Access Control and Privacy Policies (5)

Email: christian.urban at kcl.ac.uk
Office: S1.27 (1st floor Strand Building)
Slides: KEATS (also homework is there)

Satan's Computer

Ross Anderson and Roger Needham wrote:

> In effect, our task is to program a computer which gives answers which are subtly and maliciously wrong at the most inconvenient possible moment... we hope that the lessons learned from programming Satan's computer may be helpful in tackling the more common problem of programming Murphy's.

Protocol Specifications

The Needham-Schroeder Protocol:

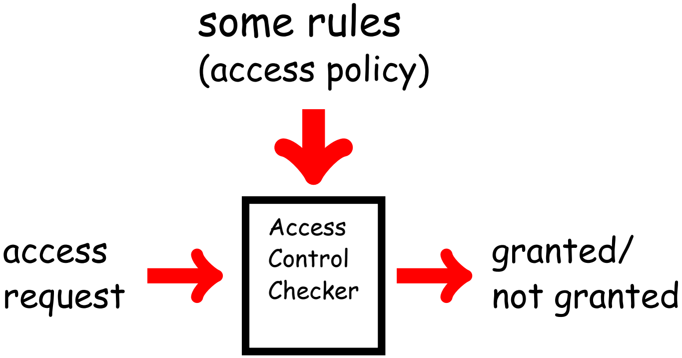
Message 1 $A \rightarrow S : A, B, N_A$
Message 2 $S \rightarrow A : \{N_A, B, K_{AB}, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$
Message 3 $A \rightarrow B : \{K_{AB}, A\}_{K_{BS}}$
Message 4 $B \rightarrow A : \{N_B\}_{K_{AB}}$
Message 5 $A \rightarrow B : \{N_B - 1\}_{K_{AB}}$

Cryptographic Protocol Failures

Again Ross Anderson and Roger Needham wrote:

> A lot of the recorded frauds were the result of this kind of blunder, or from management negligence pure and simple. However, there have been a significant number of cases where the designers protected the right things, used cryptographic algorithms which were not broken, and yet found that their systems were still successfully attacked.

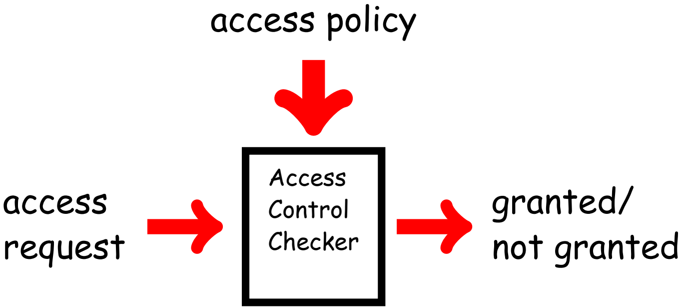
The Access Control Problem



Access Control Logic

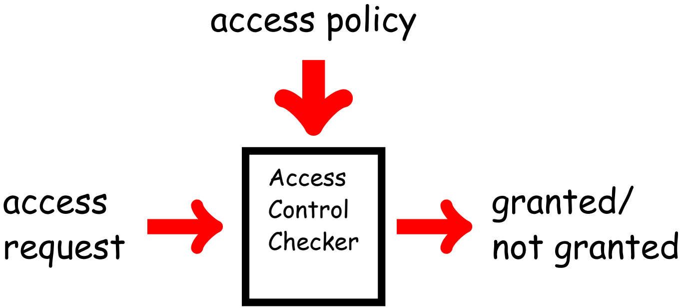
Ross Anderson about the use of Logic:

> Formal methods can be an excellent way of finding bugs in security protocol designs as they force the designer to make everything explicit and thus confront difficult design choices that might otherwise be fudged.



Assuming one file on my computer contains a virus.

Q: Given my access policy, can this file "infect" my whole computer?



Assuming one file on my computer contains a virus.

Q: Can my access policy prevent that my whole computer gets infected.

...
is_at_library (Christian)
is_student (a) $\wedge$ is_at_library (a) $\Rightarrow$ may_obtain_email (a)
is_staff (a) $\wedge$ is_at_library (a) $\Rightarrow$ may_obtain_email (a)

? may_obtain_email (Christian)

...

is_at_library (Christian)

is_student (a) $\wedge$ is_at_library (a) $\Rightarrow$ may_obtain_email (a)

is_staff (a) $\wedge$ is_at_library (a) $\Rightarrow$ may_obtain_email (a)

HoD says is_staff (a) $\Rightarrow$ is_staff (a)

HoD says is_staff (Christian)

? may_obtain_email (Christian)

$\ldots$
is_at_library (Christian)
is_student (a) ∧ is_at_library (a) ⇒ may_obtain_email (a)
is_staff (a) ∧ is_at_library (a) ⇒ may_obtain_email (a)

HoD says is_staff (a) ⇒ is_staff (a)
HoD says is_staff (Christian)

may_obtain_email (a) ∧ sending_spam (a) ⇒
¬ may_obtain_email (a)

? may_obtain_email (Christian)

There are two ways for tackling such problems:

- either you make up our own language in which you can describe the problem,
- or you use an existing language and represent the problem in this language.

Logic(s)

- Formulas

$$
\begin{array}{lcll}
F & ::= & \mathsf{true} & \\
 & | & \mathsf{false} & \\
 & | & F \wedge F & \\
 & | & F \vee F & \\
 & | & F \Rightarrow F & \text{implies} \\
 & | & \neg\, F & \text{negation} \\
 & | & p\,(t_1,\ldots,t_n) & \text{predicates}
\end{array}
$$

Terms $t ::= x \ldots \mid c \ldots$

Logic(s)

- Formulas

F	::=	true	
	|	false	
	|	$F \wedge F$	
	|	$F \vee F$	
	|	$F \Rightarrow F$	implies
	|	$\neg F$	negation
	|	$p\,(t_1,\ldots,t_n)$	predicates
	|	$\forall x.\ F$	forall quantification
	|	$\exists x.\ F$	exists quantification

Terms $t ::= x \ldots \mid c \ldots$

```
abstract class Term
case class Var(s: String) extends Term
case class Consts(s: String) extends Term
case class Fun(s: String, ts: List[Term]) extends Term

abstract class Form
case object True extends Form
case object False extends Form
case class And(f1: Form, f2: Form) extends Form
case class Or(f1: Form, f2: Form) extends Form
case class Imp(f1: Form, f2: Form) extends Form
case class Neg(f: Form) extends Form
case class Pred(s: String, ts: List[Term]) extends Form
```

Judgements

$$\Gamma \vdash F$$

Γ is a collection of formulas, called the assumptions

Judgements

$$\Gamma \vdash F$$

Γ is a collection of formulas, called the assumptions

- Example

$$\begin{array}{l} \text{is\_staff (Christian)}, \\ \text{is\_at\_library (Christian)}, \\ \forall\, x.\ \text{is\_at\_library}\ (x) \wedge \text{is\_staff}\ (x) \Rightarrow \text{may\_obtain\_email}\ (x) \end{array} \vdash \text{may\_obtain\_email (Christian)}$$

Judgements

$$\Gamma \vdash F$$

Γ is a collection of formulas, called the assumptions

- Example

$$\frac{\begin{array}{l}\text{is\_staff (Christian)}\\ \text{is\_at\_library (Christian)}\\ \forall x.\ \text{is\_at\_library}(x) \wedge \text{is\_staff}(x) \Rightarrow \text{may\_obtain\_email}(x)\end{array}}{\text{may\_obtain\_email (Christian)}}$$

Judgements

$$\Gamma \vdash F$$

Γ is a collection of formulas, called the assumptions

- Example

$$\frac{\begin{array}{l} is\_staff\ (Alice) \\ is\_staff\ (Christian) \\ is\_at\_library\ (Christian) \\ \forall x.\ is\_at\_library\ (x) \wedge is\_staff\ (x) \Rightarrow may\_obtain\_email\ (x) \end{array}}{may\_obtain\_email\ (Alice)}$$

```
abstract class Term
case class Var(s: String) extends Term
case class Consts(s: String) extends Term
case class Fun(s: String, ts: List[Term]) extends Term

abstract class Form
case object True extends Form
case object False extends Form
case class And(f1: Form, f2: Form) extends Form
case class Or(f1: Form, f2: Form) extends Form
case class Imp(f1: Form, f2: Form) extends Form
case class Neg(f: Form) extends Form
case class Pred(s: String, ts: List[Term]) extends Form

case class Judgement(Gamma: List[Form], F: Form) {
  def lhs = Gamma
  def rhs = F
}
```

Inference Rules

$$\frac{\text{premise}_1 \quad \ldots \quad \text{premise}_n}{\text{conclusion}}$$

The conlusion and premises are judgements

Inference Rules

$$\frac{\text{premise}_1 \quad \ldots \quad \text{premise}_n}{\text{conclusion}}$$

The conlusion and premises are judgements

- Examples

$$\frac{\Gamma \vdash F_1 \quad \Gamma \vdash F_2}{\Gamma \vdash F_1 \wedge F_2}$$

Inference Rules

$$\frac{\text{premise}_1 \quad \ldots \quad \text{premise}_n}{\text{conclusion}}$$

The conlusion and premises are judgements

- Examples

$$\frac{\Gamma \vdash F_1 \quad \Gamma \vdash F_2}{\Gamma \vdash F_1 \wedge F_2}$$

$$\frac{\Gamma \vdash F_1}{\Gamma \vdash F_1 \vee F_2} \qquad \frac{\Gamma \vdash F_2}{\Gamma \vdash F_1 \vee F_2}$$

Implication

$$\frac{\Gamma, F_1 \vdash F_2}{\Gamma \vdash F_1 \Rightarrow F_2}$$

$$\frac{\Gamma \vdash F_1 \Rightarrow F_2 \quad \Gamma \vdash F_1}{\Gamma \vdash F_2}$$

Universal Quantification

$$\frac{\Gamma \vdash \forall x.\ F}{\Gamma \vdash F[x := t]}$$

Start Rules / Axioms

if $F \in \Gamma$

$$\frac{}{\Gamma \vdash F}$$

Start Rules / Axioms

if $F \in \Gamma$

$$\overline{\Gamma \vdash F}$$

Also written as:

$$\overline{\Gamma, F \vdash F}$$

Start Rules / Axioms

if $F \in \Gamma$

$$\overline{\Gamma \vdash F}$$

Also written as:

$$\overline{\Gamma, F \vdash F}$$

$$\overline{\Gamma \vdash \text{true}}$$

Let Γ = is_staff (Christian),
is_at_library (Christian),
$\forall$ x. is_at_library (x) $\wedge$ is_staff (x) $\Rightarrow$ may_obtain_email (x)

Let Γ = is_staff (Christian),
is_at_library (Christian),
$\forall$ x. is_at_library (x) $\wedge$ is_staff (x) $\Rightarrow$ may_obtain_email (x)

$\Gamma \vdash$ is_staff (Christian) $\qquad$ $\Gamma \vdash$ is_at_library (Christian)

Let Γ = is_staff (Christian),
is_at_library (Christian),
$\forall x.$ is_at_library (x) $\wedge$ is_staff (x) $\Rightarrow$ may_obtain_email (x)

$$\frac{\Gamma \vdash \text{is\_staff (Christian)} \qquad \Gamma \vdash \text{is\_at\_library (Christian)}}{\Gamma \vdash \text{is\_staff (Christian)} \wedge \text{is\_at\_library (Christian)}}$$

Let Γ = is_staff (Christian),
is_at_library (Christian),
$\forall$ x. is_at_library (x) $\wedge$ is_staff (x) $\Rightarrow$ may_obtain_email (x)

$$\frac{\Gamma \vdash \text{is\_staff (Christian)} \qquad \Gamma \vdash \text{is\_at\_library (Christian)}}{\Gamma \vdash \text{is\_staff (Christian)} \wedge \text{is\_at\_library (Christian)}}$$

$\Gamma \vdash \forall$ x. is_staff (x) $\wedge$ is_at_library (x) $\Rightarrow$ may_obtain_email (x)

Let Γ = is_staff (Christian),
is_at_library (Christian),
$\forall$x. is_at_library (x) $\wedge$ is_staff (x) $\Rightarrow$ may_obtain_email (x)

$$\frac{\Gamma \vdash \text{is\_staff (Christian)} \qquad \Gamma \vdash \text{is\_at\_library (Christian)}}{\Gamma \vdash \text{is\_staff (Christian)} \wedge \text{is\_at\_library (Christian)}}$$

$$\frac{\Gamma \vdash \forall x.\ \text{is\_staff}(x) \wedge \text{is\_at\_library}(x) \Rightarrow \text{may\_obtain\_email}(x)}{\Gamma \vdash \text{is\_staff (Christian)} \wedge \text{is\_at\_library (Christian)} \Rightarrow \text{may\_obtain\_email (Christian)}}$$

Let Γ = is_staff (Christian),
is_at_library (Christian),
$\forall$ x. is_at_library (x) $\wedge$ is_staff (x) $\Rightarrow$ may_obtain_email (x)

$$\frac{\Gamma \vdash \text{is\_staff (Christian)} \qquad \Gamma \vdash \text{is\_at\_library (Christian)}}{\Gamma \vdash \text{is\_staff (Christian)} \wedge \text{is\_at\_library (Christian)}}$$

$$\frac{\Gamma \vdash \forall x.\ \text{is\_staff}(x) \wedge \text{is\_at\_library}(x) \Rightarrow \text{may\_obtain\_email}(x)}{\Gamma \vdash \text{is\_staff (Christian)} \wedge \text{is\_at\_library (Christian)} \Rightarrow \text{may\_obtain\_email (Christian)}}$$

$$\frac{\vdots \qquad\qquad \vdots}{\Gamma \vdash \text{may\_obtain\_email (Christian)}}$$

Access Control

$$\Gamma \vdash F$$

- If there is a proof $\Rightarrow$ yes (granted)
- If there isn't $\Rightarrow$ no (denied)

Access Control

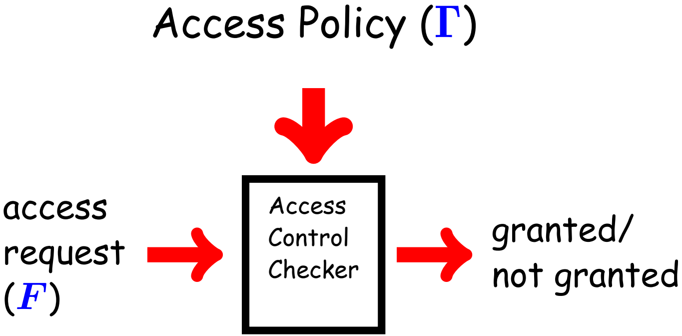
$$\Gamma \vdash F$$

- If there is a proof $\Rightarrow$ yes (granted)
- If there isn't $\Rightarrow$ no (denied)

$$\Gamma = \begin{array}{l} \text{is\_staff (Christian)}, \\ \text{is\_at\_library (Christian)}, \\ \forall x.\ \text{is\_at\_library}\ (x) \wedge \text{is\_staff}\ (x) \Rightarrow \text{may\_obtain\_email}\ (x) \end{array}$$

$$\Gamma \nvdash \text{may\_obtain\_email (Alice)}$$

The Access Control Problem



Bad News

- We introduced (roughly) first-order logic.

Bad News

- We introduced (roughly) first-order logic.
- Judgements

$$\Gamma \vdash F$$

are in general undecidable.

Bad News

- We introduced (roughly) first-order logic.
- Judgements

$$\Gamma \vdash F$$

are in general undecidable.

The problem is semi-decidable.

Access Control Logic

F	::=	true
	|	false
	|	$F \wedge F$
	|	$F \vee F$
	|	$F \Rightarrow F$
	|	p $(t_1,\ldots,t_n)$
	|	P says F — "saying predicate"

where P ::= Alice, Bob, Christian, ... (principals)

Access Control Logic

$$
\begin{array}{lcll}
F & ::= & \text{true} & \\
  & | & \text{false} & \\
  & | & F \wedge F & \\
  & | & F \vee F & \\
  & | & F \Rightarrow F & \\
  & | & p\,(t_1,\ldots,t_n) & \\
  & | & P \text{ says } F & \text{“saying predicate”}
\end{array}
$$

where P ::= Alice, Bob, Christian, ... (principals)

- HoD says is_staff (Christian)

```scala
abstract class Term
case class Var(s: String) extends Term
case class Consts(s: String) extends Term
case class Fun(s: String, ts: List[Term]) extends Term

abstract class Form
case object True extends Form
case object False extends Form
case class And(f1: Form, f2: Form) extends Form
case class Or(f1: Form, f2: Form) extends Form
case class Imp(f1: Form, f2: Form) extends Form
case class Neg(f: Form) extends Form
case class Pred(s: String, ts: List[Term]) extends Form
case class Says(s: String, f: Form) extends Form
```

Rules about Says

$$\frac{\Gamma \vdash F}{\Gamma \vdash P \text{ says } F}$$

$$\frac{\Gamma \vdash P \text{ says } (F_1 \Rightarrow F_2) \qquad \Gamma \vdash P \text{ says } F_1}{\Gamma \vdash P \text{ says } F_2}$$

$$\frac{\Gamma \vdash P \text{ says } (P \text{ says } F)}{\Gamma \vdash P \text{ says } F}$$

Consider the following scenario:

- If Admin says that $file_1$ should be deleted, then this file must be deleted.
- Admin trusts Bob to decide whether $file_1$ should be deleted.
- Bob wants to delete $file_1$.

Consider the following scenario:

- If Admin says that file_1 should be deleted, then this file must be deleted.
- Admin trusts Bob to decide whether file_1 should be deleted.
- Bob wants to delete file_1.

$$\Gamma = \begin{array}{l}(\text{Admin says del\_file}_1) \Rightarrow \text{del\_file}_1, \\ (\text{Admin says } ((\text{Bob says del\_file}_1) \Rightarrow \text{del\_file}_1)), \\ \text{Bob says del\_file}_1\end{array}$$

Consider the following scenario:

- If Admin says that $file_1$ should be deleted, then this file must be deleted.
- Admin trusts Bob to decide whether $file_1$ should be deleted.
- Bob wants to delete $file_1$.

$$\Gamma = \begin{array}{l} (\text{Admin says del\_file}_1) \Rightarrow \text{del\_file}_1, \\ (\text{Admin says } ((\text{Bob says del\_file}_1) \Rightarrow \text{del\_file}_1)), \\ \text{Bob says del\_file}_1 \end{array}$$

$$\Gamma \vdash \text{del\_file}_1$$

$$\frac{\Gamma \vdash F}{\Gamma \vdash P \text{ says } F}$$

$$\frac{\Gamma \vdash P \text{ says } (F_1 \Rightarrow F_2) \qquad \Gamma \vdash P \text{ says } F_1}{\Gamma \vdash P \text{ says } F_2}$$

$$\Gamma = \begin{array}{l} (\text{Admin says del\_file}_1) \Rightarrow \text{del\_file}_1, \\ (\text{Admin says } ((\text{Bob says del\_file}_1) \Rightarrow \text{del\_file}_1)), \\ \text{Bob says del\_file}_1 \end{array}$$

$$\Gamma \vdash \text{del\_file}_1$$

$$\underbrace{\frac{\overline{\Gamma \vdash \text{Bob says del\_file}}}{\Gamma \vdash \text{Admin says (Bob says del\_file)}}}_{X}$$

$$\underbrace{\frac{\overline{\Gamma \vdash \text{Admin says (Bob says del\_file} \Rightarrow \text{del\_file)}} \quad \begin{matrix}\vdots \\ X\end{matrix}}{\Gamma \vdash \text{Admin says del\_file}}}_{Y}$$

$$\frac{\overline{\Gamma \vdash \text{(Admin says del\_file)} \Rightarrow \text{del\_file}} \quad \begin{matrix}\vdots \\ Y\end{matrix}}{\Gamma \vdash \text{del\_file}}$$

Controls

- $P \text{ controls } F \equiv (P \text{ says } F) \Rightarrow F$
- its meaning "P is entitled to do F"
- if P controls F and P says F then F

Controls

- $\text{P controls F} \equiv (\text{P says F}) \Rightarrow \text{F}$
- its meaning "P is entitled to do F"
- if P controls F and P says F then F

$$\frac{\Gamma \vdash \text{P controls F} \quad \Gamma \vdash \text{P says F}}{\Gamma \vdash \text{F}}$$

Controls

- $P \text{ controls } F \equiv (P \text{ says } F) \Rightarrow F$
- its meaning “P is entitled to do F”
- if P controls F and P says F then F

$$\frac{\Gamma \vdash P \text{ controls } F \quad \Gamma \vdash P \text{ says } F}{\Gamma \vdash F}$$

$$\frac{\Gamma \vdash (P \text{ says } F) \Rightarrow F \quad \Gamma \vdash P \text{ says } F}{\Gamma \vdash F}$$

Speaks For

- $P \mapsto Q \equiv \forall F.\ (P \text{ says } F) \Rightarrow (Q \text{ says } F)$
- its meaning "P speaks for Q"

$$\frac{\Gamma \vdash P \mapsto Q \quad \Gamma \vdash P \text{ says } F}{\Gamma \vdash Q \text{ says } F}$$

Speaks For

- $P \mapsto Q \equiv \forall F.\ (P \text{ says } F) \Rightarrow (Q \text{ says } F)$
- its meaning "P speaks for Q"

$$\frac{\Gamma \vdash P \mapsto Q \quad \Gamma \vdash P \text{ says } F}{\Gamma \vdash Q \text{ says } F}$$

$$\frac{\Gamma \vdash P \mapsto Q \quad \Gamma \vdash Q \text{ controls } F}{\Gamma \vdash P \text{ controls } F}$$

$$\frac{\Gamma \vdash P \mapsto Q \quad \Gamma \vdash Q \mapsto R}{\Gamma \vdash P \mapsto R}$$

Tickets

- Tickets control access to restricted objects.

  Example: Permitted (Bob, enter_flight) ?

- Bob says Permitted (Bob, enter_flight)
  (access request)
- Ticket says (Bob controls Permitted (Bob, enter_flight))
- Airline controls (Bob controls Permitted (Bob, enter_flight))
  (access policy)

Tickets

- Tickets control access to restricted objects.

Example: Permitted (Bob, enter_flight) ?

- Bob says Permitted (Bob, enter_flight)
  (access request)
- Ticket says (Bob controls Permitted (Bob, enter_flight))
- Airline controls (Bob controls Permitted (Bob, enter_flight))
  (access policy)
- Ticket $\mapsto$ Airline
  (trust assumption)

Tickets

1. Bob says Permitted (Bob, enter_flight)
2. Ticket says (Bob controls Permitted (Bob, enter_flight))
3. Airline controls (Bob controls Permitted (Bob, enter_flight))
4. Ticket $\mapsto$ Airline

Is $\Gamma \vdash$ Permitted (Bob, enter_flight) derivable ?

$$\frac{\Gamma \vdash \text{P controls F} \quad \Gamma \vdash \text{P says F}}{\Gamma \vdash \text{F}} \qquad \frac{\Gamma \vdash \text{P} \mapsto \text{Q} \quad \Gamma \vdash \text{P says F}}{\Gamma \vdash \text{Q says F}}$$

Tickets

- Access Request:

  Person says Object

- Ticket:

  Ticket says (Person controls Object)

- Access policy:

  Authority controls (Person controls Object)

- Trust assumption:

  $$\text{Ticket} \mapsto \text{Authority}$$

Derived Rule for Tickets

$$\frac{\begin{array}{l}\text{Authority controls (Person controls F)}\\ \text{Ticket says (Person controls F)}\\ \text{Ticket} \mapsto \text{Authority}\\ \text{Person says F}\end{array}}{\text{F}}$$

$$\frac{\Gamma \vdash \text{P controls F} \quad \Gamma \vdash \text{P says F}}{\Gamma \vdash \text{F}} \qquad \frac{\Gamma \vdash \text{P} \mapsto \text{Q} \quad \Gamma \vdash \text{P says F}}{\Gamma \vdash \text{Q says F}}$$

Security Levels

- Top secret (TS)
- Secret (S)
- Public (P)

$$slev(P) < slev(S) < slev(TS)$$

Security Levels

- Top secret (TS)
- Secret (S)
- Public (P)

$$slev(P) < slev(S) < slev(TS)$$

- Bob has a clearance for "secret"
- Bob can read documents that are public or sectret, but not top secret

Reading a File

$$\frac{\text{Bob controls Permitted (File, read)} \quad \text{Bob says Permitted (File, read)}}{\text{Permitted (File, read)}}$$

Reading a File

$$\frac{\begin{array}{l} slev(\text{File}) < slev(\text{Bob}) \Rightarrow \\ \qquad\qquad \text{Bob controls Permitted (File, read)} \\ \text{Bob says Permitted (File, read)} \\ slev(\text{File}) < slev(\text{Bob}) \end{array}}{\text{Permitted (File, read)}}$$

Reading a File

$$\frac{\begin{array}{l} slev(\text{File}) < slev(\text{Bob}) \Rightarrow \\ \qquad\qquad \text{Bob controls Permitted (File, read)} \\ \text{Bob says Permitted (File, read)} \\ slev(\text{File}) = P \\ slev(\text{Bob}) = S \\ slev(P) < slev(S) \end{array}}{\text{Permitted (File, read)}}$$

Substitution Rule

$$\frac{\Gamma \vdash slev(P) = l_1 \quad \Gamma \vdash slev(Q) = l_2 \quad \Gamma \vdash l_1 < l_2}{\Gamma \vdash slev(P) < slev(Q)}$$

Substitution Rule

$$\frac{\Gamma \vdash slev(P) = l_1 \quad \Gamma \vdash slev(Q) = l_2 \quad \Gamma \vdash l_1 < l_2}{\Gamma \vdash slev(P) < slev(Q)}$$

- $slev(\mathsf{Bob}) = S$
- $slev(\mathsf{File}) = P$
- $slev(P) < slev(S)$

Reading a File

$$\frac{\begin{array}{l} slev(\text{File}) < slev(\text{Bob}) \Rightarrow \\ \qquad\qquad \text{Bob controls Permitted (File, read)} \\ \text{Bob says Permitted (File, read)} \\ slev(\text{File}) = P \\ slev(\text{Bob}) = TS \\ ? \end{array}}{\text{Permitted (File, read)}}$$

Reading a File

$$\frac{\begin{array}{l} slev(\text{File}) < slev(\text{Bob}) \Rightarrow \\ \qquad\qquad \text{Bob controls Permitted (File, read)} \\ \text{Bob says Permitted (File, read)} \\ slev(\text{File}) = P \\ slev(\text{Bob}) = TS \\ slev(P) < slev(S) \\ slev(S) < slev(TS) \end{array}}{\text{Permitted (File, read)}}$$

Transitivity Rule

$$\frac{\Gamma \vdash l_1 < l_2 \qquad \Gamma \vdash l_2 < l_3}{\Gamma \vdash l_1 < l_3}$$

- $slev(P) < slev(S)$
- $slev(S) < slev(TS)$

  $slev(P) < slev(TS)$

Reading Files

- Access policy for reading

$$\frac{\begin{array}{l}\forall f.\ slev(f) < slev(\mathsf{Bob}) \Rightarrow \\ \qquad\qquad \mathsf{Bob\ controls\ Permitted}\ (f, \mathsf{read}) \\ \mathsf{Bob\ says\ Permitted}\ (\mathsf{File}, \mathsf{read}) \\ slev(\mathsf{File}) = P \\ slev(\mathsf{Bob}) = TS \\ slev(P) < slev(S) \\ slev(S) < slev(TS)\end{array}}{\mathsf{Permitted}\ (\mathsf{File}, \mathsf{read})}$$

Reading Files

- Access policy for reading

$$\forall f.\ slev(f) \leq slev(\text{Bob}) \Rightarrow \text{Bob controls Permitted}\ (f, \text{read})$$

$$\cfrac{\begin{array}{l}\forall f.\ slev(f) \leq slev(\text{Bob}) \Rightarrow \text{Bob controls Permitted}\ (f, \text{read}) \\ \text{Bob says Permitted (File, read)} \\ slev(\text{File}) = TS \\ slev(\text{Bob}) = TS \\ slev(P) < slev(S) \\ slev(S) < slev(TS)\end{array}}{\text{Permitted (File, read)}}$$

Writing Files

- Access policy for writing

$$\frac{\begin{array}{l}\forall f.\ slev(\text{Bob}) \leq slev(f) \Rightarrow \\ \qquad\qquad \text{Bob controls Permitted}\ (f, \text{write}) \\ \text{Bob says Permitted (File, write)} \\ slev(\text{File}) = TS \\ slev(\text{Bob}) = S \\ slev(P) < slev(S) \\ slev(S) < slev(TS)\end{array}}{\text{Permitted (File, write)}}$$

Bell-LaPadula

- Read Rule: A principal P can read an object O if and only if P's security level is at least as high as O's.
- Write Rule: A principal P can write an object O if and only if O's security level is at least as high as P's.
- Meta-Rule: All principals in a system should have a sufficiently high security level in order to access an object.

This restricts information flow $\Rightarrow$ military

Bell-LaPadula

- Read Rule: A principal P can read an object O if and only if P's security level is at least as high as O's.
- Write Rule: A principal P can write an object O if and only if O's security level is at least as high as P's.
- Meta-Rule: All principals in a system should have a sufficiently high security level in order to access an object.

This restricts information flow $\Rightarrow$ military

Bell-LaPadula: **'no read up'** - **'no write down'**

Principle of Least Privilege

A principal should have as few privileges as possible to access a resource.

- Bob (TS) and Alice (S) want to communicate

  $\Rightarrow$ Bob should lower his security level

Biba Policy

Data Integrity (rather than data confidentiality)

- Biba: **'no read down'** - **'no write up'**
- Read Rule: A principal P can read an object O if and only if P's security level is lower or equal than O's.
- Write Rule: A principal P can write an object O if and only if O's security level is lower or equal than P's.

Biba Policy

Data Integrity (rather than data confidentiality)

- Biba: **'no read down' - 'no write up'**
- Read Rule: A principal P can read an object O if and only if P's security level is lower or equal than O's.
- Write Rule: A principal P can write an object O if and only if O's security level is lower or equal than P's.

E.g. Generals write orders to officers; officers write oders to solidiers
Firewall: you can read from inside the firewall, but not from outside
Phishing: you can look at an approved PDF, but not one from a random email

Point to Take Home

- Formal methods can be an excellent way of finding bugs as they force the designer to make everything explicit and thus confront difficult design choices that might otherwise be fudged.