

CMSC 838F, Spring 2008: Language-Based Security Introduction

Jeff Foster

Many slides stolen from Steve Zdancewic and Andrew Myers:

<http://www.cis.upenn.edu/~cse331/lectures/CSE331-01.pdf>

<http://www.cs.cornell.edu/andru/pldi06%2Dtutorial/06jun-pldi-tutorial.pdf>

<http://www.cs.uoregon.edu/research/summerschool/summer04/lectures/myers.pdf>

Prerequisites

- CMSC 631, or
- Another grad-level PL class, or
- Permission of the instructor

2

Grading

- Class participation (35%)
 - Written reviews of papers
 - Homework and programming projects
 - In-class presentation of paper
 - In-class discussion
- Project (40%)
 - A substantial research project
- Final Exam (25%)

3

Reading

- There is no required textbook for this class
- Will read approximately three papers per week for remainder of semester
 - Write **one** paper review per week
 - Must still read other papers!
 - Scribe **1-2** discussions

4

Homework and Programming Projects

- From time to time, we will have short homeworks or programming projects
 - 1-4 during the semester
 - First one available next Monday
- Must be individual efforts
 - Due at time stated on project assignment
 - Follow submission instructions on assignment
 - No late assignments accepted
 - Email me in advance if you have a conflict

5

Course Project

- Largest component of your grade
- Three choices:
 - Survey of an area we will not cover in class
 - Read 4-6 papers carefully, skim 4-6 more, write up discussion
 - Individual or small group project
 - Choose your own topic
 - “Big” class-wide project
 - Details TBA (when I figure them out!)
- Must decide on project by **Monday, March 3**
 - Periodic project updates due throughout semester
 - Project milestones **will be graded**

6

Final Exam

- Most likely a take-home exam given out on/near last class
 - Will have 24-48 hours to do exam (TBA)
- Most likely will have a broad range of questions on all papers we covered
 - You choose \geq 100 points of questions to answer

7

Topics

- Exact mix is up to you!
 - If there’s something else you want to cover, or a paper that’s interesting, please suggest it
- Current topics (tentative)
 - What is “security”?
 - Buffer overruns
 - SQL injection attacks and cross-site scripting
 - Format-string vulnerabilities
 - Other attacks
 - Secure information flow
 - Stack inspection and access control
 - Reference monitors and software fault isolation
 - Protocol analysis

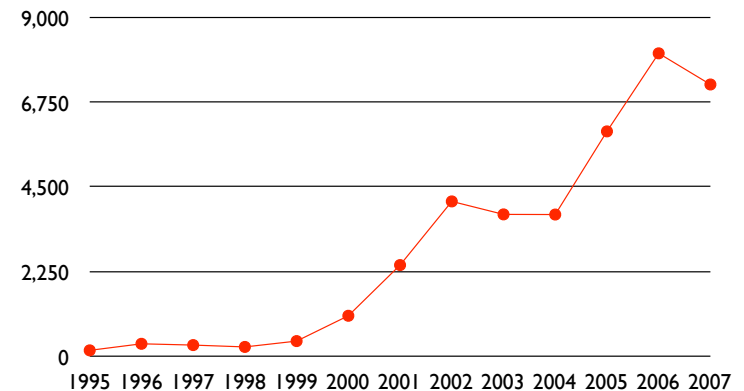
8

Motivation

- In the 1970's, computing systems were isolated
 - Software updates infrequent, done by admin
 - Users always trusted code they ran
 - Physical access to machine required
 - Crashes and outages had limited effect
- The Internet has changed all this
 - Depend on software for everyday services
 - Software constantly updated
 - Remote hackers are as close as your neighbor
 - Everything is executable (web pages, email, ...)

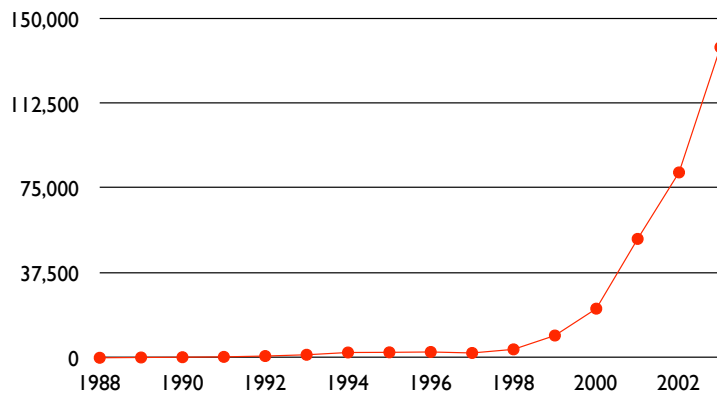
9

CERT Vulnerabilities Reported



10

CERT Incidents Reported



11

What is Security?

- Goal: prevent bad things from happening
 - Clients not paying for services
 - Critical services unavailable
 - Confidential information leaked
 - Important information damaged
 - System used to violate laws
 - Money stolen
 - Loss of value
- Prevent **an adversary** from doing these bad things
 - Normal flaws avoided by users
 - Malicious adversary seeks out weaknesses

12

When is a Program Secure?

- When it does exactly what it should!
 - But what is it supposed to do?
 - Someone tells us (do we trust them?)
 - We decide ourselves (how do we check?)
 - We write the code ourselves (how much of the software you use have you written?)
- Perfect security does not exist
 - Must trade off performance, cost, usability, functionality
 - When is software “secure enough”?

13

Language-Based Security

- Conventional security: program is a black box
 - Encryption
 - Firewalls
 - System calls/privileged mode
 - OS-provided process isolation
- Major limitations — doesn’t address:
 - Downloaded and mobile code
 - Buffer overruns and other software flaws
 - Application-level security policies
 - System-wide security guarantees
- PL technology to the rescue!

14

Example #1: Morris Worm

- 1988: Penetrated estimated 5-10% of 6,000 machines on the internet
- Used a number of clever methods to gain access to a host
 - Brute force password guessing
 - Bug in default sendmail configuration
 - X windows vulnerabilities, rlogin, etc
 - Buffer overrun in fingerd
- Remarks
 - System diversity helped to limit the spread
 - “Root kits” for cracking modern systems are easily available and largely use the same techniques

15

Example #2: Love Bug, Melissa

- 1999: Two email-based viruses that exploited
 - A common mail client (MS outlook)
 - Trusting (i.e., uneducated) users
 - VB scripting extensions within messages to
 - Look up address in the contacts database
 - Send a copy of the message to those contacts
- Melissa: Hit an estimated 1.2 million machines
- Love Bug: Caused an estimated \$10B in damage
- Remarks:
 - No passwords, crypto, or native code involved

16

Example #3: Hotmail

- 1999: All Hotmail email accounts fully accessible by anyone, without a password
 - Just change username in an access URL (no programming required!)
- Selected other Hotmail headlines
 - Hotmail bug allows password theft
 - Hotmail bug pops up with JavaScript code
 - Malicious hacker steals Hotmail passwords
 - New security glitch for Hotmail
 - Hotmail bugfix not a cure-all

17

Example #4: Yorktown

- 1998: “Smart Ship” USS Yorktown suffers propulsion system failure, is towed into Norfolk Naval Base
- Cause: computer operator accidentally types a zero, causing divide-by-zero error that overflows database and crashes all consoles
- Problem fixed **two days** later

18

Example #5: Samy Worm

- 2005: MySpace user Samy Kamkar found cross-site scripting vulnerability in MySpace
- Users who visited his page
 - Became friends with him
 - Displayed “but most of all, Samy is my hero” on their profile
 - New user who viewed profile infected
- Within 20 hours, >1,000,000 users infected

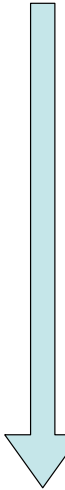
19

Example #6: Insiders

- Average cost of an outsider penetration is \$56,000
- Average insider attack cost \$2.7 million
 - (~2004, Computer Security Institute/FBI)
- Rogue trader cost Société Générale €3.7B!
 - Most likely cause of stock market sell-off last week
- 63% of companies surveyed reported insider misuse of their organization’s computer systems
 - (WarRoom Research)
- Some attacks: Backdoors, “Logic bombs”, Hiding data hostage with encryption, Reprogramming cash flows
- Attacks may use legitimately held privileges!
- Many attacks (90%?) go unreported

20

Who are the Attackers?



Operator/users who make mistakes
Hackers driven by intellectual challenge
Insiders: employees or customers seeking revenge/gain
Criminals seeking financial gain
Organized crime seeking gain or hiding criminal activities
Organized terrorist groups or nation states trying to influence national policy
Foreign agents seeking information for economic, political, or military purposes
Tactical countermeasures intended to disrupt military capability

21

What are the Vulnerabilities?

- Poorly-chosen passwords
- Software bugs
- Automatically running active content
 - Macros, scripts, Java programs
- Incorrect configuration
 - File permissions
 - Administrative privileges
- Untrained users/system admins
- Trap doors (intentional security holes)
- Unencrypted communication
- Limited Resources

22

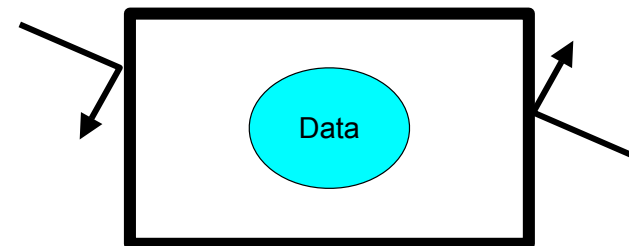
What are the Attacks?

- Password guessers/crackers
- Viruses
- Worms
 - Viruses require user interaction, worms don't
- Trojan Horses
- Root kits
- Phishing
- Packet sniffers
- Denial of service

23

Integrity

- No improper modification of data

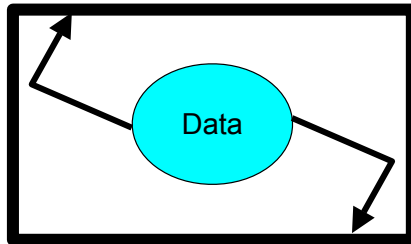


- E.g., account balance updated only by authorized transactions, only you can change your password
- Integrity of security mechanism is crucial
- Enforcement: access control, digital signatures, ...

24

Confidentiality

- Protect information from improper release

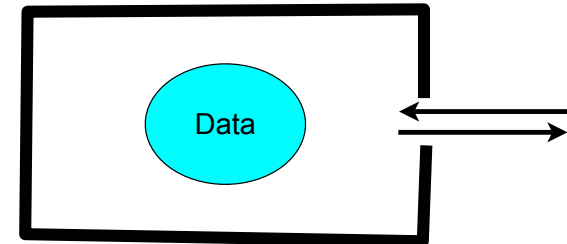


- Limit knowledge of data or actions (secrecy)
 - E.g., D-Day attack date, contract bids
- Enforcement: access control, encryption, ...
- Hard to enforce after the fact...

25

Availability

- System must respond to requests



- I.e., do not ensure confidentiality and integrity by unplugging your computer!

26

Other Properties

- Privacy: prevent misuse of personal information
- Anonymity: prevent connection from being made between identity of actor and actions

- Attack detection
 - Monitor, intrusion detection
- Recovery from attacks
 - Traceability and auditing of security-relevant actions

- Others?

27

Project Goals Conflict with Security

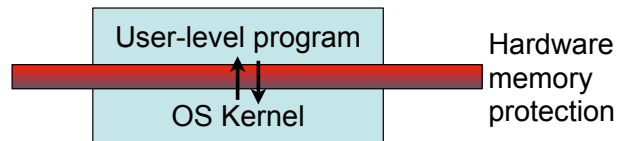
- Functionality
- Usability
- Efficiency
- Time-to-market
 - Hard to achieve these and be secure

- Risk assessment
 - *What* needs to be protected, from *whom*, and for *how long*? What is the protection *worth*?

28

Conventional OS Security

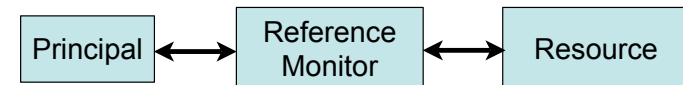
- Model: Program is black box
- Program talks to OS via system calls
 - Protected interface
 - Multiplexes hardware
 - Isolates processes from each other
 - Restricts access to resources (sockets, files, etc)
- Language-independent, simple, limited



29

Access Control Model

- The standard way to prevent “bad things”
- *Principals* make *requests* to access resources (*objects*)
- *Reference monitor* (e.g., kernel) permits or denies request



30

Authentication vs. Authorization

- Abstraction of a principal divides enforcement into two parts
 - *Authentication* — who is making the request?
 - *Authorization* — are they allowed to make the request?

31

Design Principle: Complete Mediation

- Principle of complete mediation:
 - *Every access to every object must be checked by the reference monitor*
- Issues
 - System must have ability to mediate all security-relevant operations
 - Dangerous to assume operation is not relevant
 - Many places to mediate: hardware, compiler, ...
 - Assumption: Mediation mechanism cannot be compromised
 - Problem: OS-level security does not support complete mediation

32

OS: Coarse-grained Control

- OS enforces security at system call layer
 - Hard to control application when it is not making system calls
- Security enforcement decisions made with regard to large-granularity objects
 - Files, sockets, processes
- Coarse notion of principal
 - If you run an untrusted program, should the authorizing principal be “you”?

33

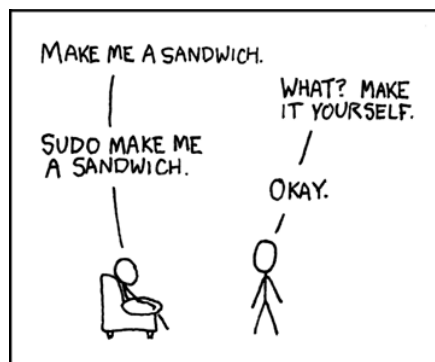
Need: Fine-grained Control

- Modern programs make security decisions with respect to *application* abstractions
 - UI: access control at window level
 - Mobile code: No network send after file read
 - E-commerce: No goods until payment
 - Intellectual property: DRM
- Need extensible, reusable mechanism for enforcing security policies
 - Language-based security can support an extensible protected interface, e.g., the Java security model

34

Design Principle: Least Privilege

- Principle of least privilege:
 - *Each principal is given the minimum access needed to accomplish its task [Saltzer & Schroeder 1975]*



<http://xkod.com/149/>

35

Design Principle: Least Privilege

- Principle of least privilege:
 - *Each principal is given the minimum access needed to accomplish its task [Saltzer & Schroeder 1975]*
- Issues:
 - What is the minimal set of privileges?
 - What is the granularity of privileges?
 - Separation of privileges (read vs. write access)
- How and when do the privileges change?
 - Example violations?

36

Least Privilege Problems

- OS privilege is coarse: user/group
 - Applications need finer granularity
 - Web apps: principals unrelated to OS
- Who is the “real” principal?
 - Trusted program? Full power of the user
 - Untrusted? Something less
 - Trusted program with untrusted extension?
 - Untrusted program accessing secure trusted system?
- Requests may filter through a chain of programs or hosts
 - Loss of information is typical
 - E.g., client browser to web server to web app to database

37

Design Principle: Minimize TCB

- TCB = Components whose failure compromises the security of a system
- Complex things are more likely to not work correctly
 - Make trusted computing base as small and simple as possible
 - *“Things should be made as simple as possible—but no simpler.” — A. Einstein*
 - Fewer errors in implementation, easier to convince someone that it’s correct

38

Small TCB and LBS

- Conventional wisdom (1975)
 - OS is small and simpler, compiler is large and complex
 - OS is a small TCB, compiler a large one
- Today
 - OS = millions of lines of code
 - Hard to believe is implemented correctly!
 - Many authors (untrustworthy: device drivers)
 - Implementation bugs
 - Compiler = 100k lines of code
 - Possible to prove correct (in theory)
 - Smaller, functional, not concurrent

39

Design Principle: Be Open

- Success of mechanism should not depend on it being secret
 - No security through obscurity
 - The only secrets are cryptographic keys
 - Increased assurance if many critics
- An age-old controversy:
 - Open design makes critics’ jobs easier, but also attackers’ job
 - Analysis tends to concentrate on core functionality; vulnerabilities remain off the beaten path

40

Design Principle: Security is a Process

- Every system has vulnerabilities
 - Impossible to eliminate all of them
 - Goal: Assurance
- Systems change over time
 - Security requirements change over time
 - Context of mechanisms changes over time
- Secure systems require maintenance
 - Check for defunct users
 - Update virus software
 - Patch security holes
 - Test firewalls