Computer Security

- Protection of systems against an adversary
- **Secrecy:** Can’t *view* protected information
- **Integrity:** Can’t *modify* protected information or process
- **Availability:** Can’t deny access to system for other users
An adversary is someone who seeks an outcome detrimental to your interests

- An adversarial setting involves implicit or explicit conflict

An adversary is rational if he acts to maximize his payoff

- We usually assume an adversary is rational
Adversarial Setting

- **Example:** Games
- Structured adversarial setting
- Opposing objectives well-defined
Adversary or Attacker?

- An adversary becomes an attacker when he acts in a way that is detrimental to your interests

- Adversary is a term often used in cryptography

- Attacker is a term often used in computer security

- Technical distinction not hugely important
  - Both ultimately mean “bad guy”
Adversaries and Attackers

- Defined by motives and resources

- Motives:
  - Curiosity
  - Fame
  - Money
  - National interest

- Resources:
  - Time, money, and training
Classes of Attackers

From David Aucsmith, Microsoft.
Computer Security

- **Our definition**: Analysis and protection of computer systems in an adversarial setting
- **Traditional definition**: Protection of information against unauthorized access
Orange Book Definition

Fundamental Computer Security Requirements

Any discussion of computer security necessarily starts from a statement of requirements, i.e., what it really means to call a computer system "secure." In general, secure systems will control, through use of specific security features, access to information such that only properly authorized individuals, or processes operating on their behalf, will have access to read, write, create, or delete information. Six fundamental requirements are derived from this basic statement of objective: four deal with what needs to be provided to control access to information; and two deal with how one can obtain credible assurances that this is accomplished in a trusted computer system.
Computer Security

- Protection of systems against an adversary
- **Secrecy**: Can’t *view* protected information
- **Integrity**: Can’t *modify* protected information or process
- **Availability**: Can’t deny access to system for other users
Security specialists (e.g., Anderson [6]) have found it useful to place potential security violations in three categories.

1) Unauthorized information release: an unauthorized person is able to read and take advantage of information stored in the computer. This category of concern sometimes extends to “traffic analysis,” in which the intruder observes only the patterns of information use and from those patterns can infer some information content. It also includes unauthorized use of a proprietary program.

2) Unauthorized information modification: an unauthorized person is able to make changes in stored information—a form of sabotage. Note that this kind of violation does not require that the intruder see the information he has changed.

3) Unauthorized denial of use: an intruder can prevent an authorized user from referring to or modifying information, even though the intruder may not be able to refer to or modify the information. Causing a system “crash,” disrupting a scheduling algorithm, or firing a bullet into a computer are examples of denial of use. This is another form of sabotage.
Authorization

- What is authorized?
- Authorized: allowed by the operator of the system
- Clear when there is a central authority and explicit policy
  - Example: Department of Defense time-sharing systems
- Can be awkward to apply in some settings
  - Example: Click fraud malware on your smart phone
Computer Security

- Protection of systems against an adversary
- **Secrecy:** Can’t *view* protected information
- **Integrity:** Can’t *modify* protected information or process
- **Availability:** Can’t deny access to system for other users
Secrecy

- Prevent unauthorized access to information

- Also called confidentiality in the literature
  - However: some authors use confidentiality to mean an “obligation to protect secrets.” (Anderson Ch. 1)
  - I will use the term secrecy in this class

- What are some scenarios where you want secrecy?

- What are some scenarios involving computers?
Prevent unauthorized modification of information, process, or function

Example: Changing your bank account balance without depositing money

Example: Getting snacks from vending machine without paying for them

What are some other examples?
The focus of traditional computer security has been protection of information.

What about control or function of system?

- In a digital computer system everything is information.
- Why not just say integrity is “protection of information?”
Authenticity

❖ Prevent impersonation of another principal
❖ Some authors call this origin integrity
❖ Example: Getting money from someone else’s bank account using their credentials
❖ Does integrity include authenticity?
❖ What are some other examples?
Availability

- Prevent unauthorized denial of service to others
- **Example:** Physically rendering ATM unusable
- **Example:** Network denial of service attacks
- What are other examples of denial of service?
Examples

- Which security property is violated if someone ...
- ... unplugs your alarm clock while you’re sleeping?
- ... change the time on your alarm clock?
- ... Installs a camera in your room?
Privacy

- Privacy: A person’s right or expectation to control the disclosure of his/her personal information
- What is the difference between privacy and secrecy?
Security Policy

- **Security policy**: Set of allowed actions in a system
- **Security mechanism**: Part of system responsible for implementing the security policy
- **Security model**: Abstraction used by security mechanism
  - We will study several models in the next two lectures
  - Policy may be formulated using model
Policy, Mechanism, Model

Informal Policy → Formal Policy

Model

Mechanism
Assurance

- **Assurance**: Procedures ensuring that policy is enforced
- **Examples**: Testing, code audits, formal proofs
- Assurance is what justifies our *trust* in a system
Trust

- **Trust**: Belief that system or component will perform as expected or required
- **Trusted**: assumed to perform as expected or required
  - *Anderson*: component whose failure compromises security
- **Trustworthy**: will perform as expected or required
  - Some authors use *trustworthy* to mean that “there is sufficient credible evidence” that system or component will perform as expected or required
Trusted Computing Base

- **Trusted Computing Base (TCB):** Part of the system assumed to function as required
- Malfunction in TCB can lead to loss of protection
- Assurance gives us confidence of our trust of TCB
Trusted Computing Base

- In Unix: CPU, memory, boot disk, operating system kernel, operating system utilities (e.g. passwd)
- On a Web server?
- At a store?
- What assurance do we have for above?
  - How do we know TCB will work as required?
Incentives and Deterrents

❖ **Incentive:** Promise of reward for desirable action

❖ **Deterrent:** Threat of punishment that discourages undesirable action

❖ **Examples:**
  • UCSD punishes violation of academic integrity policy
  • Gun ownership believed to deter criminals
  • Reward for information about lost pet

❖ **Other examples?**
Incentives and Deterrents

- **Attacker’s equation:**
  \[(\text{expected gain}) > (\text{cost of attack})\]

- **Defender’s equation:**
  \[(\text{cost of protection}) < (\text{expected loss})\]
Incentives and Deterrents

❖ Attacker’s equation:
  (expected gain) > (cost of attack) + (expected punishment)

❖ Defender’s equation:
  (cost of protection) < (expected loss)
Accountability

- **Accountability**: Ability to attribute actions to individuals
- Why is accountability necessary?
- System must be amenable to forensic analysis after something goes wrong
- **Audit log**: record of all security-relevant events in system
  - Availability and integrity of audit log is critical
  - What about secrecy?
Security Model

- **Subjects**: Individuals or processes acting on their behalf
- **Objects**: Protected information or function
  - Objects often also include subjects
- **Subjects** operate on objects
  - System mediates and facilitates subject-object interaction
- **Policy**: what *action* is *subject* allowed to do with *object*?
  - And who can introduce new subjects and objects into system?
- Nearly all security models built on this idea
## Access Control Matrix

<table>
<thead>
<tr>
<th>Subjects</th>
<th>Objects</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>{allowed}</td>
</tr>
<tr>
<td></td>
<td>{actions}</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Access Control Matrix

<table>
<thead>
<tr>
<th></th>
<th>Broccoli</th>
<th>Fruit from Tree of Life</th>
<th>Fruit from Tree of Knowledge</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Adam</strong></td>
<td>{see, eat}</td>
<td>{see, eat}</td>
<td>{see}</td>
</tr>
<tr>
<td><strong>Eve</strong></td>
<td>{see, eat}</td>
<td>{see, eat}</td>
<td>{see}</td>
</tr>
</tbody>
</table>
Access Control Lists (ACLs)

- An access control list of an object identifies which subjects can access the object and what they are allowed to do.
- ACLs are object-centric: access control is associated with objects in the system.
- Each access to object is checked against object’s ACL.
- Example: guest list at a night club.
Capabilities

- A capability grants a subject permission to perform a certain action
  - Unforgeable
  - Usually transferrable

- Capabilities are subject-centric: access control is associated with subjects in the system

- Example: car key
ACLs & Capabilities

- Columns of the Access Control Matrix define objects’ ACLs
- Rows of the Access Control Matrix define users’ capabilities
Unix File System Sec. Model

- **Subjects**: Users
- **Objects**: Files and directories
- **Actions**: read, write, execute
  - Execute a file means can call `exec()` on file
  - Directory “execute” means user can traverse it
- Unix is a simplified ACL system
  - Arbitrary ACLs not possible in traditional Unix
  - Modern Unix operating systems allow arbitrary ACLs
Unix Superuser

- **Superuser** allowed do anything that is possible
- Called **root** and mapped to user id 0
- A superuser is a *role* rather than a particular user
- System administrators assume the superuser role to perform privileged actions
  - Good practice to assume superuser role only when necessary
Permissions

- Each file has an **owner** and a **group**
  - **Group**: named set of users

- File permissions specify what owner, group, and other (neither owner nor group) is allowed (read, write, exec)
Permissions

- Only owner and superuser can change permissions
- Only superuser can change owner
- Only owner and superuser can change group
  - Owner can only change to group she belongs to
- User’s allowed actions on file are:
  - Owner’s permissions if the user is the owner,
  - Group’s permissions if the user is in the group,
  - Other’s permissions otherwise
Permissions

- Users interact with system via processes acting on their behalf
- When you interact with system via terminal, command shell acts on your behalf
- Each process is associated with a user
Login

- When user connects to system via physical terminal, system runs `login` process as `root` to start session
  - Authenticates user using username and password
  - Changes its user id and group id to that of user
    - This is possible because superuser allowed to do anything
  - Executes user’s shell

- `sshd` performs similar actions

- **Critical**: dropping privileges from `root` to regular user
Changing Privilege

- Superuser can drop privilege to become regular user
- Want way to elevate privilege in controlled manner
- How?
Elevating Privilege

- Executable files have a setuid and setgid bit
- If setuid is set, files is executed with privilege of owner
  - ruid is that of executing user, euid and suid that of owner
- The setgid bit does same for group
  - But supplementary groups remain that of executing user
- The passwd command is setuid and owned by root
  - Executes as superuser (root) — why?
Unix Security Model

- What do you like about the Unix security model?
- What do you dislike about it?
- Is it a good model?