Internet

• Decentralized approach -> no control/responsibility
• Open architecture -> anonymous endpoints/spoofing
• End-to-end principle -> security built at the edges
• Automation, scripting, action-at-a-distance
Partial Answer: Firewalls

- Devices examining traffic -> access control decisions
  - Divide the world between trusted and not
  - Only authorized traffic is allowed to pass
Firewalls

• "Crunchy shell, soft chewy center"

• Originally devised to contain bad network software problems

• Can operate at various levels in the stack
  ▶ Link, network, application
  ▶ Packet filtering vs. circuit switching
  ▶ Transparent vs. proxies
  ▶ Stateful vs. stateless
Operation At Different Layers

Application/proxy level firewalls
- FTP
- HTTP
- TELNET
- NFS

Network/packet filtering firewalls
- TCP
- UDP
- IP
- ICMP

Link-layer firewalls
- Ethernet
- FDDI
- PPP
Problems With Firewalls

- All attachments to the public network must be protected
  - High degree of connectivity
  - Consistency, administration becomes a problem
  - Unidentified network attachments can bypass security
  - Topologies not as clear-cut as in the past
    - Telecommuting, extranets
- Complicated protocols
- End-to-end encryption
- Performance bottleneck
- Very coarse-grain protection
  - Majority of attacks are from insiders
Distributed Firewalls

- Firewalls are convenient for specification and enforcement of policy
- Keep specification centralized, distribute enforcement
  - Each node on the network becomes its own firewall
  - Encryption becomes an asset
  - Protocol/application information available
  - Distributed performance
  - Anyone can be treated as an outsider
- Firewall policy can be pushed or pulled
- Commercial products available
  - Even at the network card level
Current Problems

- Network denial of service (DoS)
- Remote software exploits
- Worms
Denial of Service

• Limited resources
  ▶ Bandwidth, memory, CPU cycles
  ▶ More abstract: service (e.g., web server)

• Saturate with requests for resource
  ▶ Deny service to other users
  ▶ Degrade performance, exhaust resources

• Real-life examples
  ▶ Yell near someone, pull the plug on a machine, etc.

• Resource accounting problem
Denial of Service (cont.)

- Easier to launch than other (e.g., crypto) attacks
  - Often, this is sufficient

- DoS in operating systems
  - CPU: "while (1) ;"
  - Memory: "while (1) malloc(65537);"
  - OS tables: "while (1) fork();"

- Resource allocation per-user/process
  - getrusage()
  - Sometimes it works
Network DoS

• Over a network
  ▶ No need to be a legitimate user

• Action at a distance
  ▶ Minimize risk

• Larger volume
  ▶ Distributed DoS (DDoS)

• Authentication/encryption do not help by itself
  ▶ Firewall becomes easy DoS target
Types of DoS

• Link congestion (forward or reverse)
  ▶ Send many large packets or ask for many larger web pages
  ▶ Saturate target’s access link

• Router processing capacity
  ▶ Send many small packets
  ▶ High processing overhead on router
  ▶ Also acts as link congestion

• End-host (server) processing capacity
  ▶ Ask for "expensive" operations (show complete database)
Currently...

Source

ISP

Core

ISP

ISP

ISP

Target

Congestion points

Attacker
Simple Protection

- Reserving bandwidth (RSVP/Diffserv)
- Authentication + process/task scheduling by the OS
- Load balancing (multiple links/servers)

- Do not help with congestion attacks
- High-speed Internet core
  - Routers cannot spend cycles verifying packets
  - Routers close to target can, but links are smaller
Distributed DoS (DDoS)

- Coordinate attack on target from various sources
- Higher aggregate attack bandwidth
- Subvert hosts, use them as "zombies"
  - Hard-coded attack, time-based, or control channel
- Worms, email viruses make it easy to launch DDoS
  - Fundamentally, it’s bad security
- IP address spoofing may be used
  - Ingress filtering would help
  - Cannot be and is not universally deployed
Defenses Against DDoS

- Data replication (Akamai, CDNs)
  - Only works with static content
- Black-holing by ISPs
- Attack prevention
- Trace the source of attacks
- Secure Overlay Services
Attack Prevention

• Better security
• Ingress filtering
  ▶ Only helps against spoofing
  ▶ A worm that takes over 1M hosts need not spoof
• Apply IDS techniques
  ▶ At the edge routers or the core
• Build models of good traffic
  ▶ Treat preferentially
• Build models of bad traffic
  ▶ Filter or limit such traffic
• Susceptible to probing attacks (guess characteristics of good traffic and spoof)
Pushback

- Detect attack
- Determine characteristics
  - Predicates on packet fields that can be used to filter
- Contact upstream routers and pass them the predicates
  - Continue as far as possible
  - Potentially all the way to the sources’ edge routers
- Automated mechanism
Pushback (cont.)

• Potentially subject to "gaming"
• Can be used to deny service to innocent hosts, if filters are not pushed all the way to edge routers
  ▶ Spoof from real DoS target
  ▶ Network will filter/rate-limit traffic from that host
• Who is allowed to push filters to an ISP’s routers?
  ▶ Business weapon...
Attack detection

- Determine who the real sources of an attack are
- Contact administrator or use pushback
  - Generally, of limited use
  - Attack clouds of 10,000 hosts or more
Algebraic approaches to detection

- In-band notification of target
- First approach: probabilistically add router identity in packet
  - Use "opaque" fields, e.g., IP ID field
- Second approach: encode a digital watermark in packet
  - Again, use "opaque" fields on packet
- When target receives enough attack packets, router path can be determined
ICMP Traceback

• Out-of-band notification of target
• Routers probabilistically send ICMP message to destination of sampled packet
• Include the packet header of sampled packet
• In a DDoS, target will eventually receive ICMPs from all routers in the path of the DDoS
Polling-based Traceback

- Source Path Isolation Engine (SPIE)
  - Routers "remember" whether packet was recently seen
  - Targets query upstream routers to determine who has seen attack packet
  - Apply recursively
- Use Bloom filters to probabilistically remember if packet was seen
- Considerable hardware support required
Secure Overlay Services (SOS)

• Proactive mechanism using overlay services and distributed firewalls
  ▶ Build filtering perimeter around target
  ▶ Permit traffic only from authorized nodes (Secret Servlets)
    • Packet filtering faster than crypto
    • Identity of SS variable over time
  ▶ Authorized users authenticate to any node
  ▶ Traffic then relayed to Secret Servlet node

• Assumptions
  ▶ Attackers cannot saturate Internet core
  ▶ Attackers cannot eavesdrop in arbitrary links
Current Problems

- Network denial of service (DoS)
- Remote software exploits
- Worms
Remote Software Exploits

• Protocols and algorithms may be perfect
  ▶ Implementations is another story!

• **Majority of vulnerabilities are result of bad code**
  ▶ Buffer overflows
  ▶ Race conditions
  ▶ Insufficient/wrong argument validation

• Backdoors, malicious code, viruses
Applicability

• Applications
  ▶ Usually privileged ones
• Extensible (operating) systems
• Mobile agents
Buffer Overflows

• Overwrite return pointer in caller’s stack frame
  ▹ Arguments on the stack
  ▹ Missing bounds checking
• BSS and heap overflows
  ▹ Virtual functions, object methods
Race Conditions

• Time Of Check To Time Of Use (TOCTTOU) bugs

• Example of updating /etc/passwd
  ▶ Pick "random" filename
  ▶ Check that it does not exist in /tmp
    • If it does, loop
  ▶ If not, open file
  ▶ Copy contents of /etc/passwd
  ▶ Add new entry
  ▶ Copy temp file to /etc/passwd

• Other example: changing symbolic link pointer between check and use
Bad Argument Validation

- Example: sendmail debug flag
  - Given as number in command line
  - Used as index in table to set appropriate debug flag
  - But: no bounds checking
  - And: sendmail running "setuid"

- Result: able to add code (and execute it)

- Example: sprintf format string

- Solutions?
Better APIs

• Engineering solution
  ▶ strcpy/strcat -> strncpy/strncat
  ▶ sprintf -> snprintf
  ▶ tmpnam -> mkstemp
  ▶ ...

• Not always possible (thanks to standards)
  ▶ Sometimes, new API confusing
    ▶ strlcpy/strlcat
Safe Languages

• Use a language where "bad thoughts" are impossible
• Examples: Java, ML/Caml, Erlang, etc.
  ▶ Type safety
  ▶ Memory management
• VM may still be unsafe (Java bytecode, JIT, ...)


Proof-Carrying Code

• Input: piece of code, safety policy
• Output: safety proof
• Proof generation is computationally expensive
  ▶ Verification simpler and less expensive
• Compiler need not be trusted
  ▶ Only the verifier
Proof-Carrying Code (2)

• Burden is on the code producer
  ▶ Prove once, use everywhere (with same policy)
• Reliance only on the verifier (which is small)
• Tamperproof programs: modifying a program will
  ▶ Invalidate the proof
  ▶ Make the proof non-applicable to the program
  ▶ Proof and program still valid -> good
• Simple programs (packet filters) / policies
  ▶ Promising
Software Fault Isolation (SFI)

- Software encapsulation of code
- Partition code into data and code segments
  - Prevent self-modifying code
- Code is inserted before each load, store, and jump instruction
  - Verify that the target address is safe
- Done at compiler, link, or run time
  - Increases program size, slow down
- "Tricky" for CISC architectures
Code Signing

- Code producer (or trusted compiler) digitally signs code
- User checks signature, verifies code comes from "trusted" identity

- Generally insufficient
  - Implies "binary" trust model
  - Malevolent/subverted "trusted" party can cause damage
  - Lack of a PKI -> non-scalable approach

- Reasonable as first line of defense
Unix chroot() 

• In Unix, (almost) everything is part of the filesystem 
• Limit what code/process can do by restricting their view of the filesystem 
• Typically, daemon processes run in their own mini-filesystem 
• Possible to escape, or cause damage even from inside a chroot’ed environment 
• FreeBSD jail() 
  ▶ Different virtual machine based on IP address
Capabilities

• Use fine-grained access control for all resources
• Allow users to specify exactly what resources processes have access to
  ▶ Increased administrative complexity
  ▶ Must modify existing applications
System Call Monitoring

- Sandbox untrusted applications by monitoring system calls
  - Enforce particular policy
- Policy may be uploaded to kernel
- Similar to virus checker
- Have to hand-tune policy for individual applications
  - Fine for widely-used daemons, tricky for downloaded code (e.g., plug-ins)
- Java security manager approach fundamentally similar
OpenBSD systrace

- System call interposition
- User-level daemon listening to socket to the kernel
  - Receives information about monitored process
  - Evaluates request based on policy
  - Responds to kernel
- Allows manual intervention through GUI
- Policy discovery
- Performance, complexity
Emulators

• Create virtual machine, run individual programs (or instances of the operating system) in it
  ▶ Increased reliability

• Can take advantage of hardware capabilities for improved performance

• No explicit policy to be determined
  ▶ Similar to chroot/jail

• Good for daemons/services, less so for really practical use
  ▶ Applications tend to become "little OSes" themselves
  ▶ Integrated application suites
Compiler Tricks

• StackGuard: inject runtime checks for buffer overflows
• A lot of other related work
  ‣ StackGhost, ProPolice, FormatGuard, etc.
• Not fool-proof
  ‣ Heap-based overflows, SQL-injection
• Performance penalty (sometimes significant)
Code Randomization Techniques

• Apply Kerckhoff’s principle on programs
  ▶ Key-driven randomization of certain aspects of binary
  ▶ Reveal key to OS
  ▶ Attacker must mount exhaustive-search attack

• Randomize location/size of stack/activation records
• Randomize location of linked libraries
• Randomize instruction set!
Current Problems

- Network denial of service (DoS)
- Remote software exploits
- Worms
Worms

• Self-propagating malicious code

• Infection vector and payload
  ▶ So far limited use of malicious payload

• Common vectors
  ▶ Remotely-exploitable software faults
    • Buffer overflows
  ▶ Too-smart e-mail agents/web browsers
  ▶ Unsuspecting/naive humans
    • "Click to see photo of Pamela Anderson..."

• Propagation speeds exceed human reaction
  ▶ "Slammer" (SQL) worm infected all targets in 8 minutes
  ▶ Faster worms possible
    • "Warhol" worms, hit-list scanning, ...
Protection Mechanisms

- Sandboxing
  - Only limits damage to remainder of system
- Connection throttling
- Content filtering
  - Slow, error-prone, breaks in presence of encryption
- Anti-worms
  - Dangerous, not dependable
- Artificial diversity
  - See code randomization
- Automated software patching
- Open problem
Proceedings of security conferences

- IEEE Security & Privacy
  - [http://www.ieee-security.org/TC/SP-Index.html](http://www.ieee-security.org/TC/SP-Index.html)
- USENIX Security
  - [http://www.usenix.org](http://www.usenix.org)
- ISOC NDSS
  - [http://www.isoc.org/isoc/conferences/ndss](http://www.isoc.org/isoc/conferences/ndss)
- ACM CCS
  - [http://www.acm.org/sigs/sigsac/ccs.html](http://www.acm.org/sigs/sigsac/ccs.html)