We said that in the most general terms, security seems to mean something like “protection of assets against attack.”

But this question is very specific to the context. Security for a wireless phone system may be very different from security for a military database system or an on-line banking system.
Often, security for a given system is defined in terms of a security policy, also sometimes called a security model.

The policy is the system specification wrt security. Another way to think of it is as a contract between the designer/implementor and the customer. That’s why it needs to be both achievable and adequate for the intended uses.

The policy defines what “security” means for a given system or family of systems. A policy may be characterized informally, semi-formally, or formally. It may be very abstract or very concrete.
Thought Experiment #1

Your academic records are stored on computers at the university. Design a security policy to protect them.

Start by asking: What does it mean “to protect them”? What are you protecting and what are the potential threats? Who are the stakeholders, i.e., whose interests are at risk? Do they conflict? Which of the following should you care about: confidentiality, integrity, availability?

www.utexas.edu/student/Registrar/catalogs/gi06-07/app/appc09.html outlines these rules for the university
I like to make the distinction between the *metapolicy* and the *policy*. This is not a distinction which is often drawn in the security literature, but I think it’s a very useful one.

**metapolicy:** The security goals in the most abstract sense.

**policy:** A system-specific refinement of the metapolicy adequate to provide guidance to developers and users of the system.

**Example:**

*Metapolicy:* social security numbers of students should be protected from disclosure.

*Policy:* faculty/staff may not use student SSNs in documents/files/postings; all older docs containing SSNs must be destroyed unless deemed necessary; documents deemed necessary must be kept in secure storage; etc.
Anyone developing a secure system might address (iteratively) the following questions:

1. What are you protecting and what are the potential threats? (risk assessment)
2. What is the intuitive notion of security for such a system? (metapolicy)
3. What are appropriate security rules that attempt to capture this notion for this system? (policy)
4. What is a system architecture that supports our security goals? (system design)
5. By what specific mechanisms might the security goals be accomplished? (detailed design)

Of course, there are lots of other questions that need to be addressed as well in any development.
Thought Experiment #2

Suppose you have several secure LAN’s that are geographically distributed, and must communicate securely over an insecure backbone network (the Internet).

Try to address the Security Design Process questions for this problem.
Thought Experiment #2

Suppose you have several secure LAN’s that are geographically distributed, and must communicate securely over an insecure backbone network (the Internet).

Try to address the Security Design Process questions for this problem.

**Caution:** Don’t jump too quickly to an implementation without considering what you’re trying to accomplish.

*If you think cryptography will solve your problem, then you don’t understand cryptography ... and you don’t understand your problem.* —Bruce Schneier
After you’ve gone through the design process for your system, you have to assess how well you’ve succeeded. There are some additional questions to be asked:

1. **Do the system design and implementation accomplish the security goals expressed by the policy?**
2. **How do you know? How certain are you of the assessment? What is the evidence?**
3. **Are there intuitively insecure behaviors that fall outside the range of the policy?**
4. **If so, does that mean that the policy doesn’t adequately capture the *metapolicy* or is the metapolicy incomplete?**
Extended Thought Experiment: MLS

In the early days of computer security, the canonical model was protection of *confidentiality* within a military setting: given information at various sensitivity levels and individuals having various degrees of trustworthiness, how do you control access to information within the system.

This problem is called *multi-level security* (MLS). Note that this problem predates computers. It’s an excellent problem for a thought experiment.

The initial formalization we’ll be considering was developed in 1973 by David Bell and Len LaPadula and is called the Bell and LaPadula model (BLP). It is one of the most influential efforts in the history of computer security.
Aside: MAC vs. DAC

Security systems should distinguish between:

**Mandatory Access Controls (MAC):** security rules that are enforced on every attempted access and not at the discretion of any system user;

**Discretionary Access Controls (DAC):** security rules that are enforced by the system at the discretion and behest of some users.

**Example:** the Unix file protection system implements DAC since the protections can be modified by the file owner.

For MLS, we’ll focus on mandatory controls. (Note that the acronymn “MAC” is used for several different notions in computing and security, so don’t get confused.)
In military systems, four models of operation are often defined for computers handling classified information:

**dedicated**: all users cleared for all information on machine; no need for access control (MILS);

**system-high**: all users cleared, but must obey need-to-know compartments (discretionary access control).

**compartmented**: all users cleared, but must be need-to-know compartments (mandatory access control). System must handle requests across classifications.

**multi-level**: not all users cleared for all information; system enforces access control (MLS).

MLS is the most difficult so not widely deployed.

Often a security solution/policy (access control) is phrased in terms of the following three categories:

**Objects:** the items being protected by the system (documents, files, directories, databases, transactions, etc.)

**Subjects:** entities (users, processes, etc.) that execute activities and request access to objects.

**Actions:** operations, primitive or complex, that can operate on objects and must be controlled.

For example, in the Unix operating system, processes (subjects) may have permission to perform read, write or execute (actions) on files (objects). In addition, processes can create other processes, create and delete files, etc. Certain processes (running with root permission) can do almost anything. That is one approach to the security problem.
Picture General Eisenhower’s office in 1943 Europe:

**The problem:** Assume an environment in which there are various pieces of *information* at different sensitivity levels: the war plan, the defense budget, the base softball schedule, the general’s laundry list, etc. Also, there are a variety of *individuals* with access to selected pieces of information: Eisenhower, Patton, privates, colonels, secretaries, janitors, spies, etc.

**The goal:** Understand what “security” might mean in this context and define some rules to implement it.

**Important proviso:** For this thought experiment we are only concerned with *confidentiality*, not integrity or availability. This will lead to some counterintuitive results.
What are we protecting? Against what threats?

Notice: it’s very important that we’re only considering confidentiality in this thought experiment. Someone burning down the office and destroying the war plan might be a significant threat, but it’s not a threat to confidentiality.
Recall the questions we asked about ensuring confidentiality:

1. How do you group and categorize information?
2. How do you characterize who is authorized to see what?
3. How are the permissions administered and checked? According to what rules?
4. How can authorizations change over time?
5. How do you control the flow of “permissions” in the system? Can I authorize others to view data that I am authorized to view?

For simplicity, let’s assume an environment of static permissions. That means we’ll ignore questions 4 and 5. Let’s see if we can figure out some possible answers for this specific setting to the other questions.
Back to our thought experiment: Gen. Eisenhower’s office in 1943. The relevant “space” of information contains lots of individual atoms or factoids:

1. The base softball team has a game tomorrow at 3pm.
2. The Normandy invasion is scheduled for June 6.
3. The cafeteria is serving chopped beef on toast today.
5. Col. Smith didn’t get a raise.
6. The British have broken the German Enigma codes.
7. and so on.

Not all information is created equal. How do we group and categorize information rationally?
Object Sensitivity Levels

Information is parcelled out into documents/folders/objects/files. Documents (objects) are labeled according to some authority’s assessment of their sensitivity level. *We’ll assume a certain form for labels; they might be done differently.*

One part of the label is taken from a linearly ordered set. One common scheme has levels: *Unclassified, Confidential, Secret, Top Secret.*

There are also “need-to-know” *categories*, from an unordered set, expressing membership within one or more interest groups, e.g., *Crypto, Nuclear, Janitorial, Embarrassing*, etc.

Some labels are special, but can be treated as need-to-know categories, e.g., *FOUO, No Foreign, Eyes Only.*
Ideally, the label on any document reflects the sensitivity of the information contained in that document. The label contains both a hierarchical component and a set of categories.

For example, two documents might have levels:

(Secret: \{Nuclear\}),
(Top Secret: \{Crypto\}).

One can expect that the first contains rather sensitive information related to the category Nuclear. This second contains highly sensitive information in category Crypto.

Some entity/agency/officer makes these labeling decisions. How they are made is outside the scope of our concern.
How do you label a document that contains “mixed information”?  
- Suppose the document contains both sensitive and non-sensitive information?  
- Suppose it contains information relating to both the Crypto and Nuclear domains?  

Sometimes a decision is made that a document classification should be changed. This is called **downgrading** (or **upgrading**).
Individuals (subjects) have *clearances* or *authorization levels* that are typically of the same form as document sensitivity levels.

That is, each individual has:

- a hierarchical security level indicating the degree of trustworthiness to which he or she has been vetted;
- a *set* of “need-to-know categories” indicating groups to which he or she belongs or areas of interest in which he or she is authorized to operate.
The lowest security level in the system is called \textit{system low}. For our MLS-type system it is (\texttt{Unclassified: \{ \}}).

Higher clearances are assigned by some organization or government entity according to their assessment of the individual’s trustworthiness and need for the information.

The highest (most permissive) level in the system, if it exists, is called \textit{system high}. What would be system high for our MLS system?

Some levels may be unpopulated, i.e., no individual is cleared at that level.
The need-to-know categories are a reflection that even within a given security level (such as Top Secret) there is plenty of information to which not everyone cleared to that level should have access. This is an instance of:

**Principle of Least Privilege:** Any subject should have access to the minimum amount of information needed to do its job.

This is as close to an axiom as anything in security. Why does it make sense?
Given that we have labels for objects and clearances for subjects, how do we decide which subjects are permitted access to which objects?

Surely it’s some relationship between the subject level and the object level. But what?

For example, should a subject with clearance \( \text{Secret: \{Crypto\}} \) be able to read a document labeled \( \text{Confidential: \{Crypto\}} \)?

Should a subject with clearance \( \text{Top Secret: \{Crypto, Nuclear\}} \) be able to modify a document labeled \( \text{Confidential: \{Crypto\}} \)?
The Dominates Relation

Given a set of security labels \((L, S)\), comprising hierarchical levels and categories, we can define a **partial order** among them.

**Definition:** \((L_1, S_1)\) *dominates* \((L_2, S_2)\) iff

1. \(L_1 \geq L_2\) in the ordering on levels, and
2. \(S_2 \subseteq S_1\).

We usually write \((L_1, S_1) \geq (L_2, S_2)\).

Note that this is *not* a total order. There are security labels \(A\) and \(B\), such that neither \(A \geq B\) nor \(B \geq A\).
A partial order is a relation that is reflexive, transitive, and antisymmetric.

- **Reflexive:** $x \geq x$
- **Transitive:** $[x \geq y \land y \geq z] \rightarrow x \geq z$
- **Anti-symmetric:** $[x \geq y \land y \geq x] \rightarrow x = y$

**Exercise:** Prove that dominates is a partial order.
Algebraically, the (full) set of labels with their ordering would form a \textit{lattice}. This is sometimes called “lattice-based security.”

In mathematics, a lattice is a partially ordered set (or poset), in which all nonempty finite subsets have both a supremum (join or lub) and an infimum (meet or glb). Lattices can also be characterized as algebraic structures that satisfy certain identities.

\textbf{Exercise:} Suppose you have two hierarchical levels $H$ and $L$ such that $L < H$, and two categories $A$ and $B$. Using the dominates relation as the partial order, draw the lattice of levels in this system. \textbf{How many levels are possible?}
Given our mechanisms for classifying objects (data / files) according to security labels, and personnel according to clearances, what are the answers to these questions?

*How do you group and categorize information?* The grouping is done as documents (files) and categorized according to labels.

What does that mean? Who assigns the labels? What about documents that contain “mixed” information?
How do you characterize who is authorized to see what?

The answer seems to be a relationship between the sensitivity level of a document (file) and the authorization level of the individual.

- What is the appropriate relationship?
- How do we codify it as a system of rules for access within this system?
- Does permission depend on the type of access requested? For example, are read and write access interchangeable?
Suppose subject S with authorization \((L_S, C_S)\) asks to read an object O with classification \((L_O, C_O)\). Under what conditions should the request be granted by the system?

For example, suppose a subject has clearance \((\text{Secret: } \{\text{Crypto}\})\). Which of the following should he be able to read?

- document labeled \((\text{Confidential: } \{\text{Crypto}\})\)
- document labeled \((\text{Top Secret: } \{\text{Crypto}\})\)
- document labeled \((\text{Secret: } \{\text{Nuclear}\})\)
- document labeled \((\text{Secret: } \{\text{Crypto, Nuclear}\})\)

So what is the formal rule?
According to the Bell-LaPadula Model (BLP) of security, one of the earliest formal security policies, the first formal rule governing access is:

**The Simple-Security Property:** Subject S with clearance \((L_S, C_S)\) may be granted read access to object O with classification \((L_O, C_O)\) only if \((L_S, C_S)\) dominates \((L_O, C_O)\).

We will often write “\((L_S, C_S)\) dominates \((L_O, C_O)\)” as “\((L_S, C_S) \geq (L_O, C_O)\),” but recall that it involves both hierarchical levels and need-to-know categories.
The Simple Security Property models read access in the world of military documents \textit{and} attempts to codify it for the world of electronic information storage.

\textbf{The Simple-Security Property:} Subject $S$ with clearance $(L_S, C_S)$ may be granted read access to object $O$ with classification $(L_O, C_O)$ only if $(L_S, C_S) \geq (L_O, C_O)$.

- Why is it “only if” and not “if and only if”?
- Does this work in an electronic context?
- Is it all that is needed? Why or why not?
The Simple-Security property codifies restrictions on *read* access to documents. What about *write* access?

Is the problem different with respect to writing in the electronic context than it is in the world of military documents? Why or why not?

More generally, what assumptions can be made about *persons* in the world of military paper documents that cannot be made about *subjects* (processes) in the context of computers?
“Sandy” Berger served as the National Security Advisor under President Bill Clinton from 1997 to 2001.

U.S. Justice Department prosecuted Berger for unauthorized removal and destruction of classified documents in October 2003 from a National Archives reading room prior to testifying before the 9/11 Commission, by stuffing them down his pants. The documents were five classified copies of a single report commissioned from Richard Clarke, covering internal assessments of the Clinton administration’s handling of the unsuccessful 2000 millennium attack plots. In April 2005, Berger pled guilty to a misdemeanor charge of unauthorized removal and retention of classified material from the National Archives in Washington.

Berger obviously had permission to read the documents. Why was he prosecuted?
Subjects in the world of military documents are assumed to be *persons* trusted not to disclose (write) to unauthorized parties information to which they have legitimate access.

Subjects in the world of computing are often *programs* operating on behalf of a trusted user (and with his or her clearance). The program may have embedded malicious logic (a “trojan horse”) that causes it to collude with other users or programs to “leak” information without the knowledge or consent of the authorized user.

For that reason, it is necessary to place mandatory controls on the write accesses of subjects that might not be necessary for persons. This is sometimes called the *confinement problem*.

What is the appropriate restriction on writing?
In the Bell-LaPadula Model of security, the following rule is enforced to restrict write access:

**The *-Property:** Subject $S$ with clearance $(L_S, C_S)$ may be granted write access to object $O$ with classification $(L_O, C_O)$ only if $(L_S, C_S) \leq (L_O, C_O)$.

This is pronounced “the star property.”
The *-Property

Does this rule make sense? Is it too restrictive? Is it too lax?

According to the *-property, can a commanding general with a top secret clearance email marching orders to a foot soldier? No!

According to the *-property, can a corporal with no clearance overwrite the war plan? Yes, but that’s an integrity problem!

The simple-security and *-property are sometimes characterized as “read down” and “write up,” respectively. Alternatively, they’re characterized as “no read up” and “no write down.”
Trusted Subjects

Often, to get around the more onerous restrictions of a mandatory policy, an implementation may add *trusted subjects*, specialized subjects permitted to operate “outside the rules of the policy” in very constrained ways.

**Example:** The *-property implies that the general can never send an email to the private. We add a special *downgrader* subject to the system and extend the *-property with the proviso that an object’s level can be reduced in specific ways only if the object’s contents are reviewed by the downgrader subject *including visual inspection by a trained human being*.

Notice: this technically violates the naive *-property, but prevents any malicious *program* from leaking information (unless it is clever enough to fool the downgrader). See Steganography.
Notice that Simple Security and the *-Property control two ways in which information can flow from A to B.

1. B can “pull” information from A by reading objects in A’s space. Simple Security is designed to constrain that type of information flow.

2. A can “push” information to B by writing objects in B’s space. The *-Property is designed to constrain that type of information flow.

Are there additional ways that information can flow from A to B that don’t involve either of those mechanisms.
Our discussion of the Bell and LaPadula model explicitly included Read and Write access, but not Create, Destroy, Execute, Append, others. How might we add these operations to our BLP framework?

In particular, is Execute effectively a modify (write) operation? A reference (read) operation? Neither? Both?
Our discussion of the Bell and LaPadula model explicitly included \textbf{Read} and \textbf{Write} access, but not \textbf{Create}, \textbf{Destroy}, \textbf{Execute}, \textbf{Append}, others. How might we add these operations to our BLP framework?

In particular, is \textbf{Execute} effectively a modify (write) operation? A reference (read) operation? Neither? Both?

Maybe that’s the wrong way to think about execute. Maybe it \textit{creates a subject} with the creator’s permission levels. Then, aren’t Simple Security and the *-Property adequate?
According to BLP, security is essentially defined as follows:

**Definition:** A system is *secure* if it always satisfies the simple security condition and the *-property.

Bell and LaPadula proved a theorem about a formalization of their model that they considered to be very important.

**The Basic Security Theorem:** Let $\Sigma$ be a system with a secure initial state $\sigma_0$, and let $T$ be a set of state transitions. If every element of $T$ preserves the simple security condition and the *-property, then every $\sigma_i, i \geq 0$, is secure.

The proof is a simple induction over $i$. 
John McLean (NRL) pointed out that the Basic Security Theorem isn’t very useful, because it says what is true in the *states* of the system, but doesn’t constrain *transitions* that may occur in the system.

Consider a system (System Z) in which any attempt to read a file causes all objects and subjects in the system to be downgraded to security level system-low. Notice that this could be argued to satisfy weak tranquility.

The Basic Security Theorem can still be proved for this system but it is obviously insecure.
McLean argued that reasoning merely about *states* isn’t adequate. It is also necessary to reason about *transitions*.

Bell responded that McLean had misunderstood the nature of the model. The model is only a formalism that provides a framework for reasoning about secure systems. It doesn’t provide a definition of security.

**The Lesson:** Formal definitions and theorems don’t guarantee anything unless they are validated against reality. Any *interpretation* of the formalism is as valid as any other. This controversy raised questions about the “foundations” of computer security research.
An obvious hole in the BLP model would be the ability of a subject to change its own security level or that of an object under its control. One could add either:

**The Strong Tranquility Property:** Subjects and objects do not change levels during the lifetime of the system.

**The Weak Tranquility Property:** Subjects and objects do not change levels in a way that violates the “spirit” of the security policy.

Is this useful? Is it overly restrictive? What if a user needs to operate at different levels during the course of the day?
The Weak Tranquility Property: Subjects and objects do not change levels *in a way that violates the “spirit” of the security policy.*

What does this mean?

- Suppose your system includes a command to *lower* the level of a subject/object. Does that violate the goals of simple security or the *-property?  

- Suppose your system includes a command to *raise* the level of a subject/object. Does that violate the goals of simple security or the *-property?
The Bell and LaPadula Model in its original incarnation was somewhat more complex, but a thumbnail version is simply:

- Simple Security Property
- the \(*\)-Property
- some version of the Tranquility Property.
Are simple-security, the \*-property, and the tranquility property adequate to ensure confidentiality within the system?

What about the following issues:

- What about other types of operations? Can every operation be thought of as a *read* or *write*? Can some be both?
- What useful operations can you imagine that might subvert the protections offered?
- Our restrictions control access by subjects to objects. Are there ways in which information might be compromised without explicit read or write operations?
The Bell and LaPadula Model is an example of an Access Control Policy. This is a popular way of conceptualizing and implementing security.

The basic idea is to introduce rules that control what accesses system *subjects* have to system *objects*.

This is an important aspect of security. The problem is that there may be information channels in the system that don’t involve the access of subjects to objects.
For any secure system, we have to consider the following different areas of concern:

**Policy:** What is the notion of security that is being enforced by the system?

**Mechanism:** How is that policy enforced in the system?

**Assurance:** How certain can we be that the policy is enforced by the mechanisms we have put in place?

The Bell and LaPadula rules are somewhat ambiguous about whether they constitute a policy or a mechanism. They must be enforced by various different mechanisms in real systems. The level of assurance is a measure of the care and rigor with which the system is evaluated with respect to the policy. Sometimes it is difficult to judge.
An Aside: Firewalls

What is a firewall? Essentially, it’s just an access control mechanism applied by structuring the system in a particular way.

Your first programming assignment involves integrating the access control checks tightly into the semantics of the operations READ and WRITE. Assume instead that your system is modeled as a server receiving commands from outside. Now suppose you applied your access control checks at the system boundary before accepting any command. Note that this may involve a separate processor.

Then the semantics of the individual operations could be simpler because any request would be guaranteed to be legal. That is the basic idea of a firewall.
Some Questions

1. Can our system satisfy BLP’s security properties and still be intuitively non-secure?

2. Are there ways in which a high level subject could pass information to a low level subject without violating our security property?

3. Are there other instructions that we might naturally want to include in our system? Eg., suppose we add a CREATE instruction that allows a subject to create a new object. What constraints should be placed on their operation?

4. How should we handle exceptions? Eg., what should happen if ill-formed instructions are included in the instruction stream?

5. Is there a “stronger” security policy that we might apply?
Remember our distinction between *policy* and *metapolicy*. The metapolicy gives us a handle for tackling such questions.

The real security goal (metapolicy) of any MLS scheme is to control the flow of information in the system. I.e., sensitive information should not flow “down” in the system, from a high level to a low level.

More precisely, information must flow through the lattice only along upward channels. *That* is the metapolicy that drives and justifies the access control rules.

Is BLP adequate to ensure this metapolicy? What would a counterexample look like? What would it mean?
Consider an MLS system that has READ and WRITE operations that follow the BLP rules. Just to be concrete, suppose we define the semantics as follows:

**READ subj name obj name:** if object exists and subject has read access to it, return its current value; otherwise, return a zero.

**WRITE subj name obj name value:** if object exists and the subject has write access to it, change its current value to **value**; otherwise, do nothing.

Ordinarily, the subject would be an *implicit* parameter to the operation; we’re just making it explicit to simplify matters.
Now, suppose we want to add the following operations to our simple secure system:

```plaintext
CREATE subject_name object_name
DESTROY subject_name object_name
```

Under what conditions would these operations be secure? I.e., what should be the semantics of these operations, if they are not to violate our intuitive notions of confidentiality?
Now, suppose we want to add the following operations to our simple secure system:

CREATE subject_name object_name
DESTROY subject_name object_name

Under what conditions would these operations be secure? I.e., what should be the semantics of these operations, if they are not to violate our intuitive notions of confidentiality?

What is the level of a created object? What if an object by that name already exists? What if we try to destroy an object that doesn’t exist?
Suppose we define these operations as follows:

**CREATE:** if no object with name `obj_name` exists, create a new object at the subject’s level; otherwise, do nothing.

**DESTROY:** if an object with name `obj_name` exists and the subject has write access to it, destroy it; otherwise, do nothing.

These rules seem to satisfy BLP, but are they “secure” from the standard of the metapolicy? Why or why not?
In this system, a high level subject $H$ can signal one bit of information to a low level subject $L$ as follows:

<table>
<thead>
<tr>
<th>H Signals 0</th>
<th>H Signals 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>H: Create F0</td>
<td>H: <em>do nothing</em></td>
</tr>
<tr>
<td>L: Create F0</td>
<td>L: Create F0</td>
</tr>
<tr>
<td>L: Write F0, 37</td>
<td>L: Write F0, 37</td>
</tr>
<tr>
<td>L: Read F0</td>
<td>L: Read F0</td>
</tr>
<tr>
<td>L: Destroy F0</td>
<td>L: Destroy F0</td>
</tr>
</tbody>
</table>

In the first case, $L$ sees a value of 0; in the second case, $L$ sees a value of 37. We assume that they have agree in advance which result is to be interpreted as a 0 and which as a 1. Assuming that the two can coordinate their activities, they can repeat this over and over.

$L$ must always do the same things in both columns (at least until it gets the value). Why?
A flaw in the Bell and LaPadula scheme, and indeed in any access control scheme, is that it only controls the flow of information via objects that are explicitly recognized by the security policy as carrying information.

But information can be carried in other ways as well. Such information paths are called covert channels.
Some sources define a covert channel as any channel in violation of the security policy; that’s too broad to be useful. A better definition is:

**Definition:** A *covert channel* is a path for the flow of information between subjects within a system, utilizing system resources that were not designed to be used for inter-subject communication.

Where did the bit of information reside in the previous channel?
Covert Channel

Note the important features of this definition:

- Information flows from one subject to another, presumably in violation of the security metapolicy *though not necessarily in violation of the policy*.

- The flow is within the system (two human users talking over coffee is not a covert channel).

- The flow occurs via system resources (file attributes, flags, clocks, etc.) that were not intended as communication channels.

A system can satisfy an access control policy (such as BLP) and still contain multiple covert channels.
It is possible to distinguish the following types of covert channels:

**Implicit flows:** signal information through the control structure of the program.

**Termination channels:** signal information through termination or non-termination of a computation.

**Timing channels:** signal via the amount of time a computation takes.

**Probabilistic channels:** signal by changing the probability distribution of observable data.

**Resource exhaustion channels:** signal via possible exhaustion of a finite shared resource, such as memory or disk space.

**Power channels:** embed information in the power consumed (useful for smartcards where the energy is supplied by the host computer).

In practice, most researchers distinguish only *storage* and *timing* channels.
Process $p$ cannot communicate with process $q$ directly. However, $p$ can create and delete files in a directory. $q$ cannot read or modify files in the directory, but can list them. To send a bit of information, process $p$ deletes any file named *bit*, and then creates a file called either 0bit or 1bit in the directory. Process $q$ detects it. This repeats until the message has been delivered.

This is a classic storage covert channel.

**Note:** If $q$ could read files in the directory that wouldn’t be a covert channel. Also, why doesn’t $p$ just name his file the-attack-is-at-dawn for higher bandwidth? Would that work?
Sample Covert Channel 2

The KVM/370 operating system isolated processes on separate virtual machines. They shared the processor on a time-sliced basis. Processes alternated using the CPU, with each allowed $t$ units of processing time. However, a process could relinquish the CPU early.

Process $p$ could send a bit to process $q$ by either using its total allocation or relinquishing the processor immediately. Process $q$ reads the bit by consulting the system clock to see how much time has elapsed since it was last scheduled.

This is a classic timing covert channel.
Suppose two processes share a disk. Process $p$ either accesses cylinder 140 or 160. Process $q$ requests accesses on cylinders 139 and 161. The scanning algorithm services requests in the order of which cylinder is currently closest to the read head. Thus, $q$ receives values from 139 and then 161, or from 161 and then 139, depending on $p$’s most recent read.

Is this a timing or storage channel? Neither? Both?
An implicit channel is one that uses the control flow of a program. For example, consider the following program fragment:

```plaintext
h := h mod 2;
l := 0;
if h = 1 then l := 1 else skip;
```

The resulting value of \( l \) depends on the value of \( h \).

There are sophisticated *language-based information flow tools* that check for these kinds of dependencies in programming languages.
It might seem that these covert channels would be so slow that you wouldn’t really care.

That’s not true. Covert channels on real processors operate at thousands of bits per second with no appreciable impact on system processing.

(In fact, in your second programming assignment, you will implement a covert channel and estimate its bandwidth.)
The two important attributes of covert channels are *existence* and *bandwidth*.

It is usually infeasible for realistic systems to eliminate every potential covert channel. However it is important to identify those that can be used to advantage and to close them or restrict them in such a way that the bandwidth is reduced to a negligible amount.
A characteristic of any communication channel that affects bandwidth is whether it is noiseless or noisy. Information theory provides a very precise definition; the following is an intuitive approximation.

**Definition:** A noiseless channel is one where the message can be transmitted without distortion or loss of information. A noisy channel is one where there is distortion or loss of information.

For covert channels, a noiseless channel might be one where the shared resource is only available to the two colluding parties. A noisy channel might be one where there are other users potentially accessing the resource.
Dealing with Covert Channels

Once a potential covert channel is identified, several responses are possible.

- We can eliminate it by modifying the system implementation.
- We can reduce the bandwidth by introducing noise into the channel.
- We can monitor it for patterns of usage that indicate someone is trying to exploit it. This is *intrusion detection*.

The solution could introduce other problems. For example, one might eliminate a covert channel on a shared resource by always giving priority to the low process (and possibly terminating the high process). This obviously introduces a denial of service vulnerability.
Dealing with Covert Channels

In the early 1990’s the U.S. Government published guidelines for covert channels in secure systems they certified:

“Covert storage channels shall be treated as follows:

1. There shall be no covert storage channels with a capacity exceeding 100 bits/second;
2. All covert storage channels with capacities exceeding 10 bits/second shall be auditable;
3. All covert storage channels with capacities exceeding 1 bit/second shall be described in the product’s covert channel analysis.”

These numbers are hopelessly out of date, but note that this presumes that it is possible to find all covert channels in the system. How might you do that?
For a sender and receiver to use a covert *storage* channel, what must be true?
For a sender and receiver to use a covert storage channel, what must be true?

1. Both sender and receiver must have access to some attribute of a shared object.
2. The sender must be able to modify the attribute.
3. The receiver must be able to reference (view) that attribute.
4. A mechanism for initiating both processes, and sequencing their accesses to the shared resource, must exist.
For a sender and receiver to use a covert *timing* channel, the following must be true:

1. Both sender and receiver must have access to some attribute of a shared object.
2. Both sender and receiver have access to a time reference (real-time clock, timer, ordering of events).
3. The sender must be able to control the timing of the detection of a change in the attribute of the receiver.
4. A mechanism for initiating both processes, and sequencing their accesses to the shared resource, must exist.
Richard Kemmerer introduced the Shared Resource Matrix Methodology (SRMM). The idea is to build a table for each command and its potential effect on shared attributes of objects.

<table>
<thead>
<tr>
<th></th>
<th>readFile</th>
<th>writeFile</th>
<th>deleteFile</th>
<th>createFile</th>
</tr>
</thead>
<tbody>
<tr>
<td>file existence</td>
<td>R</td>
<td>R</td>
<td>R, M</td>
<td>R, M</td>
</tr>
<tr>
<td>file owner</td>
<td>R, M</td>
<td>M</td>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>file name</td>
<td>R</td>
<td>R</td>
<td>R, M</td>
<td>M</td>
</tr>
<tr>
<td>file size</td>
<td>R</td>
<td>M</td>
<td>M</td>
<td>M</td>
</tr>
</tbody>
</table>

An R in the matrix means the operation References (provides information about) the attribute under some possible circumstances. An M means the operation Modifies (affects the value of) the attribute under some possible circumstances.
Suppose you have the following operation:

**CREATE:** if no object with name `obj_name` exists, create a new object at the subject’s level; otherwise, do nothing.

For the attribute *file existence*, should you have an R or not for this operation? Consider this: you *know* that the file exists after this operation. Why?

But that’s not enough. It’s not important that you *know* something about the attribute; what’s important is that the operation *tells* you something about the attribute. A low-level process couldn’t use CREATE to get the information it would need to carry out its part of a covert channel.
Working with the SRMM

The only resources/attributes that are *potential* channels are those with both R and M in a row. Why?

Building the matrix requires detailed knowledge of the system architecture.

The SRMM doesn’t identify shared resources, but suggests which might be used as covert channels.

Any shared resource matrix is *for a specific system*. Other systems may have different semantics for the operations.
Using the SRMM

**Build a Shared Resource Matrix for each of the covert channel examples on the previous slides.**

**Channel 1:** Process $p$ cannot communicate with process $q$ directly. However, both can list files in a common directory. Process $p$ creates a file called either $0bit$ or $1bit$ in the directory. Process $q$ detects it and deletes it. This repeats until the message has been delivered.

<table>
<thead>
<tr>
<th></th>
<th>Read</th>
<th>Write</th>
<th>Create</th>
<th>Delete</th>
<th>ListFiles</th>
</tr>
</thead>
<tbody>
<tr>
<td>file existence</td>
<td>R</td>
<td>R</td>
<td>R, M</td>
<td>R, M</td>
<td>R</td>
</tr>
<tr>
<td>file label</td>
<td>R</td>
<td>R</td>
<td>R, M</td>
<td>M</td>
<td>R</td>
</tr>
</tbody>
</table>

*Now try a similar exercise for samples 2 and 3.*
One approach to secure system design is to use an access control security model like Bell and LaPadula, and then to use a separate technique (such as SRMM) to find and close covert channels.

The question arises, is it possible to define a security model that is strong enough to cover access control and covert channels. That is one goal of non-interference models.
An access control policy says who may access information, but not how that data is used after it is acquired.

To ensure confidentiality using an access control policy, it is necessary to grant access only to subjects that will not improperly transmit or leak the data—but that requires a stronger “information flow” policy.

That is, an access control policy such as BLP tries to assign accountability up front, by assigning a level. But if the process is not trustworthy, access control is not enough.
An alternative to access control policies is a class of policies called *information flow* policies. The best known is *non-interference*.

The policy of the system is a binary relation \((a \rightarrow b)\) over the subjects of the system that says which subjects are permitted to “interfere with” which other subjects.

You can think of “can interfere with” as meaning “can communicate to” or “can direct information to.” In the types of systems we have been discussing, \((a \rightarrow b)\) means that \(a\) can write into \(b\)’s view, or \(b\) can read from \(a\)’s view. But there is no distinction between these two.
It is possible to take any MLS policy and turn it into a non-interference policy.

Consider a BLP system with three subject’s:

- A at \( \text{Secret: } \{ \text{Crypto, Nuclear} \} \),
- B at \( \text{Secret: } \{ \text{Crypto} \} \), and
- C at \( \text{Unclass: } \{ \} \).

What is the corresponding NI policy? Suppose you add D at \( \text{Top Secret: } \{ \text{Crypto, Nuclear} \} \)?

In general, given a BLP system, how do you compute the corresponding NI policy?
Intuitively, the idea of non-interference is that a low-level user’s “view” of the system should not be affected by *anything* that a high-level user does.

*Though strictly speaking, talk of “high” and “low” here is misleading. There is only a notion of who is allowed to interfere with whom.*
Recall that we considered the following different areas of concern: policy, mechanism, and assurance.

Non-interference is another policy, more abstract than BLP. The enforcement mechanisms may be anything, including the BLP rules. In this context, enhancing our level of assurance could mean formulating and proving a theorem about the system.

The policy of the system is a binary relation \((a \mapsto b)\) over the subjects of the system that says which subjects are permitted to “interfere with” which other subjects. What would this look like for a BLP system?
One way to formalize non-interference is as follows. Suppose \( L \) is a subject in the system. Now suppose you:

1. run the system normally, interleaving the operations of all users;
2. run the system again after deleting all operations requested by subjects which should not be able to pass information to (interfere with) \( L \).

From \( L \)’s point of view, there should be *no visible difference*. The system is *non-interference secure* if this is true of *every* subject in the system.
A possible correctness theorem is something like:

$$\forall s \in \text{Subjects}, \forall S_0 \in \text{States}, \forall I \in \text{InstructionList},$$
$$\text{view}(s, \text{run}(I, S_0)) = \text{view}(s, \text{run}(\text{purge}(I, s), S_0))$$

Here \textit{run} is our execution of instruction sequence \textit{I} from initial state \textit{S_0}, and \textit{purge} removes from the instruction list any instructions by subjects that should not be allowed to interfere with \textit{s}.

Note: this assumes that the system is \textit{deterministic} and that the policy is \textit{transitive}, though it is possible to formulate a related theorem for a non-deterministic system and intransitive policy.
The policy can be made as strong as you like by characterizing “view.” The more things that you consider to be within the view of the user, the stronger the policy.

For example, if you include within a subject’s view the values of system flags, then they could not be used in a covert channel. If you include the system clock, then you could not use that in a covert channel.
Unwinding Theorem

Note that the non-interference theorem refers to all subjects, all states, and all instruction sequences and requires an induction that touches every reachable state of the system. This may seem very difficult to carry out.

However, it is possible to prove an unwinding theorem that localizes the reasoning. That is, we identify an invariant on the system state. If each instruction “locally” preserves the invariant, then the correctness theorem follows automatically. This means that when we add a new instruction, it is only necessary to prove that the new instruction preserves the invariant.
Unwinding Theorem

It happens that the global non-interference property follows from two *local* properties. Let \( (a \rightarrow b) \) mean that “a can permissibly interfere with b” and let \( i_a \) denote an instruction executed on behalf of \( a \).

The system *locally respects* the interference relation iff:

\[
\forall a, b \in Subjects, \forall s \in States, \forall i_a \in Instruction, \\
\neg (a \rightarrow b) \Rightarrow \text{view}(b, \text{step}(i_a, s)) = \text{view}(b, s)
\]

The system is *step consistent* iff:

\[
\forall a, b \in Subjects, \forall s_1, s_2 \in States, \forall i_a \in Instruction, \\
\text{view}(b, s_1) = \text{view}(b, s_2) \\
\Rightarrow \text{view}(b, \text{step}(i_a, s_1)) = \text{view}(b, \text{step}(i_a, s_2))
\]
The previous idea of non-interference assumes that the “interferes” relation is transitive and possibilistic and that the system is deterministic. It is possible to define related notions that allow for an intransitive interference relation, probabilistic notion of security, and non-deterministic systems.

What do these mean? Can you intuit what changes need to be made for these various different system models?
How does non-interference address the problem of covert channels?

**Answer:** By adding to a subject’s view elements of the system state other than files, it makes visible changes in those elements that might convey information.

Note that this is a very powerful approach, but it has some limitations. Ideally, a non-interference approach requires no separate covert channel analysis. (See ref. 389, for example)
Non-interference is very difficult to achieve for realistic systems.

- It requires identifying within the view function all potential channels of information.
- Realistic systems have many such channels.
- Modeling must be at very low level to capture many such channels.
- Dealing with timing channels is possible, but difficult.
- Very few systems are completely deterministic.
- Some “interferences” are benign, e.g., encrypted files.