ISA 862
Models for Computer Science

Class Mechanics &
Security Principles and Threats

Angelos Stavrou, George Mason University
Class Mechanics: Prerequisites

Prerequisites

- Computer Architecture
- Working knowledge of C/C++/Java

- We will have extra Labs to Cover Programming
- I am fairly reachable via email, contact me or the TA
- There are no stupid questions
Prerequisites: Computer Architecture

- Must have completed an undergraduate course on Computer Architecture/Organization (e.g. at GMU: CS 465, ECE 445, INFS 515)

- Must be familiar with concepts such as:
  - Instruction set architectures
  - Assembly language
  - CPU control and datapaths
  - Instruction cycle (fetch/decode/execute)
  - Register Organization
  - Memory Hierarchy
  - Basic I/O Operations
Prerequisites: Programming Background

- Must have completed a course on Data Structures and Program Design (e.g. CS 310, INFS 519)
- Has working knowledge of C, C++ or Java
- It will be assumed that you are truly comfortable with programming; the projects will involve advanced concepts that will build on the fundamentals
  - Multi-threaded programming
  - Distributed system programming and design
Textbook

- **Required:**
  *Understanding the Linux Kernel, Third Edition*
  O'Reilly Media Available by: [Online for GMU] [O'Reilly] [Amazon]

- **Recommended:**
  *Operating System Concepts, 8th Edition (or 7th Edition)*,
  Available by: [GMU Bookstore] [Wiley] [Amazon] [Author's site]

[GMU Bookstore] [Prentice Hall] [Amazon]
Class Mechanics: Grading

Grading

- Programming Projects (2 projects): 80%
- Class Presentations: 15%
- Class Participation: 5%
- No Midterm or Final

- Projects are going to be Collaborative
- You cannot be on your own, you need to pick a team
Class Mechanics: Contacts

Instructor:
- Angelos Stavrou, (astavrou_at_gmu.edu)
- Office Hours: Thursday 2:00 – 4:00pm, and by appointment
- Office: Research I, Room 437

Teaching Assistant:
- Chen Liang, (cliang1_at_gmu.edu)
- Office Hours: Tuesday, 4:00 – 6:00pm.
- Office: Research I, Room 438
Class Mechanics: Accounts & Lectures

Accounts & Lectures:

- Computer Accounts on mason.gmu.edu or zeus.ite.gmu.edu  
  Check if you have one!

- Class Web Page:  
  http://cs.gmu.edu/~astavrou/ISA862_F10.html

- E-mails with additional info on lectures and assignments will be sent to students’ GMU e-mail addresses

- All Class material will be posted at the class web page
Course Objectives

Why study Security Architectures & Systems?

- Understanding the *principles* behind the design of *Security Systems in different Environments*
- Observing the Deployment Challenges and how Security principles are put into *practice* in practice
- Discussing both “solved” and “open” problems and issues in Security design, recent trends
- Gaining hands-on experience in
  - Mobile Device Security
  - Emerging Security Threats and Defenses
Topics

- Anonymity and Privacy
- Security Models for Mobile and Hand-held devices
- Attacks against networks and machines
- Content-based Attacks
- Advanced Persistent Threats (APTs)
- Operating System Security
- Analysis of Security protocols
- Forensics and diagnostics for security
- Botnets, Malicious code analysis, anti-virus, anti-spyware
What is Security?

- **Computer Security**
  - Techniques for computing in the presence of adversaries
  - Three categories of security goals
    - **Confidentiality**: preventing unauthorized release of info
    - **Integrity**: preventing unauthorized modification of info
    - **Availability**: preventing Denial of Service (DoS) attacks
  - Protection is about providing all three on a single machine »
    Usually considered the responsibility of the OS » Could also be runtime (e.g., verification in JVM)

- **Cryptography**
  - Techniques for communicating even in the presence of adversaries
  - Good for link-level protection
  - No silver bullet: Key-management the weak point
What is Security?

- Security must consider also the external environment of the system, and protect it from:
  - unauthorized access
  - malicious modification or destruction

- Common security breaches
  - *Breach of confidentiality*: unauthorized reading (or, theft) of data
  - *Breach of integrity*: unauthorized modification of data
  - *Breach of availability*: unauthorized destruction of data
  - *Theft of service*: unauthorized use of services
  - *Denial of service*: prevents the legitimate use of the system
Cryptographic Operations

- **Encryption & Decryption (confidentiality)**
  - Given a message and a secret key, create a ciphertext
  - Goal is that ciphertext is confidential
    - Need the key to convert ciphertext back to message

- **Signing & Verifying (Authentication & integrity)**
  - Messages can be signed by the sender (often using their key)
  - Given a (signed) message and the supposed identity of a sender, verify the message authenticity
  - Message Integrity. Protection against Replay attacks?

- **Main design point is whether secret keys are shared**
  - Symmetric (fast, cheap) vs Public/Private key pair (easy distribute)
Security Design Principles

- Security is much, much more than just crypto
  - Crypto is a crucial mechanism for security, but is not a panacea
  - If there is a fundamental flaw in the design of the system, then all of the crypto in the world won’t help you
  - It is usually easier to find a bug in an implementation than circumvent a crypto system

- Unfortunately, systems design is still as much an art as it is a science
  - But, decades of building systems the wrong way have helped us collect some learned wisdom

- More about this shortly
Principle: Least Privilege

- Figure out exactly which capabilities a program needs to run, and grant it only those
  - Not always easy, but one algorithm: start with granting none, run and see where it breaks, add new privileges, repeat

- Unix
  - Good example: Should not normally run as root to prevent against accidents
  - Bad example: Some programs run as root just to get a small privilege, such as using a port < 1024 (privileged port)
    - E.g., ftpd
    - Exploit these programs, and you get root access to system
    - Running under a unprivileged user
    - Containing the services to a small subset of capabilities
Bad Design Example: wu-FTPD

- **wu-ftpd** tries to run with least privilege
  - But occasionally tries to elevate its privilege with:
    ```c
    seteuid(0);
    // privileged critical section runs here
    seteuid(getuid());
    ```

- **However, wu-ftpd** does not disable Unix signals
  - **wu-ftpd** doesn’t relinquish privileges after signal handler
  - While in critical section, can be “tractor-beamed” away to a signal handler
    - Does not return to original control flow

- **Remote user** can cause a signal handler to run by terminating a download in midstream!
  - But need to catch wu-ftpd in the critical section
  - Result: Can abort a download and then use wu-ftpd as root
Principle: Least Common Mechanism

- Be very careful integrating shared or reused code
  - Assumptions made may no longer be valid in current context

- Counter example: Outlook and Internet Explorer
  - Windows exports an API to IE’s HTML rendering code
    - Outlook and other programs use this to display HTML in email
    - By default, JavaScript and Java parsing are enabled
  - HTML rendering code knows Java(Script) is unsafe
    - Disables it when JavaScript is downloaded from Internet
    - Only enables it when loaded from trusted sources
      - Your own file system is trusted
  - But...email is spooled on disk...
Principle: Complete Mediation

- Check every access to every object
  - Of course, this introduces overhead
  - So, implementers try to get away with less (caching)
  - But only when nothing relevant in environment has changed

- Counter example: NFS and file handles
  - Client contacts remote “mountd” to get a file handle to a remotely exported NFS file system
    - Remote mountd checks access control at mount time
  - File handle is a capability: client presents it to read/write file
    - Client responsible for enforcing per-file restrictions
  - An eavesdropper can sniff file handle and access file system
Principle: ToCtToU

Time of Check to Time of Use

- Check permissions as close as possible to action
- Complete mediation gets even tougher with multiprogramming
  - Attacker can execute concurrently with TCB
  - Improper synchronization can lead to race conditions
  - Period between verifying authorization and execution is a critical section

- Why is ToCtToU important?
  - Parallel Multiprogramming/Mult-processing
  - Cluster Systems
  - Does this remind you of locking of shared objects?
Principle: ToCtToU

Time of Check to Time of Use

- Counter example: set-uid UNIX programs
  - Many utilities run with effective ID of root; allows regular users to perform super-user actions. May also access user’s files

```c
if (access(filename, W_OK) == 0) {
  if ((fd = open(filename, O_WRONLY)) == NULL) {
    return (0);
  }
}
// Access file
```
Principle: Deny by Default

- Deny all access first, then allow only that which has been explicitly permitted
  - Oversights will then show up as “false negatives”
    - Somebody is denied access who should have it
    - They will complain.
  - Opposites lead to “false positives”
    - Somebody is given access that shouldn’t get it
    - Not much incentive to report this kind of failure…

- Counter examples
  - SunOS shipped with “+” in /etc/hosts.equiv
    - Essentially lets anyone login as any local user to host
  - Irix shipped with “xhost +”
    - Any remote client can connect to local X server
Security through obscurity

- Attempting to gain security by hiding implementation details
- Claim: A secure system should be secure even if all implementation details are published
  - In fact, systems become more secure as people examine and check the implementation details and find flaws
  - Rely on mathematics and sound design to provide security
- Many well-published algorithms are still secure (e.g., SSL)

Counter example: GSM cell phones

- GSM committee designed their own crypto algorithm, but hid it from the world
  - Social engineering + reverse engineering revealed the algorithm
  - Turned out to be relatively weak, easy to subvert
Security is an ongoing effort

- More complex systems are being deployed
  - More and more lives are being trusted to them

- Bruce Schneier: 3 waves of security attacks
  - 1st wave: physical attacks on wires and hardware
    - Physical security to defend against this
  - 2nd wave: syntactic attacks on crypto protocols and systems
    - E.g., buffer overflows, DDoS attacks
  - 3rd wave: semantic attacks: humans and computers trust information that they shouldn’t
    - E.g., Phishing, falsified press announcements
    - Emulex corp stock hoax: CEO “resigns”, 61% stock drop
    - Semantic attack against people with preprogrammed sell orders

- User education can help us defend!
Protection Mechanisms

- In a computer system, each object should be accessed through a well-defined set of operations.

- *Protection problem* - ensure that each object is accessed through the well-defined operations and only by those processes that are allowed to do so.

- *Least privilege principle*: Programs and users should be given just enough privileges to perform their tasks (Not easy to achieve!)
Domain Structure

- A process operates within a *protection domain*.
  - Each domain defines a set of objects and the types of operations that may be invoked on objects.
  - Static or dynamic association

- Access-right = <object-name, rights-set>
  where *rights-set* is a subset of all valid operations that can be performed on the object.
The model can be viewed as a matrix (*access matrix*)

- Rows represent domains
- Columns represent objects
- \( \text{Access}(i, j) \) is *the set of operations* that a process executing in Domain\(_i\) can invoke on Object\(_j\)
- Can be expanded to dynamic protection (operations to add, delete

<table>
<thead>
<tr>
<th>Domain</th>
<th>File1</th>
<th>File2</th>
<th>File3</th>
<th>File4</th>
<th>File5</th>
<th>File6</th>
<th>Printer1</th>
<th>Plotter2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Read</td>
<td>Read</td>
<td>Read</td>
<td>Read</td>
<td>Read</td>
<td>Read</td>
<td>Read</td>
<td>Read</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>Read</td>
<td>Read</td>
<td>Read</td>
<td>Read</td>
<td>Read</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table:**

- **Domain 1:**
  - File1: Read
  - File2: Read Write
  - File3: Read
  - File4: Read Write
  - File5: Read Write
  - File6: Read Write
  - Printer1: Write
  - Plotter2: Write

- **Domain 2:**
  - File1: Read
  - File2: Read Write
  - File3: Read Write
  - File4: Read Write
  - File5: Read Write
  - File6: Read Write
  - Printer1: Write
  - Plotter2: Write

- **Domain 3:**
  - File1: Read Write
  - File2: Read Write
  - File3: Read Write
  - File4: Read Write
  - File5: Read Write
  - File6: Read Write
  - Printer1: Write
  - Plotter2: Write
Example of Access Matrix

<table>
<thead>
<tr>
<th>Domain</th>
<th>File1</th>
<th>File2</th>
<th>File3</th>
<th>File4</th>
<th>File5</th>
<th>File6</th>
<th>Printer1</th>
<th>Plotter2</th>
<th>Domain1</th>
<th>Domain2</th>
<th>Domain3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Read</td>
<td>Read</td>
<td>Read</td>
<td>Read</td>
<td>Read</td>
<td>Write</td>
<td>Write</td>
<td>Write</td>
<td>Enter</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>Read</td>
<td>Write</td>
<td>Read</td>
<td></td>
<td>Write</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td>Read</td>
<td>Write</td>
<td>Write</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Implementing the Access Matrix

- The access matrix is usually large and sparse

- We can
  - store the matrix by columns or by rows
  - store only the non-empty elements

- Storing the matrix by columns corresponds to access control lists

- Storing the matrix by rows corresponds to capabilities
Access Control List (ACL)

- Associate with each object a list containing all the domains that may access the object, and how.
- Each column of the access matrix is captured in an access control list.
ACL (Cont.)

- To condense the length of the access control list, many systems recognize three classifications of users in connection with each file (e.g. Unix)
  - Owner
  - Group
  - Others

- Only three 3-bit fields are needed to define protection for each of these groups, for *read access*, *write access* and *execution control*

- More fine-grained access control lists can be specified for each file, if needed (e.g. Solaris 2.6 and beyond)
A sample directory listing in Unix

```
-rw-rw-r--  1 pbg  staff  31200  Sep 3 08:30  intro.ps
-dwrx------  5 pbg  staff  512  Jul 8 09:33  private/
-dwrxrwxr-x  2 pbg  staff  512  Jul 8 09:35  doc/
-dwrxrwx---  2 pbg  student  512  Aug 3 14:13  student-proj/
-rw-r--r--  1 pbg  staff  9423  Feb 24 2003  program.c
-rwxr-xr-x  1 pbg  staff  20471  Feb 24 2003  program
-dwxs-x-x-x  4 pbg  faculty  512  Jul 31 10:31  lib/
-dwxs------  3 pbg  staff  1024  Aug 29 06:52  mail/
-dwxsrwrx    3 pbg  staff  512  Jul 8 09:35  test/
```

- Above, “program” has the protection bits “r w x r - x r - x“
- The owner (pbg) can read, modify and execute “program”
- The members of the group (staff) can read and execute “program”
- All “other users” can also read and execute “program”
Domains in Unix

- In Unix, each user-id defines a separate domain

- By default, each process is executed in the domain of the user who invokes it

- Assume Mike wants to change his password
  - He will need to invoke a program such as “passwd”, which needs to have R/W rights for the file /etc/passwd
  - Will the “passwd” program run in Mike’s domain?
Domain Switching in Unix

- Domain switch accomplished via file system
  - A domain bit (setuid bit) is associated with each file
  - When the file is executed and setuid = on, then user-id is set to the owner of the file being executed. When the execution completes, user-id is reset.
  - If setuid=off, then the file is executed in the domain of the user who invokes it
Capabilities

- Associate with each domain a list of objects that may be accessed, and permitted operations.
- Each row of the access matrix is captured in a capability list.
- In practice, each capability can be seen as a ticket for an operation.
User Authentication

- Correctly identifying the users is crucial for system security.

- Authentication can be done based on:
  - User possession
  - User knowledge
  - User attribute

- Authentication using login name and password
  - Each user supplies a (login name, password)
  - If the login name is among the authorized users and the password matches with system records, it is accepted.
User Authentication (Cont.)

- Attacker must correctly enter the login name and the password.
  - Login name can be easily guessed
  - Password must be selected very carefully

- Several studies show that an unexpectedly large percentage of users (between 82% and 86%) use easily predictable passwords

- (Morris and Thompson, 1979; Klein, 1990; Kabay, 1997)
  - First and last names
  - Street/city names, vacation destinations
  - Words from a moderate-sized dictionary
  - SSN or license plate numbers
  - Abusive expressions, etc.
Authentication Using Passwords

- How an attacker broke into LBL in 1989
  - a U.S. Dept. of Energy research lab

LBL> telnet elxsi
ELXSI AT LBL
LOGIN: root
PASSWORD: root
INCORRECT PASSWORD, TRY AGAIN
LOGIN: guest
PASSWORD: guest
INCORRECT PASSWORD, TRY AGAIN
LOGIN: uucp
PASSWORD: uucp
WELCOME TO THE ELXSI COMPUTER AT LBL
Trivial Attack Scenario

- Attacker can connect to the target machine and try passwords from his/her dictionary.

- Many daemons break the underlying TCP connection after some number of unsuccessful login attempts in order to slow down attackers. Attacker can simply start many threads in parallel.

- Attacker can easily automate this process and run continuously over a broadband internet connection.

- Even scripts are available for free on the Internet for this purpose.
Who will guard the guardians?

- UNIX program called “login” authenticates users
  - Users enter their account name, password
  - Program checks password against password database
  - What could go wrong?

- Why would administrator trust login program?
  - Inspect source code, verify what it does
  - I.e., no ‘backdoors’ that allowed unexpected access
  - Is the program safe?

- NO. Trusted computing base includes compiler
  - Ken Thompson put backdoor in original UNIX login
  - Hacked the C compiler to hide his tracks
One-Time Passwords

- The password is different in each instance

- Commercial implementations use hardware calculators (SecurID).
  - Mostly in the shape of a credit card
  - Have a display and keypad
  - The user enters the shared secret (PIN)
  - The display shows one-time password
  - Example of two-factor authentication
Logic Bombs

- A piece of code written by one of a company’s programmers and secretly inserted into the production operating system/application program.

- As long as the programmer “feeds” it its daily password, nothing happens.

- If the programmer is fired or physically removed from the premises without warning, the logic bomb goes off (deleting/encrypting files, making hard-to-detect changes).
Login Spoofing

- Attacker writes a program to “emulate” the login screen of the terminal.

- When a user comes and enters his/her username and password, the program sends this info to the Attacker, prints “Invalid password”, sends a signal to kill its shell.

- This logs out Attacker and triggers the real login program.

- One way to guard against this attack is to have the login sequence start with a key combination that users program cannot catch.
Malware

- Installed malware can report the address of the captured machine back to certain *base machines*

- A *backdoor* is also installed on the machine that allow remote users to command the machine
  - To send out commercial spam
  - To send information about the typed keystrokes

- A machine taken over in this fashion is sometimes called a *zombie*

- A collection of them is called a *botnet*
Malware

- Malware has an entry point, see:
  - http://www.exploit-db.com/exploits/11787/

- Usually, a backdoor is also installed on the machine that allow remote users to command the machine
  - To send out commercial spam
  - To send information about the typed keystrokes

- A machine taken over in this fashion is sometimes called a zombie

- A collection of them is called a botnet
Trojan Horses

- A seemingly innocent program containing code to perform an unexpected and undesirable function (modify, delete, copy files).

- The person installing it first has to get the Trojan Horse executed.
  - Place the program on the Internet as a “free” utility.
  - Place the program in one of the directories heavily used.
Trojan Horses (Cont.)

- Scenario
  - In UNIX, the environment variable $PATH controls the directories that are searched for a command.

- ECHO $PATH
  :/usr/local/bin:/usr/bin:/bin:/usr/ucb:
  /usr/java/bin:/usr/bin/X11: /opt/util

- Attacker prepares a Trojan Horse and installs it in /usr/bin/X11 under the name ‘la’
Spyware Programs

- Spyware is a software that comes along with program the user has chosen to install (freeware, shareware or commercial programs).

- Spyware may
  - Download ads to display on the user’s system
  - Create pop-up browser windows when certain sites are visited
  - Change the user’s default home page, default media player, default search engine, bookmarks
  - Add new toolbars to the browser
  - Add new icons to the “desktop”
Worms and Viruses

- **Worms**: Processes that uses the spawn mechanism to clobber system performance.
  - A worm spawn copies of itself, using up system resources and network channels.

- **Viruses**: Fragments of code embedded in a legitimate program.
  - When executed, they may modify/destroy files or cause system crashes
Macro Viruses

- Macro viruses take advantage of a feature found in Microsoft Office applications such as Word or Excel.

- A macro is an executable program embedded in a word processing document or other type of file.
  - Users employ macros to automate repetitive tasks and thereby save keystrokes.
  - Macros are automatically executed on certain events (Opening/closing files, starting an application).
  - Macro viruses are easily spread through e-mail
Parasitic Viruses

- Parasitic virus attaches itself to executable files and replicates, when the infected program is executed -- by finding other executable files to infect.
Some Other Types of Viruses

- Memory-resident virus lodges in main memory as part of the resident system program. It infects every program that executes.

- Boot sector virus infects a boot record and spreads when the system is booted from the disk containing the virus.

- Encrypted virus includes the decryption code, along with the virus.

- Stealth virus is designed to avoid detection by modifying parts of the system.

- Polymorphic virus mutates with every infection, making detection by the “signature” of the virus very difficult or impossible.

- Compression/decompression is a frequently used technique by virus writers to avoid detection/disinfection.
Mutations of a Polymorphic Virus

- A piece of code that can mutate a sequence of machine instructions without changing its functionality is called a mutation engine.
- All five code segments above implement $X = A + B + C - 4$

\[
\begin{align*}
(a) & \quad \text{MOV A,R1} & \quad \text{MOV A,R1} & \quad \text{MOV A,R1} & \quad \text{MOV A,R1} & \quad \text{MOV A,R1} \\
(b) & \quad \text{ADD B,R1} & \quad \text{NOP} & \quad \text{ADD #0,R1} & \quad \text{OR R1,R1} & \quad \text{TST R1} \\
(c) & \quad \text{ADD C,R1} & \quad \text{ADD B,R1} & \quad \text{ADD B,R1} & \quad \text{ADD B,R1} & \quad \text{ADD C,R1} \\
(d) & \quad \text{SUB #4,R1} & \quad \text{OR R1,R1} & \quad \text{MOV R1,R5} & \quad \text{ADD C,R1} & \quad \text{ADD B,R1} \\
(e) & \quad \text{MOV R1,X} & \quad \text{ADD C,R1} & \quad \text{ADD C,R1} & \quad \text{ADD C,R1} & \quad \text{ADD B,R1} \\
(f) & \quad \text{NOP} & \quad \text{SHL #0,R1} & \quad \text{SHL R1,0} & \quad \text{SUB #4,R1} & \quad \text{CMP R2,R5} \\
(g) & \quad \text{SUB #4,R1} & \quad \text{SUB #4,R1} & \quad \text{SUB #4,R1} & \quad \text{SUB #4,R1} & \quad \text{SUB #4,R1} \\
(h) & \quad \text{NOP} & \quad \text{JMP .+1} & \quad \text{ADD R5,R5} & \quad \text{JMP .+1} & \quad \text{MOV R1,X} \\
(i) & \quad \text{MOV R1,X} & \quad \text{MOV R1,X} & \quad \text{MOV R1,X} & \quad \text{MOV R1,X} & \quad \text{MOV R1,X} \\
\end{align*}
\]
Covert Channels

- Lampson’s Confinement Problem
  - The client and server do not trust each other.
  - The collaborator and the server will try to co-operate to steal the client’s confidential data.

- How to make this system secure for the client
  - Use a protection matrix?
  - Forbid inter-process Communication?
The server can still leak info to collaborator via *covert channels*
- Modulate CPU Usage (heavy computation for 1, stop for 0)
- Modulate the number of page faults
- Locking/Unlocking a specific file
Covert Channels: Steganography

- Can secret information be passed even if the messages are subject to inspection?

- The picture on the left is the original one.
- The picture on the right has the text of 5 Shakespeare plays embedded in it.