaX RANDKSTACK

- Linux assigns two pages of kernel memory for each process to be used during the execution of system calls, interrupts, and exceptions
- PaX randomizes each process's kernel stack pointer before returning from kernel to userspace
  - 5 bits of randomness
- Each system call is randomized differently
  - By contrast, user stack is randomized once when the user process is invoked for the first time

slide 32

PaX RANDMMAP

- When Linux allocates heap space, it starts at the base of the process's unmapped memory and finds the nearest chunk of unallocated space which is large enough
  - This is done in do_mmap() routine
- PaX modifies do_mmap() so that it adds a random delta_mmap to the base address before looking for new memory
  - 16 bits of randomness

slide 33

PaX RANDEXEC

- Randomizes location of ELF binaries in memory
  - Problem if the binary was created by a linker which assumed that it will be loaded at a fixed address and omitted relocation information
- PaX maps the binary to its normal location, but makes it non-executable; creates an executable mirror copy at a random location
  - Access to the normal location will result in a page fault; page handler determines whether it is safe to redirect to the randomized mirror
    - Looks for "signatures" of return-to-libc attacks and may result in false positives

slide 34

Base-Address Randomization

- Note that only base address is randomized
  - Layouts of stack and library table remain the same
  - Relative distances between memory objects are not changed by base address randomization
- To attack, it's enough to guess the base shift
- A 16-bit value can be guessed by brute force
  - Try 2^{15} (on average) overflows with different values for addr of known library function – how long does it take?
    - Shacham et al. attacked Apache with return-to-libc
    - usleep() is used
  - If address is wrong, target will simply crash

slide 35

Ideas for Better Randomization (1)

- 64-bit addresses
  - At least 40 bits available for randomization
    - Memory pages are usually between 4K and 4M in size
  - Brute-force attack on 40 bits is not feasible
- Does more frequent randomization help?
  - ASLR randomizes when a process is created
  - Alternative: re-randomize address space while brute-force attack is still in progress
    - E.g., re-randomize non-forking process after each crash (recall that unsuccessful guesses result in target's crashing)
  - This does not help much
    - See Shacham et al. paper for probability calculations

slide 36

Ideas for Better Randomization (2)

- Randomly re-order entry points of library functions
  - Finding address of one function is no longer enough to compute addresses of other functions
    - What if attacker finds address of system()?
- ... at compile-time
  - Access to source, thus no virtual memory constraints; can use more randomness
  - What are the disadvantages??
- ... or at run-time
  - How are library functions shared among processes?
  - How does normal code find library functions?

slide 37

Javascript: The next frontier

- We've seen that Javascript/AJAX creates a new kind of distributed operating system platform
  - One that has vulnerabilities too
  - And easy to attack: almost everything has web browsers, web email, web services mechanisms
- If the O/S becomes more robust, attackers will just focus on applications or on Javascript "vector"

38

Consolidation conclusions

- Probably won't save money on client machines
- Could benefit if you run a lot of servers
- Improved management and configuration will make systems more robust, and synthetic diversity helps too
- But predictability of the *application* base could increase risk of rapid virus outbreaks
  - And today, the web is the ultimate application

39

Readings

- **How to Own the Internet in Your Spare Time.** Stuart Staniford, Vern Paxson, Nicholas Weaver. Security 2002, 149-167
- **On the Effectiveness of Address-Space Randomization.** Hovav Shacham et al. CCS 2004
- **The Monoculture Risk Put into Context.** Fred B. Schneider and Ken Birman. IEEE Security & Privacy. Volume 7, Number 1. Pages 14-17. Jan/Feb 2009.

40