CIS 551 / TCOM 401

Computer and Network Security

Spring 2007

Lecture 1

Course Staff

- Steve Zdancewic (Instructor)
  - E-mail: stevez@cis.upenn.edu
  - Web: www.cis.upenn.edu/~stevez
  - Office hours: Tues: 9:30 - 10:30 am, and by appointment
  - Office: Levine 511
- Jeff Vaughan (Teaching assistant)
  - E-mail: vaughan2@seas.upenn.edu
  - Office hours: Weds: 3:00
  - Office: Levine 514

Course Information

- Course Web Page:
  - www.cis.upenn.edu/~cis551
- News group:
  - upenn.cis.cis551
- Textbook: none
  - Assigned reading: articles and web pages
  - Lecture slides will be available on the course web pages
  - Student scribes: Designated note takers

Prerequisites

- Would like to learn about computer and network security.
- Some programming experience
  - Java
  - C or C++ helpful (but not necessary - you can pick up what you need to know)
- Some computer networks experience
  - Do you know what a protocol stack is?
  - Do you generally understand TCP/IP?
  - TCOM 500

Grading Criteria

- 16% Midterm I - tentative date: Feb. 8th
- 16% Midterm II - tentative date: Mar. 20th
- 25% Final exam
- 40% Course projects (group projects)
- 03% Course participation

- Policies:
  - No individual work on group projects
  - Only “reasonable” regrade requests permitted
  - See course web pages

Announcement

- I will be out of town next Tuesday
  - Jeff Vaughan will be giving the lecture

Student Background…

1. How many of you have programmed in C or C++?
2. How many of you have programmed in Java?
3. How many of you have written shell scripts?
4. How many of you have never done any programming?
5. How many of you can explain how a buffer overflow exploit works?
6. Have any of you written a buffer overflow exploit?
7. How many of you can explain how TCP/IP works?
8. How many of you have set up a wireless network?
9. How many of you have had experienced a virus or worm attack on some computer you care about?
10. Have any of you written a virus or worm?

Student Background…

11. How many of you regularly use SSH or SFTP?
12. How many of you can explain how they work?
13. How many of you have run a packet sniffer or port scanner?
14. How many of you can define the term “Trusted Computing Base”?
15. How many of you have used a debugger?
16. How many of you are masters students?
17. How many of you are PhD students?
18. How many of you are undergraduates?

Course Topics

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

Outline

- Try to answer the questions:
  - What is computer security?
  - What do we mean by a secure program?
- Historical context
  - Basic definitions & background
  - Examples of security
- General principles of secure design
- Focus on one widespread example:
  - Buffer overflows

Software Vulnerabilities

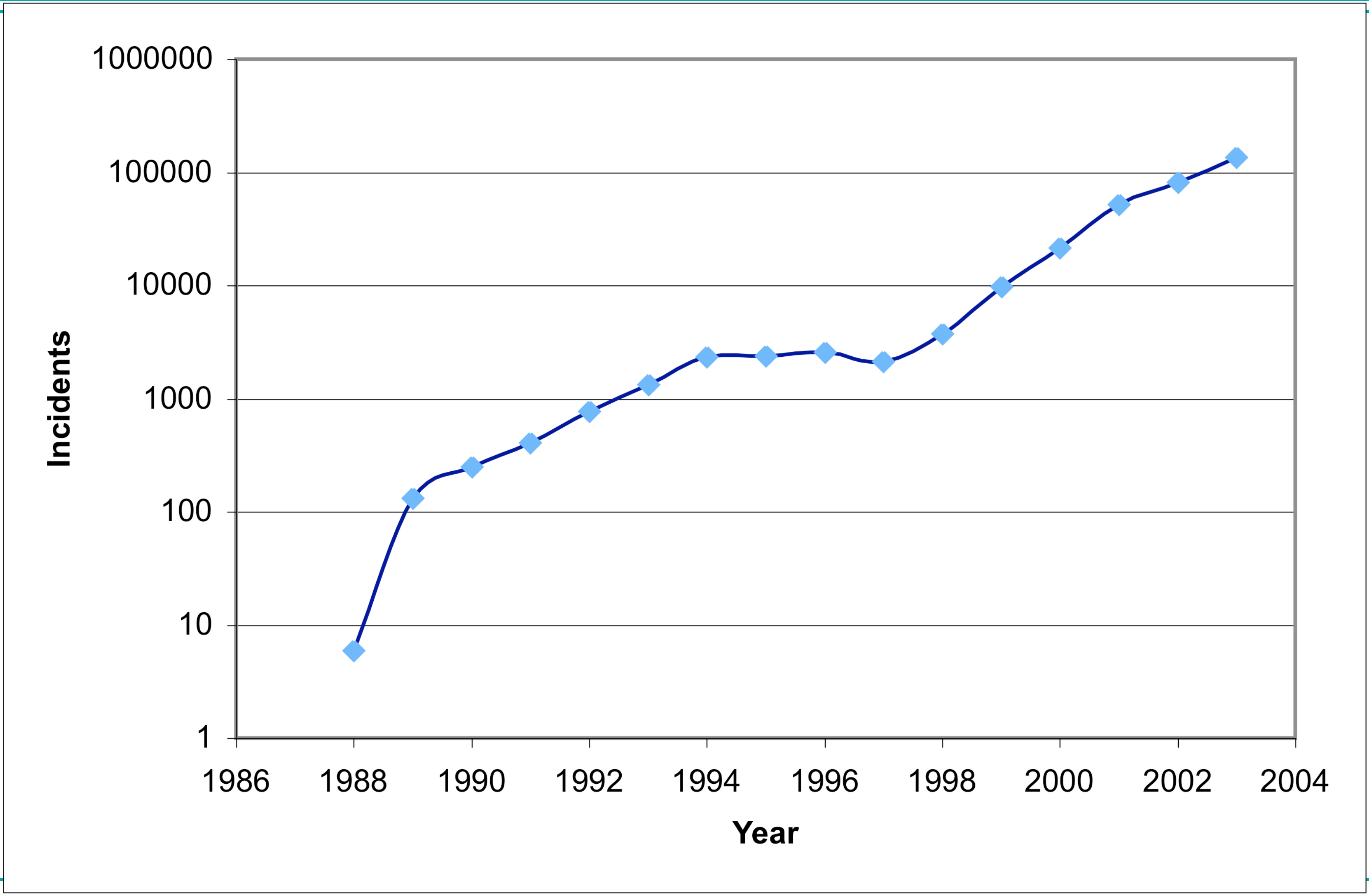
- Every day you read about new software vulnerabilities in the news
  - Buffer overflows
  - Cross-site scripting
  - Format-string vulnerabilities
  - Spam
  - Worms/Viruses
  - Phishing

- Check out www.cert.org for plenty of examples

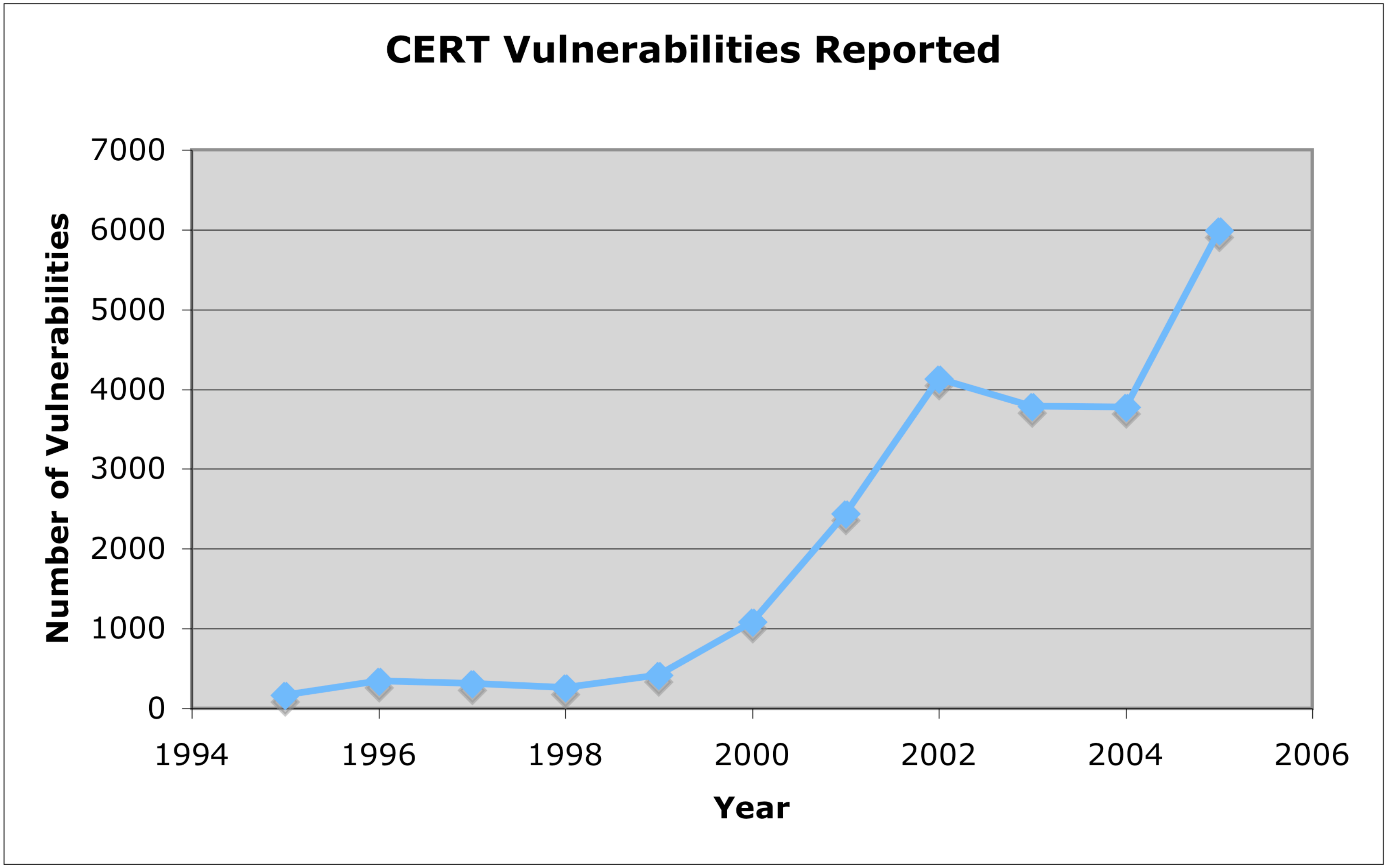
Slashdot Security Headlines in 2007

- Microsoft Gets Help From NSA for Vista Security
- NYT Security Tip - Choose Non-Microsoft Products
- Blurring Images Not So Secure
- The NYT on the Proliferation of Botnets
- AJAX May Be Considered Harmful
- Opera Security Patched In Secret
- Voice Over IP Under Threat?
- A Tour of the Google Blacklist
- Hackers Disagree On How, When To Disclose Bugs
- Social Networking Site Safety Questioned
- IE6 Was Unsafe 284 Days In 2006
- Adobe Acrobat JavaScript Execution Bug
- Five Hackers Who Left a Mark on 2006
- Memories of a Media Card
- DieHard, the Software
- GMail Vulnerable To Contact List Hijacking

CERT Incidents



CERT Vulnerabilities



What do we mean by security?

- What does it mean for a computer system to be secure?
- Comments generated from class discussion:
  - Only those users who are permitted to use the system can.
  - The system protects privacy/secrecy/confidentiality/anonymity.
  - The system can't be abused. (Is only used for its designed purpose.)
  - Detect error conditions & react appropriately. (How do you detect the error/anomaly?)
  - Stability and consistency -- reliability or availability.
  - Only the services that should be running are.
  - Backup -- in case of failure.
  - Auditing, logging -- watching the system.

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
  - Usabilitity
  - 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
  - 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
  - Copy on reserve in the library
  - 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:
  - Threat model -- what are the threats? Risk assessment.
  - Quickly recover from worst case scenario.
  - Auding -- both the software and function of the system
  - Separate security critical functions from others -- compartmentalization.

#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.

Case Study: Buffer Overflows

- First project: Due: 1 Feb. 2007 at midnight
- http://www.cis.upenn.edu/~cis551/project1.html

- Group project:
  - 2 or 3 students per group
  - Send e-mail to Jeff Vaugan with your group by Jan. 25th

- Assigned Reading:
  Aleph One (1996)
  Smashing the Stack for Fun and Profit
- This is essentially a tutorial for the project