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)

Last Week

$$\begin{array}{rcl} A & \rightarrow & B : \ldots \\ B & \rightarrow & A : \ldots \\ & \vdots & \end{array}$$

- by convention A, B are named principals Alice... but most likely they are programs
- indicates one "protocol run", or session, which specifies an order in the communication
- there can be several sessions in parallel (think of wifi routers)
- nonces (randomly generated numbers) used only once

Cryptographic Protocol Failures

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.

Protocols

Examples where "over-the-air" protocols are used

- wifi
- card readers (you cannot trust the terminals)
- RFI (passports)

"On the Internet, nobody knows you're a dog."

Chip-and-PIN

- A “tamperesitant” terminal playing Tetris on youtube.
  (http://www.youtube.com/watch?v=wWTzkD9M0sU)

Oyster Cards

- good example of a bad protocol (security by obscurity)

Wirelessly Pickpocketing a Mifare Classic Card

The Mifare Classic is the most widely used contactless smartcard on the market. The stream cipher CRYPTO1 used by the Classic has recently been reverse engineered and serious attacks have been proposed. The most serious of them retrieves a secret key in under a second. In order to clone a card, previously proposed attacks require that the adversary either has access to an eavesdropped communication session or executes a message-by-message man-in-the-middle attack between the victim and a legitimate reader. Although this is already disastrous from a cryptographic point of view, system integrators maintain that these attacks cannot be performed undetected.

This paper proposes four attacks that can be executed by an adversary having only wireless access to just a card (and not to a legitimate reader). The most serious of them recovers a secret key in less than a second on ordinary hardware. Besides the cryptographic weaknesses, we exploit other weaknesses in the protocol stack. A vulnerability in the computation of parity bits allows an adversary to establish a side channel. Another vulnerability regarding nested authentications provides enough plaintext for a speedy known-plaintext attack.

Oyster Cards

- good example of a bad protocol
(security by obscurity)

- “Breaching security on Oyster cards should not allow unauthorised use for more than a day, as TfL promises to turn off any cloned cards within 24 hours...”

Another Example

In an email from Ross Anderson

From: Ross Anderson <Ross.Anderson@cl.cam.ac.uk>
Sender: cl-security-research-bounces@lists.cam.ac.uk
To: cl-security-research@lists.cam.ac.uk
Subject: Birmingham case
Date: Tue, 13 Aug 2013 15:13:17 +0100

As you may know, Volkswagen got an injunction against the University of Birmingham suppressing the publication of the design of a weak cipher used in the remote key entry systems in its recent-model cars. The paper is being given today at Usenix, minus the cipher design.

I've been contacted by Birmingham University's lawyers who seek to prove that the cipher can be easily obtained anyway. They are looking for a student who will download the firmware from any newish VW, disassemble it and look for the cipher. They'd prefer this to be done by a student rather than by a professor to emphasise how easy it is.

Volkswagen's argument was that the Birmingham people had reversed a locksmithing tool produced by a company in Vietnam, and since their key fob chip is claimed to be tamper-resistant, this must have involved a corrupt insider at VW or at its supplier Thales. Birmingham's argument is that this is nonsense as the cipher is easy to get hold of. Their lawyers feel this argument would come better from an independent outsider.

Let me know if you're interested in having a go, and I'll put you in touch
Ross

Authentication Protocols

Alice (A) and Bob (B) share a secret key K_{AB}

Passwords:

$$B \rightarrow A : K_{AB}$$

Authentication Protocols

Alice (A) and Bob (B) share a secret key K_{AB}

Passwords:

$$B \rightarrow A : K_{AB}$$

Problems: Eavesdropper can capture the secret and replay it; A cannot confirm the identity of B

Authentication Protocols

Alice (A) and Bob (B) share a secret key K_{AB}

Simple Challenge Response:

$$
\begin{array}{ll}
A \rightarrow B: & N \\
B \rightarrow A: & \{N\}_{K_{AB}}
\end{array}
$$

Authentication Protocols

Alice (A) and Bob (B) share a secret key K_{AB}

Mutual Challenge Response:

$$
\begin{array}{ll}
A \rightarrow B: & N_A \\
B \rightarrow A: & \{N_A, N_B\}_{K_{AB}} \\
A \rightarrow B: & N_B
\end{array}
$$

One Time Passwords

$$B \to A : C, C_{K_{AB}}$$

A counter C increases with each transmission; A will not accept a C which has already been accepted (used in car key fob).

Person-in-the-Middle

“Normal” protocol run:

- A sends public key to B
- B sends public key to A
- A sends message encrypted with B’s public key, B decrypts it with its private key
- B sends message encrypted with A’s public key, A decrypts it with its private key

Person-in-the-Middle

Attack:

- A sends public key to B — C intercepts this message and send his own public key
- B sends public key to A — C intercepts this message and send his own public key
- A sends message encrypted with C's public key, C decrypts it with its private key, re-encrypts with B's public key
- similar

Person-in-the-Middle

Prevention:

- A sends public key to B
- B sends public key to A
- A encrypts message with B's public key, send's **half** of the message
- B encrypts message with A's public key, send's **half** of the message
- A sends other half, B can now decrypt entire message
- B sends other half, A can now decrypt entire message

Person-in-the-Middle

Prevention:

- A sends public key to B
- B sends public key to A
- A encrypts message with B's public key, send's **half** of the message
- B encrypts message with A's public key, send's **half** of the message
- A sends other half, B can now decrypt entire message
- B sends other half, A can now decrypt entire message

C would have to invent a totally new message

Motivation

The ISO/IEC 9798 specifies authentication mechanisms which use security techniques. These mechanisms are used to corroborate that an entity is the one that is claimed. An entity to be authenticated proves its identity by showing its knowledge of a secret. The mechanisms are defined as exchanges of information between entities and, where required, exchanges with a trusted third party.

Motivation

But...

The ISO/IEC 9798 standard neither specifies a threat model nor defines the security properties that the protocols should satisfy.

Motivation

But...

The ISO/IEC 9798 standard neither specifies a threat model nor defines the security properties that the protocols should satisfy.

Unfortunately, there are no general precise definitions for the goals of protocols.

Best Practices

Principle 1: Every message should say what it means: the interpretation of a message should not depend on the context.

Best Practices

Principle 1: Every message should say what it means: the interpretation of a message should not depend on the context.

Principle 2: If the identity of a principal is essential to the meaning of a message, it is prudent to mention the principal's name explicitly in the message (though difficult).

Best Practices

Principle 3: Be clear about why encryption is being done. Encryption is not wholly cheap, and not asking precisely why it is being done can lead to redundancy. Encryption is not synonymous with security.

Possible Uses of Encryption

- Preservation of confidentiality: $\{X\}_K$ only those that have K may recover X.
- Guarantee authenticity: The partner is indeed some particular principal.
- Guarantee confidentiality and authenticity: binds two parts of a message — $\{X, Y\}_K$ is not the same as $\{X\}_K$ and $\{Y\}_K$.

Best Practices

Principle 4: The protocol designer should know which trust relations his protocol depends on, and why the dependence is necessary. The reasons for particular trust relations being acceptable should be explicit though they will be founded on judgment and policy rather than on logic.

Example Certification Authorities: CAs are trusted to certify a key only after proper steps have been taken to identify the principal that owns it.

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.*

Logic(s)

- Formulas

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

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

Logic(s)

- Formulas

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

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

```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
```

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

is_staff (Alice)
is_staff (Christian)
is_at_library (Christian)
∀ x. is_at_library (x) ∧ is_staff (x) ⇒ may_obtain_email (x)

---

may_obtain_email (Alice)

```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 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

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

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

Universal Quantification

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

Start Rules / Axioms

if $F \in \Gamma$

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

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:

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

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

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

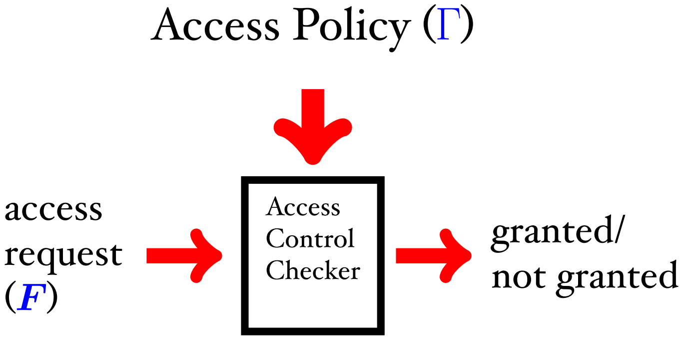
$$\Gamma \vdash F$$

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

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

$$\Gamma \not\vdash \text{may\_obtain\_email (Alice)}$$

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

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

"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 $P ::= \text{Alice}, \text{Bob}, \text{Christian}, \ldots$ (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.

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

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

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

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

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

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

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

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

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

Controls

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

Controls

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

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

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

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

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

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"

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

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

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

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

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

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

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

Substitution Rule

$$\frac{\Gamma \vdash \boldsymbol{slev}(\boldsymbol{P}) = \boldsymbol{l}_1 \quad \Gamma \vdash \boldsymbol{slev}(\boldsymbol{Q}) = \boldsymbol{l}_2 \quad \Gamma \vdash \boldsymbol{l}_1 < \boldsymbol{l}_2}{\Gamma \vdash \boldsymbol{slev}(\boldsymbol{P}) < \boldsymbol{slev}(\boldsymbol{Q})}$$

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

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

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

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

  $slev(P) < slev(TS)$

Reading Files

- Access policy for reading

$$\forall f.\ slev(f) < slev(\text{Bob}) \Rightarrow$$
$$\text{Bob controls Permitted } (f, \text{read})$$
$$\text{Bob says Permitted (File, read)}$$
$$slev(\text{File}) = P$$
$$slev(\text{Bob}) = TS$$
$$slev(P) < slev(S)$$
$$slev(S) < slev(TS)$$
$$\overline{\hspace{10em}}$$
$$\text{Permitted (File, read)}$$

Reading Files

- Access policy for reading

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

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

Writing Files

- Access policy for writing

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

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.