

Announcements

- Project 3 Final Design Docs will be returned next week
- CONGRATS on an excellent job on proj3!
 - o Grades were superb.

Today's Menu:

- Project 4: Things to keep in mind
- NFS and AFS
- Computer Worms and Internet Security

Project 4: Things to Keep in Mind

- This is another testing-critical project
 - o You will be asked to write a program that deals with random loss!
 - o As we already know. . .debugging with randomness is HARD
 - How will you test such that you can make statements of correctness about your various modules?
 - Your answer should be in your design docs.
- DO NOT WORRY ABOUT DESIGNING A PROTOCOL.
 - o We give you a state diagram and thorough description of a protocol for you to use!
 - You do NOT have to design one.
- Use only one thread for send and receive.
 - o Do NOT spawn a thread for every send and every receive operation.
 - o Basically, this thread is going to be taking care of buffers.
- Each connection is uniquely determined by a 4-tuple
 - o (Source Address, Source Port, Destination Address, Destination Port)
 - o Apply this fact to the following question: how can a server's sshd server daemon listen only on port 22 for incoming connections?

NFS and AFS

- One of the most common uses of distribution is to provide distributed file access through a distributed file system
 - o Basic idea: support sharing of files and sharing of devices (disks) network wide.
- Basic Issues
 - o File naming
 - how are files named?

- are those names location transparent (is the file location visible to the user)?
 - are those names location independent?
 - do the names change if the file moves?
 - do the names change if the user moves?
 - Caching
 - caching exists for performance reasons
 - where are file blocks cached?
 - On the file server?
 - On the client machine?
 - what happens when a cached block/file is modified
 - how does a node know when its cached blocks are out of date?
 - Replication
 - replication can exist for performance of availability
 - can there be multiple copies of a file in the network?
 - if multiple copies, how are updates handled?
 - what if there's a network partition and clients work on separate copies?
 - at what level is replication visible?
 - Performance
 - what is the cost of remote operation?
 - what is the cost of file sharing?
 - how does the system scale as the number of clients grows?
 - what are the performance limitations: network, CPU, disks, protocols, data copying?
- How to do this?
 - The obvious solution is to talk to server for all operations.
 - Slow
 - Coherency issues among multiple files.
 - Not scalable.
- Network File System
 - Every client and server maintains local caches of files stored centrally.
 - NFS runs over LANS (even over WANs -- slowly).
 - Allows a remote directory to be "mounted" (spliced) onto a local directory, giving access to that remote directory and all its descendants as if they were part of the local hierarchy.
- NFS defines a set of RPC operations for remote file access:
 - searching a directory
 - reading directory entries
 - manipulating links and directories
 - reading/writing files
- Every node may be both a client and server.
 - NFS defines new layers in the Unix file system

- Connections are ‘stateless’ in that the server does not maintain any information about the client
 - Except performance hints, such as the contents of the cache
- On open, the client asks the server whether its cached blocks are up to date.
 - Once a file is open, different clients can write it and get inconsistent data.
 - Modified data is flushed back to the server every 30 seconds.
 - This also creates a consistency problem!
 - If multiple people are working on the same file simultaneously, other users can open the file and get completely different things!
 - No ordering constraint on updates.
- On close, modified blocks are sent back immediately to the server.
 - Client ‘close’ doesn’t return until this operation is complete.
- Andrew File System
 - Consists of workstation clients and dedicated file server machines.
 - Workstations have local disks, used to cache files being used locally.
- Andrew has a single name space -- your files have the same names everywhere in the world.
 - Andrew is good for distant operation because of its local disk caching: after a slow startup, most accesses are to local disk.
- Need for scaling led to reduction of client-server message traffic.
 - Once a file is cached, all operations are performed locally.
 - On close, if the file is modified, it is replaced on the server.
 - The client assumes that its cache is up to date, unless it receives a *callback* message from the server saying otherwise.
 - On file open, if the client has received a callback on the file, it must fetch a new copy
 - Otherwise it uses its locally-cached copy.

QUESTION: What are the major differences between AFS and NFS?

ANSWER:

- NFS uses polling to update the file on the server. AFS uses a callback model.
 - Callback model reduces network usage.
- NFS stores shared blocks in local memory. AFS uses disk. (minor)
- In both of these systems:
 - Central server is the bottleneck
 - All data written through to the server eventually. . .
 - All cache misses must go to server
 - This makes the server a point of failure

Computer Worms and Internet Security

- What are Computer Worms?
 - They are self replicating network programs that exploit vulnerabilities to infect remote machines.

- Victim machines continue to propagate the infection
- There are three main stages in the lifecycle of a worm
 - Detect new targets
 - Attempt to infect new targets
 - Activate the code on the victim machine
- With most modern worms, this is done without human intervention!
 - i.e. faster than human reaction—WICKED FAST. More on this later.
 - Example: Code Red required ~13 hours to spread worldwide
 - How often do you update your virus protection?
- Worms can have highly malicious payloads
 - Distributed Denial of Service Attacks
 - Internet scale espionage
 - Data corruption, manipulation
 - BIOS reflashing
- Why do attackers like worms?
 - Worms are useful attacker tools
 - Can attack an entire vulnerable population at once
 - Can be harder to trace than conventional attacks
 - Worms are easy to write
 - Propagation routines can be generic, enabling code reuse (Slapper)
 - Drop in an exploit and release
 - Payload is independent of propagation
- How is an exploitable vulnerability found?
 - It is disturbingly easy to find vulnerabilities to exploit.
 - Bug lists, newsgroups
 - Also published bugs!
 - eEye security finds and publishes several bugs to try and get fixes done before they are exploited.
 - Still, because one operating systems dominates the marketplace. . .
 - And because people rarely patch their machines promptly . . .
- How does a worm detect new targets?
 - This is the meat of how a computer worm works! Several methods have been seen.
 - Random Target Selection (scanning) – Generic method
 - Repeat Forever:
 - Pick a "random" address, if vulnerable, infect it
 - Simple to implement
 - Most code is generic
 - Speed (K) depends on:
 - Rate of scanning
 - Number of vulnerable machines
 - Size of address space
 - Some optimizations that one can do to make this more effective:
 - Local subnet scanning: Preferentially scan the local network
 - Code Red II, Nimda did this

- Exploit a single breach to attack the local Intranet
- Permutation Scanning
 - Guarantees distributed scanning without explicit cooperation
- How fast are worms who use this?
 - Code Red: scanner is latency-limited
 - In many threads: send SYN to random address, wait for response or timeout
 - Code Red: ~6 scans/second, population doubles about every 40 minutes
 - Sapphire: every copy sent infectious packets at maximum rate
 - 1 Mb upload bandwidth = 280 scans/second
 - 100 Mb upload bandwidth = 28,000 scans/second
 - Any reasonably small TCP worm can spread like Sapphire
 - Needs to construct SYNs at line rate, receive ACKs in a separate thread
- Internal target list (topological) -- Fast, but application specific
 - Look for local information to find new targets
 - URLs on disk and in caches
 - Mail addresses
 - .ssh/known_hosts
 - Ubiquitous in mail worms
 - More recent mail worms are more aggressive at finding new addresses
 - Basis of the Morris worm
 - Address space was too sparse for scanning to work
- Passive (contagion) -- Slow and stealthy, propagate in response to external events
 - Wait for information about other targets
 - CRclean, an anti-CodeRed II worm
 - Wait for Code Red, respond with counterattack
 - Nimda: Infect vulnerable IE versions with Trojan web-page
 - Speed is highly variable
 - Depends on normal communication traffic
 - Very high stealth
 - Have to detect the act of infection, not target selection
- Attacker can mix and match strategies

- Here are some of the better known worms:

Worm	Year	Strategy	Victims	Other Notes
Morris	1988	Topological	6000	First major autonomous worm. Attacked multiple vulnerabilities.
Code Red	2001	Scanning	~300,000	First recent "fast" worm
Nimda	2001	Scanning Others	~200,000	Local subnet scanning. Effective mix of techniques
Slammer	2003	Scanning	>75,000	Spread worldwide in 10 minutes
MS Blaster	2003	Scanning	>200,000	Fast spread rate. Easy to get re-infected. DDOS Payload.

- Code Red and Code Red II
 - o Original Code Red first appeared July 2001
 - Named for the favorite beverage of the security team that first reverse-engineered it! (no, I'm not kidding)
 - o Targets machines running Microsofts IIS, using a known buffer-overflow vulnerability.
 - Vulnerability was published the month before the virus came out.
 - o How propagation worked
 - The worm scanned address spaces.
 - Collisions between instances of the worm, hosts scanned repeatedly.
 - Network traffic brought to a crawl.
 - Code Red I had a bug in its random number generator in that it used a fixed seed, and thus had a limited pool of people it could infect.
 - Every infected computer goes go through the same list of "random" IP addresses
 - Code Red II fixed this.
 - o Once a machine is infected:
 - First, the initial worm environment is set up on the infected system and 100 threads of the worm are created.
 - The first 99 threads spread the worm (infect other web servers).
 - The 100th thread checks to see if it is running on an English (US) Windows NT/2000 system.
 - If the infected system an English (US) system, the infected system's website is defaced.
 - If the system is not an English (US) Windows NT/2000 system, the 100th worm thread is also used to infect other systems.
 - Each worm thread checks for c:\notworm.
 - If the file c:\notworm is found, the worm goes dormant.

- Each worm thread checks the infected computer's system time.
 - If the date is past the 20th of the month (GMT), thread will DDOS www.whitehouse.gov (410 megabytes of data every 4 and a half hours per instance of the worm).
 - If the date is between the 1st and the 19th of the month, thread continues to scan and infect.
- MS Blaster
 - Extremely virulent worm that kicked the crap out of our department.
 - First appeared in early August, 2003
 - Example of a multi-component worm!
 - Infection packets do not contain the payload.
 - The first component is a publicly available Win32 RPC DCOM exploit that binds a system level shell to port 4444.
 - Once the target is compromised, the worm transmits the msblast.exe executable (the main body of the worm) via TFTP to infect the host.
 - The payload used in the public DCOM exploit, as well as the TFTP functionality, are both encapsulated within msblast.exe.
 - When a host is infected with the Blaster worm through the execution of msblast.exe, a registry value of "msblast.exe" is created.
 - The executable specified by the newly created registry value will be run automatically each time the machine is booted.
 - Once registry value is in place, the Blaster worm next creates a mutex named "BILLY."
 - This allows Blaster to recognize that a target has already been infected.
 - An infected machine won't be interfered with by multiple infection attempts.
 - If an internet connection is available the infected host will begin SYN flooding windowsupdate.com on port 80 using spoofed source addresses.
 - The worm will continue infecting hosts while carrying out a SYN flood attack against windowsupdate.com.
- Obviously these worms are dangerous.
 - Damages in cleanup, lost business and support fees immense.
 - 2001: Nimda cost the tech sector \$635 million.
 - 2001: Code Red? \$2.62 billion
 - 2000: Love Bug accumulated a staggering total of \$8.75 billion in damages.
 - National security issue!
 - These are not terribly hard to write.
 - Bringing down network communications in a time of war would be a bad thing.
- How do we protect against them?
 - Current knee-jerk reaction model not working.
 - As Slammer and Blaster showed, propagation is getting faster and faster.

- No time to react and patch!
- Will need other ways to detect and react to threats before they propagate!
 - This is a hot research area.

References:

eEye Digital Security. <http://www.eeye.com/html/Research/Advisories/index.html>

Nicholas Weaver, Vern Paxson, Stuart Staniford, and Robert Cunningham, "A Taxonomy of Computer Worms", *First Workshop on Rapid Malcode (WORM)*, 2003.

David Moore, Vern Paxson, Stefan Savage, Colleen Shannon, Stuart Staniford, and Nicholas Weaver, "Inside the Slammer Worm", *IEEE Magazine of Security and Privacy*, August 2003.

Nicholas Weaver, Yury Markovskiy, Yatish Patel, and John Wawrzynek, "Post Placement C-slow Retiming for the Xilinx Virtex FPGA", *11th ACM Symposium of Field Programmable Gate Arrays (FPGA)*, 2003.

Steve Martin, Anil Sewani, Blaine Nelson. *Adaptive Prevention Environments*.
http://www.eecs.berkeley.edu/~steve0/papers/APE_martin_nelson_sewani_262A_03.pdf