CS329E: Elements of Security
Covert Channels

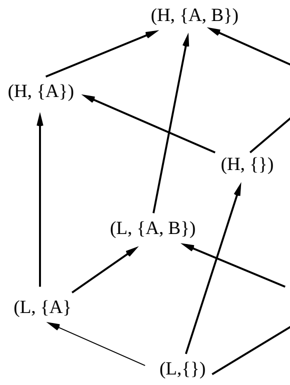
Dr. Bill Young
Department of Computer Sciences
University of Texas at Austin

Last updated: February 16, 2015 at 11:41

The BLP Metapolicy

Remember our distinction between *policy* and *metapolicy*. The metapolicy gives us a handle for tackling such questions.

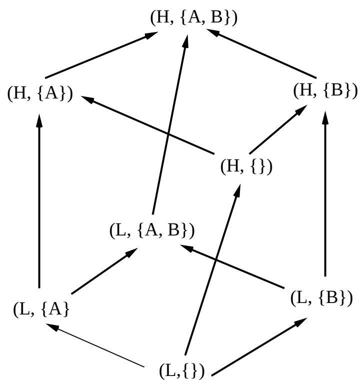
The real security goal (metapolicy) of any MLS scheme is to *control the flow of information* in the system. I.e., sensitive information should not flow "down" in the system, from a high level to a low level.



The BLP Metapolicy

To be precise, information in a BLP system 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?



A Simple BLP System

Consider an MLS system that has READ and WRITE operations that follow the BLP rules. Just to be concrete, suppose we define the *semantics* (meaning) of the rules as follows:

READ subj_name obj_name: if object exists and subject has read access to it, return its current value to the subject; 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.

A BLP System (Cont.)

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

A BLP System (Cont.)

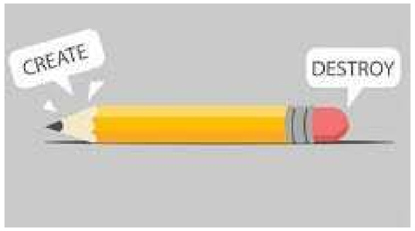
Now, suppose we want to add the following operations to our simple secure system:

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

Under what conditions would these operations be BLP 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?

A BLP System (Cont.)



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" by the standards of the metapolicy? Why or why not?

Covert Channel Example

Consider the following program:

H Signals 0	**H Signals 1**
H: Create F0	H: *do nothing*

L: Create F0
L: Write F0, 1
L: Read F0
L: Destroy F0

That is, H performs one of two actions, but L always does the same thing.
What does L see as a result of the its Read operation in the two cases?

Covert Channel Example

H Signals 0	**H Signals 1**
H: Create F0	H: *do nothing*

L: Create F0
L: Write F0, 1
L: Read F0
L: Destroy F0

In one case, L sees a value of 0; in the second case, L sees a value of 1. But, so what?

Covert Channel Example

H Signals 0	**H Signals 1**
H: Create F0	H: *do nothing*

L: Create F0
L: Write F0, 1
L: Read F0
L: Destroy F0

In one case, L sees a value of 0; in the second case, L sees a value of 1. But, so what?

Using this mechanism, H can send one bit of information to L. If they can repeat this multiple times, then H can send any amount of information to L.

Covert Channel Questions

1. Must H and L work together for this channel to work?
2. Must H and L interleave their actions?
3. Must L see only 0 or 1, as in this case? Could it have been different values?
4. H varies his action, but L always does the same thing. Is that important?
5. Does one bit of information really matter?

Covert Channel Answers

1. Yes. L has to be "on the lookout" for the bit H is sending and know how to read and store it.
2. Yes. It can't really work otherwise.
3. No. As long as the results differ, L can interpret them as 0's and 1's.
4. Very. L doesn't know what bit H is sending, so couldn't do different things in the two cases.
5. Probably not. But if you do this in a loop, you can repeat it thousands of times. *Any information can be encoded as a sequence of bits.*

When students implement this channel in a simple BLP system (in my security class for majors) they can transmit all of Moby Dick in a few seconds.

Covert Channels

The weakness with Bell and LaPadula, as in most access control schemes, 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*.

Covert Channel

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

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?

It wasn't stored as the contents of F0! F0 contained a 1 in both cases. It was "stored" in the level of F0 and the ability of L to read it.

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.

Sample Covert Channel 1

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 *they wouldn't need a covert channel*. Also, why doesn't p just encode his message in the filename (e.g., *the-attack-is-at-dawn*) for higher bandwidth? Would that work?

Sample Covert Channel 2

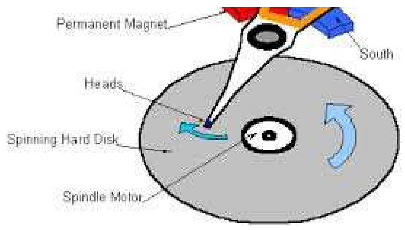
The KVM/370 operating system isolated processes on separate virtual machines. They shared the processor on a time-sliced basis. Processes alternated using the CPU, with each allowed t units of processing time. However, a process could relinquish the CPU early.

Process p could send a bit to process q by either using its total allocation or reliquishing 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*.

Sample Covert Channel 3

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?

Sample Covert Channel 4

An implicit channel is one that uses the control flow of a program. For example, consider the following program fragment:

```
high := high mod 2;
low := 0;
if high = 1 then low := 1 else skip;
```

The resulting value of `low` depends on the value of `high`.

There are sophisticated *language-based information flow tools* that check for these kinds of dependencies in programming languages.

Taxonomy of 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.

Covert Channels: Who Cares

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.

Covert Channels

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.

Noisy vs. Noiseless Channels

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?

Using a Covert Storage Channel

For a sender and receiver to use a covert *storage* channel, what must be true?

Using a Covert Storage Channel

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.

Using a Covertg Channel

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.

Detecting Covert Channels

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.

	readFile	**writeFile**	**deleteFile**	**createFile**
file existence	R	R	R, M	R, M
file owner			R, M	M
file name	R	R	R, M	M
file size	R	M	M	M

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 *under some possible circumstances*.

A Subtlety of SRMM

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.

	Read	**Write**	**Create**	**Delete**	**ListFiles**
file existence	R	R	R, M	R, M	R
file label	R	R	R, M	M	R

Which R and M manifest the channel? Now try a similar exercise for samples 2 and 3.

Covert Channels and System Analysis

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?

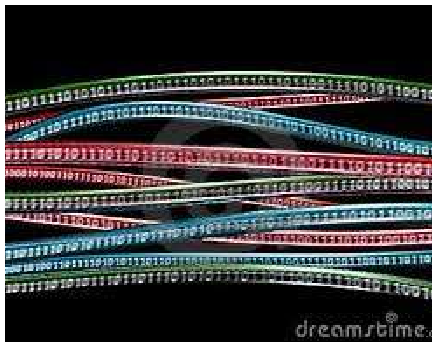
Limits of Access Control

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.

Information Flow Policies



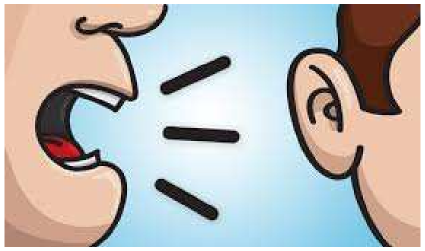
An alternative to access control policies is a class of policies called *information flow* policies.

If what we really care about (e.g. in military security) is where information can flow, why not sa that directly instead of talking about read and write access?

Non-Interference

The best known information flow policy is *non-interference*.

The *non-interference 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.



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, ($a \mapsto b$) 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.

Non-Interference (Cont.)

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

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

In general, given a BLP system, how do you compute the corresponding NI policy?

Non-Interference (Cont.)

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.

Levels of Concern

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?

Non-Interference

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.

Non-Interference (Cont.)

The policy can be made as strong as you like by characterizing "point of 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 timing channel.

Non-Interference

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.

Limitations of Non-Interference

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.