A Propositional Logic Model for Access Control

Dimitar Guelev
The University of Birmingham

Joint work with Mark Ryan and Pierre Yves Schobbens

Plan of talk

The Harrison-Russo-Ullman Model - an introductory example.

Formal definition of our model

An example: conference paper review system

Programming access

- semantics of programs
- some high-level accessibility operators
- the need to consider general access goals

A general accessibility operator

- reduction rules for accessibility

Some generalisations

- intruders who act concurrently
- high-level actions
- general knowledge states

Some properties of the modelling framework

The model of Harrison, Russo and Ullman

An access control matrix:

	$\dots$	agent a	$\dots$
$\dots$	$\dots$	$\dots$	$\dots$
resource r	$\dots$	Abstract rights of a on r	$\dots$
$\dots$	$\dots$	$\dots$	$\dots$

and high level actions of the form

> If agents $x_1, \dots, x_n$ have certain rights on $r_1, \dots, r_m$ then some sequence of actions $s(x_1, \dots, x_n, r_1, \dots, r_m)$ can be carried out.

Actions include creating, deleting and updating resources and agents and assigning and revoking rights.

Definition of our model

- Σ — set of agents; finite $A \subseteq \Sigma$ are called coalitions;
  P — set of propositional variables.

- An access control system S is $\langle P, \Sigma, \mathtt{r}, \mathtt{w} \rangle$, where $\mathtt{r}$ and $\mathtt{w}$ are mappings of type $P \times P_{\mathit{fin}}(\Sigma) \to \Phi(P)$, satisfying

$$A \subset A' \text{ implies } \vdash \mathtt{x}(p, A) \Rightarrow \mathtt{x}(p, A') \text{ where } \mathtt{x} \in \{\mathtt{r}, \mathtt{w}\}.$$

- A can read p at state s iff $s \models \mathtt{r}(p, A)$.
  A can overwrite p at state s iff $s \models \mathtt{w}(p, A)$.

  A can run $p := e$ at s iff $s \models \mathtt{w}(p, A) \wedge \bigwedge_{q \in \mathrm{Var}(e)} \mathtt{r}(q, A)$.

- All agents know the definitions of $\mathtt{r}$ and $\mathtt{w}$, but cannot necessarily read the values of formulas $\mathtt{r}(p, A), \mathtt{w}(p, A)$.

Example: a conference paper review system

Let $\texttt{Papers}$ be a fixed set, $\texttt{author} : \texttt{Papers} \times \Sigma \rightarrow \texttt{Boolean}$ a fixed function, and $c \in \Sigma$ a constant denoting the PC chair.

For each $p \in \texttt{Papers}$ and each $a, b \in \texttt{Agents}$, P contains:

$\texttt{pcmember}(a)$	a is a PC member
$\texttt{reviewer}(p, a)$	paper p is assigned to PC member a
$\texttt{subreviewer}(p, a, b)$	p is assigned to sub-reviewer b by a
$\texttt{submittedReview}(p, b)$	a review of p has been submitted by b
$\texttt{review}(p, b)$	the review of p from b

Example continued

$\texttt{pcmember}(.)$:

$$\texttt{r}(\,\texttt{pcmember}(a),\ A\,) \rightleftharpoons \top.$$

$$\texttt{w}(\,\texttt{pcmember}(a),\ A\,) \rightleftharpoons \{a, c\} \subseteq A \vee (a \in A \wedge \texttt{pcmember}(a)).$$

$\texttt{reviewer}(.,.)$:

$$\texttt{r}(\texttt{reviewer}(p, a), \{x\}) \rightleftharpoons \texttt{pcmember}(x) \wedge \neg\texttt{author}(p, x)$$

$$\begin{aligned}&\texttt{w}(\texttt{reviewer}(p, a), A) \rightleftharpoons \\ &\quad (\neg\texttt{reviewer}(p, a) \wedge \{a, c\} \subseteq A \wedge \texttt{pcmember}(a) \wedge \neg\texttt{author}(p, a)) \vee \\ &\quad (\texttt{reviewer}(p, a) \wedge (a \in A \vee c \in A) \wedge \neg \bigvee_{b \in \Sigma} \texttt{subreviewer}(p, a, b))\end{aligned}$$

Comparison with the HRU model

We can assume there is an access control matrix again:

	...	Coalition A	...
...	...	...	...
Resource p	...	$\mathtt{r}(p, A),\ \mathtt{w}(p, A)$	...
...	...	...	...

However, rights in the matrix are updated automatically by whatever other updates are made on the system.

Programming access

$\alpha ::= \texttt{skip} \mid p{:=}\varphi \mid \texttt{if } \varphi \texttt{ then } \alpha \texttt{ else } \alpha \mid (\alpha; \alpha)$

Semantics of programs: $[\![\alpha]\!]$: State $\rightarrow$ State where State $= (P \rightarrow \{0, 1\})$

- $[\![\texttt{skip}]\!](s) = s$;
- $[\![p := \varphi]\!](s) = s[p := s(\varphi)]$
- $[\![\texttt{if } \varphi \texttt{ then } \alpha \texttt{ else } \beta]\!](s) = \begin{cases} [\![\alpha]\!](s), & \text{if } s(\varphi) = \text{ true;} \\ [\![\beta]\!](s) & \text{otherwise.} \end{cases}$
- $[\![(\alpha; \beta)]\!](s) = [\![\beta]\!]([\![\alpha]\!](s))$.

Let φ_p hold at s iff $[\![\alpha]\!](s)(p) = 1$. Then we denote the substitution

$$[\varphi_p/p : p \in P]$$

by $[\![\alpha]\!]$ too.

Semantics of programs continued

$[\![\alpha, A]\!]$ is a boolean formula expressing whether A can execute α.

- $[\![A, \mathtt{skip}]\!] = \top$;
- $[\![A, p := \varphi]\!] = \bigwedge_{q \in \mathrm{Var}(\varphi)} \mathtt{r}(q, A) \wedge \mathtt{w}(p, A)$;
- $[\![A, \mathtt{if}\ \varphi\ \mathtt{then}\ \alpha\ \mathtt{else}\ \beta]\!] =$
  $\mathtt{r}(\varphi, A) \wedge (\varphi \Rightarrow [\![A, \alpha]\!]) \wedge (\neg\varphi \Rightarrow [\![A, \beta]\!])$;
- $[\![A, (\alpha; \beta)]\!] = [\![A, \alpha]\!] \wedge \underbrace{[\![\alpha]\!]}_{\text{regarded as a substitution}} [\![A, \beta]\!]$

Two kinds of property

- Is there a program to achieve φ? $\exists\alpha.\forall s.\,[\![\alpha]\!](s) \models \varphi$
  – useful for user goals;
- Is φ achievable (e.g. by guessing)? $\forall s.\exists\alpha.\,[\![\alpha]\!](s) \models \varphi$
  – useful for intruder goals.

Example 2

A	u	x	y	z
$\mathtt{r}$	$\bot$	$\top$	$\top$	$\top$
$\mathtt{w}$	$\bot$	$\neg u$	u	$x \vee y$

Initial state: $s \models \neg x, \neg y$. Goal: to overwrite z.

Being unable to read u, A cannot overwrite z starting at state s without risking a failure.

Expressing user capabilities

System administrators would like to verify

- $\mathtt{R}_A\varphi$, expressing that there is a program which A can execute which will enable them to discover the value of φ;
- $\mathtt{W}_A\varphi$, expressing that there is a program which A can execute which will enable them to write the value of φ.

Of course, $\mathtt{r}(\varphi, A)$ should imply $\mathtt{R}_A\varphi$, and $\mathtt{w}(\varphi, A)$ should imply $\mathtt{W}_A\varphi$, but not conversely.

Defining $\mathtt{R}_A\varphi$ and $\mathtt{W}_A\varphi$

A can obtain the truth value of φ if it can run some α that copies the value of φ into a dedicated variable p_φ, starting from any state.

- $\forall p\, \varphi \rightleftharpoons [\bot/p]\varphi \wedge \varphi[\top/p]$
- $\forall\varphi \rightleftharpoons \forall p_1 \ldots \forall p_k \varphi$, where $\{p_1, \ldots, p_k\} = \mathrm{Var}(\varphi)$.

Then we can define

$\mathtt{R}_A\varphi \rightleftharpoons \exists\alpha\forall([\![A, \alpha]\!] \wedge ([\![\alpha]\!]p_\varphi \Leftrightarrow \varphi))$.

$\mathtt{W}_A\varphi \rightleftharpoons \exists\alpha_\top, \alpha_\bot\forall([\![\alpha_\top]\!]\varphi \wedge [\![\alpha_\bot]\!]\neg\varphi \wedge [\![A, \alpha_\top]\!] \wedge [\![A, \alpha_\bot]\!])$

$\mathtt{R}_A\varphi$ and $\mathtt{W}_A\varphi$ can be computed by replacing $\exists\alpha$ with a disjunction over the finitely many pairwise non-equivalent programs on S.

Note that the programs α, $\alpha_\top$ and $\alpha_\bot$ may have destructive side effects.

General access goals

$\mathtt{R}_A\varphi$ and $\mathtt{W}_A\varphi$ define the existence of programs for reading and writing. Because of the possible side effects, complex goals need to be treated as a whole:

Example 3 Initial state: $s \models x, y$.

A	x	y	z	t
$\mathtt{r}$	$\top$	$\top$	x	$\neg x$
$\mathtt{w}$	$x \wedge y$	$x \wedge y$	$\top$	$\top$

Goal 1: set both x and y to 0. Goal 2: Read z and t.

Goal 1 cannot be achieved. We have $\mathtt{W}_A x$ and $\mathtt{W}_A y$, but not $\mathtt{W}_A(x \vee y)$.

To achieve Goal 2, reading z should be done first.

A general accessibility operator

Basic goals:

- $\overline{\varphi}'$, the goal of making φ true;
- $\overline{\varphi}$, the goal of realising that φ is true

Generalised goals:

$$\psi ::= \bot \mid \top \mid \overline{\varphi}' \mid \overline{\varphi} \mid \psi \wedge \psi \mid \psi \vee \psi \mid B(\Phi, \psi)$$

Examples:

$\overline{p}' \wedge (\overline{q} \Leftrightarrow \overline{q}')$ - set p to 1 and preserve the value of q;

$\overline{p} \wedge \overline{\neg p}' \vee \overline{\neg p} \wedge \overline{p}'$ - change the value of p.

$A(\Phi, \psi)$ denotes the goal for A to read each formula the list Φ and achieve the generalised goal ψ. $A(.,.)$ subsumes $\mathtt{R}_A$ and $\mathtt{W}_A$:

$$\mathtt{R}_A\varphi = A(\langle\varphi\rangle, \top), \qquad \mathtt{W}_A\varphi = A(\emptyset, \overline{\varphi}')$$

Reduction rules for $A(.,.)$

Let $T \subseteq V \subseteq P$. Then $A^{V,T}(\Phi, \psi)$ means:

A can achieve (Φ, ψ), knowing that for $p \in V$, p holds iff $p \in T$.

Let

$$\sigma_{p,\tau} \rightleftharpoons [\overline{[\tau/p]\varphi}/\overline{\varphi} : \varphi \in \mathbf{L}(P)], \quad \tau \in \{\bot, \top\}$$
$$\sigma_{V,T} \rightleftharpoons [\bot/p : p \in V \setminus T][\top/p : p \in T]$$

Reduction for sampling steps:

$$\left(\begin{array}{c} \forall \sigma_{V,T} \mathtt{r}(p, A) \wedge \\ A^{V \cup \{p\}, T \cup \{p\}}([\top/p]\Phi, \sigma_{p,\top}\psi) \wedge \\ A^{V \cup \{p\}, T}([\bot/p]\Phi, \sigma_{p,\bot}\psi) \end{array}\right) \Rightarrow A^{V,T}(\Phi, \psi)$$

Reduction rules for $A(.,.)$ continued

Let $\Phi = \langle \varphi_1, \ldots, \varphi_l \rangle$ and $G \rightleftharpoons \bigwedge_{i=1}^{l} ([\bot/p]\varphi_i \Leftrightarrow [\top/p]\varphi_i)$. G expresses that the truth values of $\varphi_1, \ldots, \varphi_l$ do not depend on p.

Reduction for overwriting $p \notin V$ with 0:

$$A^{V\cup\{p\},T}(\Phi, \overline{G} \wedge \sigma_{p,\bot}\psi \wedge \sigma_{p,\top}\psi) \wedge \forall\sigma_{V,T}\mathtt{w}(p, A) \Rightarrow A^{V,T}(\Phi, \psi)$$

Simplification for nested goals:

$$B^{V,T}(\Phi' * \Phi, \psi') \Rightarrow A^{V,T}(\Phi, B(\Phi', \psi'))$$

Immediate simplifications:

$$\bigwedge_{i=1}^{l} (\forall\sigma_{V,T}\varphi_i \vee \forall\neg\sigma_{V,T}\varphi_i) \wedge A^{V,T}(\langle\rangle, \psi) \Rightarrow A^{V,T}(\Phi, \psi)$$

$$\forall\sigma_{V,T}[\varphi/\overline{\varphi}', \varphi/\overline{\varphi} : \varphi \in \mathbf{L}(P)]\psi \Rightarrow A^{V,T}(\langle\rangle, \psi)$$

Some generalisations: 1. Concurrent intruder

Consider regular user A running α and intruder B running β concurrently. To assume the worst case, let B control the interleaving.

$[\![\alpha \| \mathtt{skip}]\!] = [\![\alpha]\!]$;

$[\![\alpha \| (p := \varphi; \beta)]\!] = \lambda s.[\![(\alpha \| \beta)]\!](s[p := s(\varphi)])$;

$[\![\alpha \| (\mathtt{if}\ \varphi\ \mathtt{then}\ \beta_1\ \mathtt{else}\ \beta_2); \beta_3)]\!] =$

$$\lambda s.\mathtt{if}\ s(\varphi)\ \mathtt{then}\ [\![\alpha \| (\beta_1; \beta_3)]\!](s)\ \mathtt{else}\ [\![\alpha \| (\beta_2; \beta_3)]\!](s)$$

$[\![\mathtt{skip} \| (\mathtt{sleep}; \beta)]\!] = [\![\beta]\!]$;

$[\![(p := \varphi; \alpha) \| (\mathtt{sleep}; \beta)]\!] = \lambda s.[\![(\alpha \| \beta)]\!](s[p := s(\varphi)])$;

$[\![(\mathtt{if}\ \varphi\ \mathtt{then}\ \alpha_1\ \mathtt{else}\ \alpha_2); \alpha_3) \| (\mathtt{sleep}; \beta)]\!] =$

$\lambda s.\mathtt{if}\ s(\varphi)\ \mathtt{then}\ [\![(\alpha_1; \alpha_3) \| (\mathtt{sleep}; \beta)]\!](s)\ \mathtt{else}\ [\![(\alpha_2; \alpha_3) \| (\mathtt{sleep}; \beta)]\!](s).$

Generalisations continued

2. Arbitrary atomic actions

- Instead of assignments, programs are built of atomic actions with predefined meanings and permissions (cf. system calls).
- Allows us to model preserving integrity constraints, logging, etc.

3. Arbitrary knowledge states and knowledge acquisitions

Use $A^{\xi,\xi',V}(.,.)$, where $\xi,\xi' \in \mathbf{L}(P)$ instead of $A^{V,T}(.,.)$.

A can realise that $\mathtt{r}(p,A)$ is false as an outcome of sampling $p \notin V$:

$$\left(\begin{array}{c} A^{\xi\wedge p,\xi'\wedge p,V}([\top/p]\Phi,\psi)\wedge \\ A^{\xi\wedge\neg p,\xi'\wedge\neg p,V}([\bot/p]\Phi,\psi)\wedge \\ A^{\xi\wedge\rho,\xi'\wedge\neg\mathtt{r}(p,A),V}(\langle\rho\Rightarrow\varphi_i : i=1,\ldots,l\rangle,\psi) \end{array}\right) \Rightarrow A^{\xi,\xi',V}(\Phi,\psi)$$

where $\rho \rightleftharpoons \exists v_1\ldots\exists v_k(\xi'\wedge\neg\mathtt{r}(p,A))$, $\{v_1,\ldots,v_k\}=V$.

Properties of the modelling framework

Desirable properties cited in the literature include:

- $\surd$ Conditional authorisations.
- $\surd$ Expressibility of joint action.
- $\surd$ Delegation mechanisms. In particular, permission to delegate a privelege should be independent of the privelege.
- $\times$ Support for open and closed systems.
- ($\surd$) Expressibility of administrative policies. They are "one of the most important, although less understood" aspects of access control, and "usually receive little consideration".
- $\surd$ Avoidance of root bottleneck.
- $\times$ Support for fine- and coarse-grained specifications.

The End