Lecture 22: Security I
What it’s all about

• Risks: Loss of ...  
  • Confidentiality/Privacy, Integrity, Availability/Access  
  • Risk Analysis: cost/loss * loss freq. vs. cost to protect  
  • Engineering trade-offs, not either-or decisions  
  • often, Security == 1 / (functionality * convenience)

• Vulnerabilities  
  • examples abound, many reasons behind

• Countermeasures  
  • carefulness, cryptography, firewalls, detection, recovery
Some key security goals

• Confidentiality
  • keep information content away from the unauthorized

• Integrity
  • prevent undetected, unauthorized modification of data

• Availability
  • ensure that resources and services are available when needed

• Authentication
  • prove the identity of entities or source of information

• Non-repudiation
  • prevent denial of previous commitments

• Privacy and Anonymity
  • prevent Big Brother from invading your space
Some key security problems

• 1. Misplaced trust
• 2. Buggy implementations
• 3. Poor configuration choices
• ...
• 12. Unsafe design assumptions
• ...
• 997. Cryptanalysis
Terminology: Threats

- Threat: A potential vector (means, mechanism) for a system’s security to be compromised
  - An attack exercises a threat
  - A successful attacks leads to a security compromise
- Examples of threats:
  - Network traffic arriving from the internet
  - Self-administered systems connected to a corporate (or university) LAN
Terminology: Vulnerabilities

• A vulnerability is a flaw in a system that has a security implication
• Examples:
  • Unchecked string copy allows buffer overflow
  • Administrator forgets to disable debug mode on a program during configuration, leaving unsafe but convenient features in deployed service
  • Naïve home user buys wireless router, but does not alter default password on router
• Compromises occur when attacker matches threats with vulnerabilities
Scary aspects of “bad guys”

• Patience and time
  • Historically successful crackers have been willing to spend endless hours trying to get into systems

• Automated Tools
  • Crackers don’t even have to know anything anymore
    • Copious “cookbooks” and packaged kits
  • One clever person finds a hole, everyone runs her tools

• New profit motive
  • Rent-a-bot-net brokers
...and if that weren’t bad enough

- attackers only need one weakness
  - no need to break thru strongest wall
  - they’ll try lots and exploit the weakest
Example: The Morris Worm

- Released on November 2, 1988
- Written by Robert T. Morris
- Invaded around 6,000 computers within hours (10% of the Internet at the time)
- Disabled many systems and services
- Morris had a friend post instructions for disabling the worm - but it was too late
- Damage estimates between $10,000 and $97 million (shows how hard it is to estimate)
- Details in June 1989 Comm. of the ACM
  - "Crisis and Aftermath", Eugene H. Spafford
How the worm worked

- Copied itself to remote systems via 3 holes
- Exploit hole in finger daemon that caused buffer overflow to create remote shell
  - `gets()` used to read input
- Exploit hole in Unix sendmail daemon
  - `listen()`'s on TCP port, `accept()`'s connections from mailers
  - Exchanges messages about mail envelope and content
  - when running in debug mode, worm could give it commands to execute
    - `sendmail` ran the malicious code
- Password cracking with a dictionary of 432 words
  - accounts tested against words in a random order
“People pick bad passwords, and either forget, write down, or resent good ones.”

Steven M. Bellovin
Effect of worm

• Formation of CERT
• $10,000 fine, 3 year probation, and 400 hours of community service for Morris
• Heightened awareness of computer system vulnerabilities
• Something for security professionals to quote
  • not so much a problem now 😞
Example: Melissa virus (1999)

• On all news shows, newspapers, etc.
• Code is in the form of MS Word macro
  • 107 lines of visual basic
• Most impactful mobile code attack since Morris worm
  • millions of $$$ of damage
• Virus didn’t “do anything bad” except copy self
  • plus clog and shut down internet mail servers
• If date == time, prints

  Twenty-two points, plus triple-word-score, plus fifty points for using all my letters. Game’s over. I'm outta here.
Melissa (cont.)

• First public appearance on alt.sex
• Many copycat viruses immediately appeared
• How Melissa works:
  • word macro virus implanted in MSWord file
  • word file contained a list of pornographic sites
  • MsWord file mailed with subject:
    • Important Message From xxx
  • Body:
    • Here is that document you asked for... don't show anyone else ;-)  
    • and an attachment: list1.doc
How Melissa works (cont.)

• When user opens document
  • warning about document containing macro
  • If user clicks okay, word launches, with the virus
• The virus then:
  • disables future checking for macro viruses (no prompt)
  • check to see if already infected (keyword Kwyjibo)
  • if not infected, then look in outlook address book
  • mail the infected document to the first 50 names
  • infect word template for new documents
    • this is a classic macro virus
    • all future word documents will be infected
Why Melissa worked

• many, many people using same mailer (outlook)
• many, many people use MsWord in Windows
• many, many people click okay to macro warning
• no separation between applications on Microsoft platforms
• virus protection software only works against known viruses

• lack of domain separation allows access to Outlook data

It could have been A LOT worse!
Example: W32/Sircam (2001)

- Malicious code that propagates via e-mail
  - “Hi! How are you?<RET>Random<RET>See you later. Thanks”
- Click on the attachment, and virus springs into action
  - Send itself to folks in address book (*.wab)
  - Do various nasty things
    - Search for and e-mail out sensitive files (breach confidentiality)
    - Fill up C: drive (reduce availability)
    - Delete files from C: drive (breach integrity)
- This one isn’t as “harmless” as Melissa…
Recent Examples

• 2006: Nyxem – mass email worm
  • Disables security related software
  • Attempts to destroy certain types of files
• 2005: MySpace virus Samy (fastest spreading to date)
• 2004: MyDoom – mass email worm
• Many, many others
“Network Security”

- Attacks that use or go after the network
- Many examples we’ve seen already
  - holes in apps/services can be targeted from a distance
- Network-specific attacks
  - eavesdropping (aka sniffing)
  - impersonation (aka spoofing)
  - protocol disruption/exploitation
  - denial of service
  - Distributed denial of service
Buggy Code

- 85% of CERT Advisories describe problems that cannot be fixed with cryptography.
- Most of these are bugs in code.
- But writing correct code is the oldest -- and probably the most difficult -- problem in computer science. We're not going to solve it any time soon -- and possibly not ever.
Preventing Bugs

• Structuring code properly can help
  • isolate the security-critical sections
  • use better languages and libraries (and programmers)

• Reducing code complexity
  • when was the last time a vendor deleted features when shipping a new release?
  • when did you ever see an ad bragging that some Web browser doesn’t have Javascript?
  • people don’t understand how to use what’s already there - so vendors add even more complexity to help people find the knobs and buttons...
Buffer Overruns

- 30-40% of newly-reported holes are due to these:
  - 9 of 13 CERT advisories in 1998
- C uses character arrays for strings (and for arrays):
  - C compilers do no bounds checking on arrays
  - language design makes such automated checking hard
  - programmers often fail to check all lengths and indices
  - common libc functions (e.g., gets, sprintf) also flawed
  - “Safe” versions also flawed
- Too many programmers say “this array is big enough” -- and it often is, for normal purposes...
Stack Overruns

• A particularly nasty form of buffer overrun
  • normal buffer overruns allow corruption of data
  • stack overruns can give control of execution
• Happens when buffers allocated on stack
  • e.g., local procedure variables
• Since return address also on stack...
  • attacker can subvert return address
  • ... and insert code to be executed
  • ... and point the return address at the new code
• Example: fingerd bug from Morris Worm
Format String Bugs

- C library formatted I/O strings can allow almost arbitrary inspection and modification of program if misused
  - First came to light ~2000
  - E.g. “printf(inputstr)” where inputstr is supplied by user
    - inputstr can contain format characters (like %x)
    - printf assumes extra arguments have been pushed on stack, dumps stack contents to output
    - %n format allows write to a memory location
Incomplete input checks

• Before using an input value, check it
  • such a simple rule, yet it is so often not done
  • Checking can be harder than it seems

• Examples:
  • Format string vulnerabilities
  • MySpace Samy virus
  • passing NULL, -1, 0, 0xdeadbeaf, etc...
  • Ballista results: 15-35% of bad POSIX parameters cause robustness failures

• Why?
  • Laziness, optimization, forgetsies, reorgs/changes, assumptions, mindsets
From bug to vulnerability

• So you have a buggy user-level application
  • Why is this so bad?
• In general, compromising a process allows attacker to obtain privileges of that process for arbitrary activities
  • Bad for you, but not necessarily bad for the system
  • Compromising a process with root privilege (on Unix systems) provides a lot of power
    • Read/write any file
    • Read/write kernel memory though /dev/kmem
    • Attach to and trace any running process
    • Install kernel modules / change system configuration
Insufficient Domain Separation

• Authorization domains should be clearly separated
  • otherwise, less-privileged code can get more-privileged code to do bad things
• Unfortunately, this is often not the case
• Examples:
  • environment inheritance by setuid programs in UNIX
    • e.g., max file length or number of files open
Security policies are critical

- Most organizations have a stated policy about control of private information and access to resources
- These policies can help guide protocol implementation
  - and can help with political and “clueless user” problems
- If you can’t say what’s important, how am I supposed to protect it?
  - ... and why should I bother trying?
Useful “principles” for security

• Principle of Easiest Penetration
  • passwords: crack, sniffing, trojan horses, social engineering

• Principle of Adequate Protection
  • See “risk analysis” and “cost/benefit”...

• Principle of Effectiveness
  • Countermeasures must be used to be useful!!
  • Remember: users are impatient, lazy ...
Firewalls

- Perimeter defenses for nets with unsafe hosts
  - we seem to be unable to harden hosts
  - firewalls are an admission that hosts may not be secure
  - firewalls provide a barrier between us and them
- Single point of control and expertise
  - access limitation and auditing
  - limits communication to/from the outside world
  - only leave a very few machines exposed to direct attack
    - ... and minimize their functionality
- Sounds great - how come we’re not done?
OS Security Mechanisms

- Access Controls
- FreeBSD Jails
- SELinux
Access Control

- **Common Assumption:**
  - System knows identity of user (authentication)
  - Access requests pass through some gatekeeper (authorization)

- **Implemented using Access Control Matrix**
  - Access control list
  - Capability

- **Two main types**
  - Discretionary Access Control (DAC)
    - User sets access rights for objects they own
  - Mandatory Access Control (MAC)
    - System sets rights that users can’t override
FreeBSD Jails

- **Goal**: isolation of processes to contain possible damage without lots of extra security management complexity
- **Built on chroot concept**
  - Give process (and all its children) separate view of file system tree (chdir /tmp/limited_fs; chroot /tmp/limited_fs)
  - Originally introduced for development
- **Added new “jail” command**
  - Each jail has own superuser
  - Privileges of superuser restricted to only affect things inside jail
  - Process in jail isolated from ones outside jail
SELinux

- Security Enhanced Linux
  - Version of Linux created by the NSA and Secure Computing Corporation (SCC)
- Incorporates University of Utah Flask security model
- Supports mandatory access control to all objects
- Supports various policy configurations
  - Role-based access control: rights are assigned to roles which users can take on
  - Multi-level access control
  - Type Enforcement: Each subject and object has a type, policies are represented by relationships between each type
Ethics and Computer Security

• When technology and legality fail, society must fall back on ethics of right and wrong...
• Ethics are a fuzzy thing
  • everyone must arrive at their own choices
• But, I say again,
  • you should not abuse your extensive knowledge of system design and security weaknesses to break into other people’s systems or commit other computer crimes!