

CSE331: Introduction to Networks and Security

Lecture 1
Fall 2006



Basic Course Information

- Steve Zdancewic — lecturer
 - Web: <http://www.cis.upenn.edu/~stevez>
 - E-mail: stevez@cis.upenn.edu
 - Office hours: Tues. 9:30-10:30 Levine 511
- Savi Basavaraj — TA
 - E-mail: savi@seas.upenn.edu
 - Office hours: TBA
- Course Web Pages:
<http://www.cis.upenn.edu/~cse331>
- Course news group:
upenn.cis.cse331

Prerequisites

- Would like to learn network and security fundamentals
- Programming experience
 - Java: CSE 121/115
 - C / Assembly: (not mandatory) CSE 240
- Have not taken TCOM 500
 - take CIS 551 instead.

Recommended Reading

- *No required text: instead, articles (mostly from the web) will be provided.*
- Supplementary reading (available at library):
 - **Security in Computing 3rd edition**
Charles P. Pfleeger
 - **Computer Networks: A Systems Approach**
Larry L. Peterson & Bruce S. Davie
 - **Applied Cryptography**
Bruce Schneier
- Slides will be available on the web (PDF)
 - Linked to from the course schedule page.
- Material: Slides, Lectures, Articles, HW, Projects

Assessment

- Individual homework assignments 20%
- Group projects 40%
 - 2 or 3 people per group
 - Programming (in Java)
- Midterms I & II 24%
- Final exam 15%
 - Time & place determined by registrar
 - Covers entire course (mostly security)
- Participation (mandatory and subjective) 01%



Course Policies

- No late homework accepted
 - Unless prior permission
 - Emergency
- No collaboration on individual homework
- No individual work on group projects
- Regrades:
 - Only “reasonable” requests
 - Entire homework is regraded
 - Scores may go up or down

Course Topics

- Software Security / Malicious Code
 - Buffer overflows, viruses, worms, protection mechanisms
- Networks & Infrastructure
 - Ethernet, 802.11, TCP/IP, Denial of Service, IPSEC, TLS/SSL
- Basic Cryptography
 - Shared Key Crypto (AES/DES), Public Key Crypto (RSA)
- Crypto Software & Applications
 - Cryptographic libraries, authentication, digital signatures
- System Security
 - Hacker behavior, intrusion & anomaly detection, hacker and admin tools



Plan for today

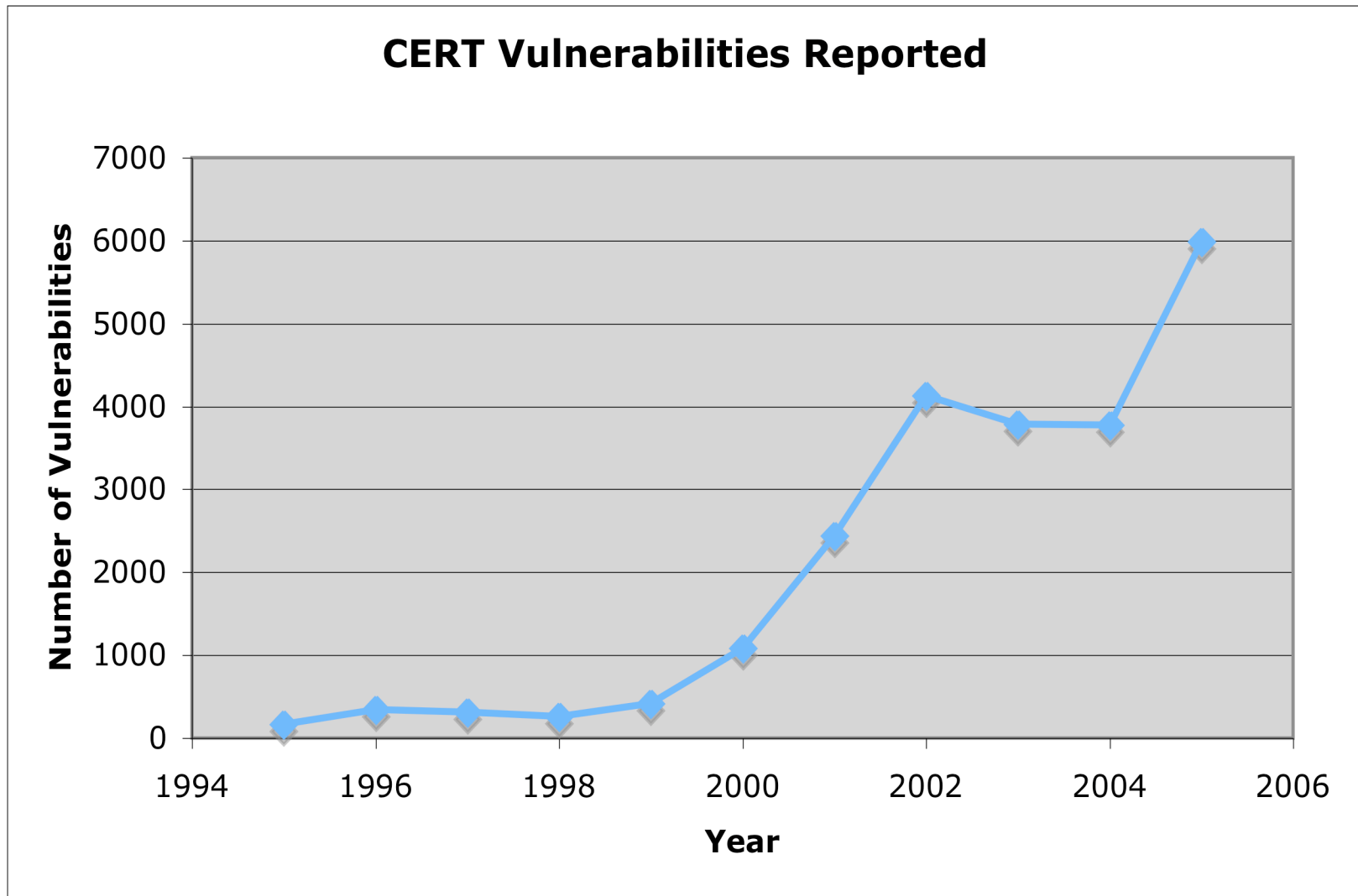
- Try to answer the question:
 - What is security?
 - What do we mean by a secure system?
- Historical context
 - Basic definitions & background
 - Examples
- General principles of secure design



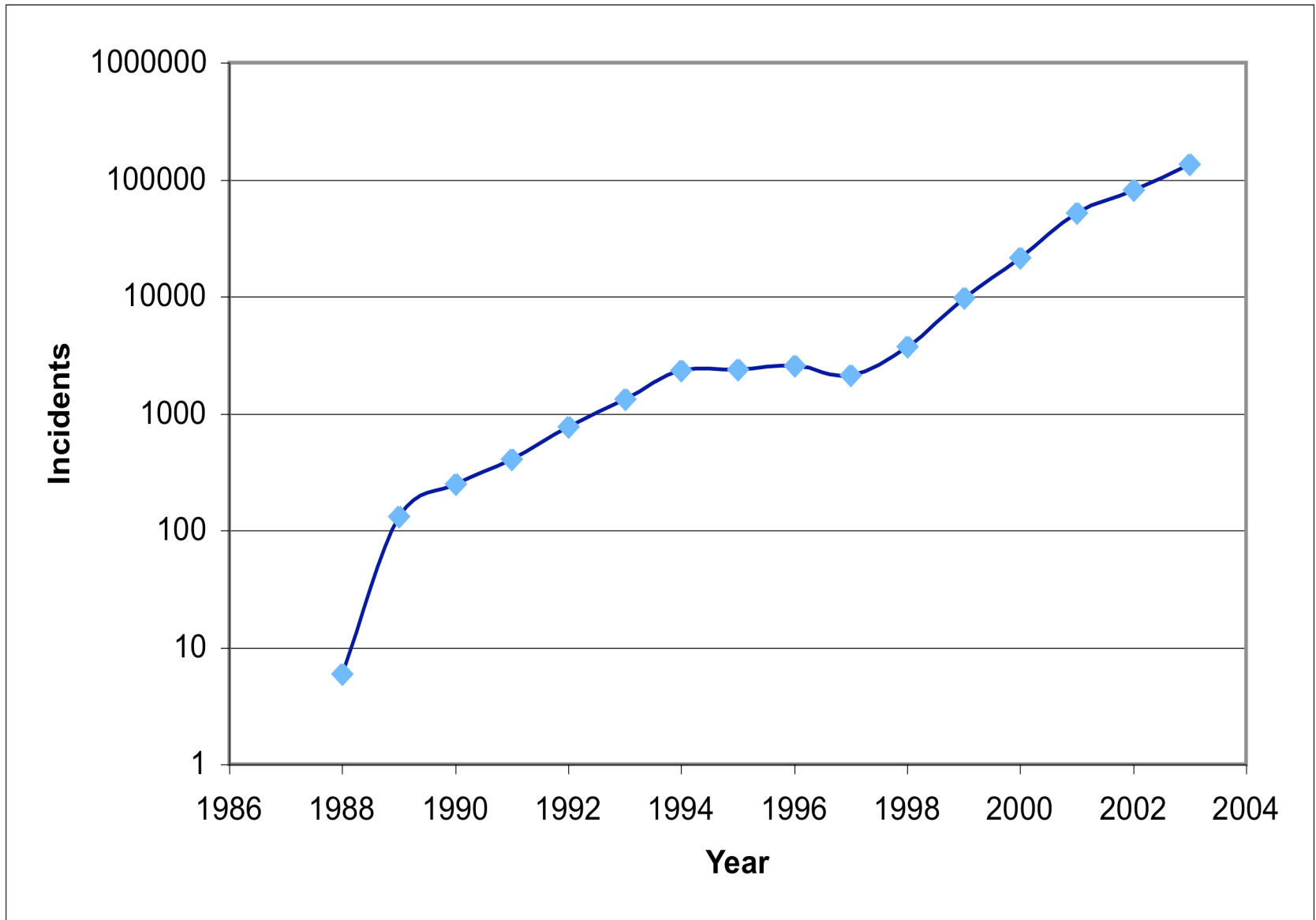
Software Vulnerabilities

- Every day you read about new software *vulnerabilities* in the news
 - Buffer overflow vulnerabilities
 - Format-string vulnerabilities
 - Cross-site scripting vulnerabilities
- Check out www.cert.org for plenty of examples

CERT Vulnerabilities



CERT Incidents





What do we mean by security?

- What does it mean for a computer system to be secure?
- Comments generated from class discussion:
 - The system only does things the way the designer intended.
 - Should prevent unauthorized use.
 - What about spam?

When is a program secure?

- When it does exactly what it should?
 - Not more.
 - Not less.
- But how do we know what a program is supposed to do?
 - Somebody tells us? (But do we trust them?)
 - We write the specification ourselves? (How do we verify that the program meets the specification?)
 - We write the code ourselves? (But what fraction of the software you use have you written?)

When is a program secure?

- 2nd try: A program is secure when it doesn't do something it shouldn't.
- Easier to specify a list of "bad" things:
 - Delete or corrupt important files
 - Crash my system
 - Send my password over the Internet
 - Send threatening e-mail to the president posing as me
- But... what if most of the time the program doesn't do bad things, but occasionally it does? Is it secure?

When is a program secure?

- Claim: Perfect security does not exist.
 - Security vulnerabilities are the result of violating an assumption about the software (or, more generally the entire system).
 - Corollary: As long as you make assumptions, you're vulnerable.
 - And: You *always* need to make assumptions!

- Example: Buffer overflows
 - Assumption (by programmer) is that the data will fit in the buffer.
 - This leads to a vulnerability: Supply data that is too big for the buffer (thereby violating the assumptions)
 - Vulnerabilities can be *exploited* by an *attack*.

When is a program secure enough?

- Security is all about tradeoffs
 - Performance
 - Cost
 - Usability
 - Functionality
- The right question is: how do you know when something is secure enough?
 - Still a hard question
 - Requires understanding of the tradeoffs involved
- Is Internet Explorer secure enough?
 - Depends on context



How to think about tradeoffs?

- What is it that you are trying to protect?
 - Music collection vs. nuclear missile design data
- How valuable is it?
- In what way is it valuable?
 - Information may be important only to one person (e.g. private e-mail or passwords)
 - Information may be important because it is accurate and reliable (e.g. bank's accounting information)
 - A computer system may be important because of a service it provides (e.g. Google's web servers)

Historical Context

- Assigned Reading:
 - Saltzer & Schroeder 1975
The Protection of Information in Computer Systems
 - Chapter 1 of Anderson's book:
Security Engineering
 - Both available from course web pages
- Unauthorized information release
 - *Confidentiality*
- Unauthorized information modification
 - *Integrity*
- Unauthorized denial of use
 - *Availability*
- What does “unauthorized” mean?



Example Security Techniques

- Labeling files with a list of authorized users
 - Access control (must check that the user is permitted on access)
- Verifying the identity of a prospective user by demanding a password
 - Authentication
- Shielding the computer to prevent interception and subsequent interpretation of electromagnetic radiation
 - Covert channels
- Enciphering information sent over telephone lines
 - Cryptography
- Locking the room containing the computer
 - Physical aspects of security
- Controlling who is allowed to make changes to a computer system (both its hardware and software)
 - Social aspects of security



Building Secure Software

- Source: book by John Viega and Gary McGraw
 - Strongly recommend buying it if you care about implementing secure software.
- Designing software with security in mind
- What are the security goals and requirements?
 - Risk Assessment
 - Tradeoffs
- Why is designing secure software a hard problem?
- Design principles
- Implementation
- Testing and auditing



Security Goals

- Prevent common vulnerabilities from occurring (e.g. buffer overflows)
- Recover from attacks
 - Traceability and auditing of security-relevant actions
- Monitoring
 - Detect attacks
- Privacy, confidentiality, anonymity
 - Protect secrets
- Authenticity
 - Needed for access control, authorization, etc.
- Integrity
 - Prevent unwanted modification or tampering
- Availability and reliability
 - Reduce risk of DoS



Other Software Project Goals

- Functionality
- Usability
- Efficiency
- Time-to-market
- Simplicity

- Often these conflict with security goals
 - Examples?

- So, an important part of software development is risk assessment/risk management to help determine the design choices made in light of these tradeoffs.

Risk Assessment

- Identify:
 - What needs to be protected?
 - From whom?
 - For how long?
 - How much is the protection worth?
- Refine specifications:
 - More detailed the better (e.g. "Use crypto where appropriate." vs. "Credit card numbers should be encrypted when sent over the network.")
 - How urgent are the risks?
- Follow good software engineering principles, but take into account malicious behavior.



Principles of Secure Software

- What guidelines are there for developing secure software?
- How would you go about building secure software?

Class answers:



#1: Secure the Weakest Link

- Attackers go after the easiest part of the system to attack.
 - So improving that part will improve security most.
- How do you identify it?
- Weakest link may not be a software problem.
 - Social engineering
 - Physical security
- When do you stop?



#2: Practice Defense in Depth

- Layers of security are harder to break than a single defense.
- Example: Use firewalls, and virus scanners, and encrypt traffic even if it's behind firewall

#3: Fail Securely

- Complex systems fail.
- Plan for it:
 - Aside: For a great example, see the work of George Candea who's Ph.D. research is about something called "microreboots"
- Sometimes better to crash or abort once a problem is found.
 - Letting a system continue to run after a problem could lead to worse problems.
 - But sometimes this is not an option.
- Good software design should handle failures gracefully
 - For example, handle exceptions

#4: Principle of Least Privilege

- Recall the Saltzer and Schroeder article
- Don't give a part of the system more privileges than it needs to do its job.
 - Classic example is giving root privileges to a program that doesn't need them: mail servers that don't relinquish root privileges once they're up and running on port 25.
 - Another example: Lazy Java programmer that makes all fields public to avoid writing accessor methods.
- Military's slogan: "Need to know"

#5: Compartmentalize

- As in software engineering, modularity is useful to isolate problems and mitigate failures of components.
- Good for security in general: Separation of Duties
 - Means that multiple components have to fail or collude in order for a problem to arise.
 - For example: In a bank the person who audits the accounts can't issue cashier's checks (otherwise they could cook the books).
- Good examples of compartmentalization for secure software are hard to find.
 - Negative examples?

#6: Keep it Simple

- KISS: Keep it Simple, Stupid!
- Einstein: "Make things as simple as possible, but no simpler."
- Complexity leads to bugs and bugs lead to vulnerabilities.
- Failsafe defaults: The default configuration should be secure.
- Ed Felten quote: "Given the choice between dancing pigs and security, users will pick dancing pigs every time."

#7: Promote Privacy

- Don't reveal more information than necessary
 - Related to least privileges
- Protect personal information
 - Consider implementing a web pages that accepts credit card information.
 - How should the cards be stored?
 - What tradeoffs are there w.r.t. usability?
 - What kind of authentication/access controls are there?

#8: Hiding Secrets is Hard

- The larger the secret, the harder it is to keep
 - That's why placing trust in a cryptographic key is desirable
- Security through obscurity doesn't work
 - Compiling secrets into the binary is a bad idea
 - Code obfuscation doesn't work very well
 - Reverse engineering is not that difficult
 - Software antipirating measures don't work
 - Even software on a "secure" server isn't safe (e.g. source code to Quake was stolen from id software)

#9: Be reluctant to trust

- Trusted Computing Base: The set of components that must function correctly in order for the system to be secure.
- The smaller the TCB, the better.
- Trust is transitive
- Be skeptical of code quality
 - Especially when obtained from elsewhere
 - Even when you write it yourself



#10 Use Community Resources

- Software developers are not cryptographers
 - Don't implement your own crypto
 - (e.g. bugs in Netscape's storage of user data)
- Make use of CERT, Bugtraq, developer information, etc.