

# Access-Control Policies via Belnap Logic

Glenn Bruns and Michael Huth

24 June, 2008

# Motivation and Scope

An access-control policy language defines a predicate on access requests.

# Motivation and Scope

An access-control policy language defines a predicate on access requests.

- Motivation: to solve problems in policy composition
  - ▶ *ad-hoc* collections of policy composition operators
  - ▶ *library problem* of Halpern/Weissman
  - ▶ can't express properties like conflict-freedom directly

# Motivation and Scope

An access-control policy language defines a predicate on access requests.

- Motivation: to solve problems in policy composition
  - ▶ *ad-hoc* collections of policy composition operators
  - ▶ *library problem* of Halpern/Weissman
  - ▶ can't express properties like conflict-freedom directly
- Scope
  - ▶ we don't use a trust logic (where focus is on trust in principals), and we don't support delegation

# Motivation and Scope

An access-control policy language defines a predicate on access requests.

- Motivation: to solve problems in policy composition
  - ▶ *ad-hoc* collections of policy composition operators
  - ▶ *library problem* of Halpern/Weissman
  - ▶ can't express properties like conflict-freedom directly
- Scope
  - ▶ we don't use a trust logic (where focus is on trust in principals), and we don't support delegation

We define a policy language based on Belnap logic.

# Policy Results and Belnap Space

Suppose two predicates used for access-control decisions:

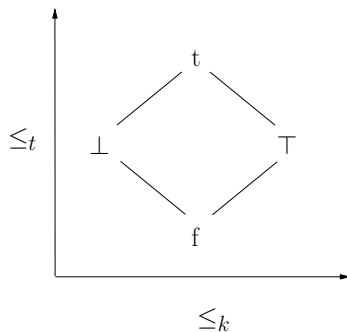
- $\text{grant}(\text{access-request})$
- $\text{deny}(\text{access-request})$

Then a policy outcome becomes a (grant, deny) pair:

- Grant, written  $\mathbf{t}$ , is  $(\text{true}, \text{false})$
- Deny, written  $\mathbf{f}$ , is  $(\text{false}, \text{true})$
- Conflict, written  $\top$ , is  $(\text{true}, \text{true})$
- Unspecified, written  $\perp$ , is  $(\text{false}, \text{false})$

These values can be ordered by *truth* or *information*.

# The Belnap Bilattice



In truth ordering: meet is  $\wedge$ , join is  $\vee$ , negate is  $\neg$

In information ordering: meet is  $\otimes$ , join is  $\oplus$ , negate is  $-$

We also use implication operator  $\supset$  and update operator  $[v \mapsto b]$ .

## Access-control States

*access-control state*  $\mathcal{M} = (A, ap_1^{\mathcal{M}}, ap_2^{\mathcal{M}}, \dots)$

where  $A$  is a set of access requests,

$ap_i^{\mathcal{M}} \subseteq A$  is an atomic access predicate

## Access-control States

*access-control state*  $\mathcal{M} = (A, ap_1^{\mathcal{M}}, ap_2^{\mathcal{M}}, \dots)$

where  $A$  is a set of access requests,

$ap_i^{\mathcal{M}} \subseteq A$  is an atomic access predicate

- Hospital example:  $(A, read\_op, \dots)$ 
  - ▶  $A = HospitalRole \times FileOp \times PatientRecord$
  - ▶ for example:  $(doctor, read, patient123)$
  - ▶  $read\_op$  - holds of accesses in which the file op is “read”

## Access-control States

access-control state  $\mathcal{M} = (A, ap_1^{\mathcal{M}}, ap_2^{\mathcal{M}}, \dots)$

where  $A$  is a set of access requests,

$ap_i^{\mathcal{M}} \subseteq A$  is an atomic access predicate

- Hospital example:  $(A, read\_op, \dots)$ 
  - ▶  $A = HospitalRole \times FileOp \times PatientRecord$
  - ▶ for example:  $(doctor, read, patient123)$
  - ▶  $read\_op$  - holds of accesses in which the file op is “read”
- Library example:  $(A, public\_am, \dots)$ 
  - ▶  $A = LibraryRole \times LibraryAction \times Date$
  - ▶ for example:  $(staff, (remove, book123), 11Aug2008 : 2017)$
  - ▶  $public\_am$  - holds of accesses in which the role is “public” and the time is before noon

## Access-control States

access-control state  $\mathcal{M} = (A, ap_1^{\mathcal{M}}, ap_2^{\mathcal{M}}, \dots)$

where  $A$  is a set of access requests,

$ap_i^{\mathcal{M}} \subseteq A$  is an atomic access predicate

- Hospital example:  $(A, read\_op, \dots)$ 
  - ▶  $A = HospitalRole \times FileOp \times PatientRecord$
  - ▶ for example:  $(doctor, read, patient123)$
  - ▶  $read\_op$  - holds of accesses in which the file op is “read”
- Library example:  $(A, public\_am, \dots)$ 
  - ▶  $A = LibraryRole \times LibraryAction \times Date$
  - ▶ for example:  $(staff, (remove, book123), 11Aug2008 : 2017)$
  - ▶  $public\_am$  - holds of accesses in which the role is “public” and the time is before noon

Each application domain will want a language for access predicates. E.g.:

$$role = public \wedge date < 12 : 00$$

# PBel – A Policy Language based on Belnap Logic

Basic expressions:

- **t** if  $role = doctor \wedge action = delete$
- **f** if  $role = administrator \wedge action = delete$

# PBel – A Policy Language based on Belnap Logic

Basic expressions:

- $\mathbf{t}$  if  $role = doctor \wedge action = delete$
- $\mathbf{f}$  if  $role = administrator \wedge action = delete$

Compositions:

- $(\mathbf{t} \text{ if } role = doctor \wedge action = delete) \oplus (\mathbf{f} \text{ if } role = administrator \wedge action = delete)$
- $(\mathbf{t} \text{ if } role = doctor \wedge action = delete) > (\mathbf{f} \text{ if } role = hospitalStaff \wedge action = delete)$
- $p1 \wedge p2$
- $p[\perp \mapsto false]$

# PBel – A Policy Language based on Belnap Logic

Basic expressions:

- $\mathbf{t}$  if  $role = doctor \wedge action = delete$
- $\mathbf{f}$  if  $role = administrator \wedge action = delete$

Compositions:

- $(\mathbf{t} \text{ if } role = doctor \wedge action = delete) \oplus (\mathbf{f} \text{ if } role = administrator \wedge action = delete)$
- $(\mathbf{t} \text{ if } role = doctor \wedge action = delete) > (\mathbf{f} \text{ if } role = hospitalStaff \wedge action = delete)$
- $p1 \wedge p2$
- $p[\perp \mapsto false]$

$p > q$  is a derived operation that means: use  $p$ 's result if defined, else use  $q$ 's result

# PBel Syntax and Semantics

$p, q ::=$	<i>Policy</i>	
$b \text{ if } ap;$	Basic policy	
$\neg p$	Logical negation	
$p \wedge q$	Logical meet	
$p \supset q$	Implication	
$p \oplus q$	Knowledge join	
$p[v \mapsto q]$	Refinement	$v \in \{\perp, \top\}$

# PBel Syntax and Semantics

$p, q ::=$	<i>Policy</i>
$b \text{ if } ap_i$	Basic policy
$\neg p$	Logical negation
$p \wedge q$	Logical meet
$p \supset q$	Implication
$p \oplus q$	Knowledge join
$p[v \mapsto q]$	Refinement $v \in \{\perp, \top\}$

$$\llbracket b \text{ if } ap_i \rrbracket_{\mathcal{M}}(a) \stackrel{\text{def}}{=} \begin{cases} b & \text{if } a \in ap_i^{\mathcal{M}} \\ \perp & \text{otherwise} \end{cases}$$

$$\llbracket p \wedge q \rrbracket_{\mathcal{M}}(a) \stackrel{\text{def}}{=} \llbracket p \rrbracket_{\mathcal{M}}(a) \wedge \llbracket q \rrbracket_{\mathcal{M}}(a)$$

$$\llbracket p[v \mapsto q] \rrbracket_{\mathcal{M}}(a) \stackrel{\text{def}}{=} \llbracket p \rrbracket_{\mathcal{M}}(a)[v \mapsto \llbracket q \rrbracket_{\mathcal{M}}(a)]$$

...

## Queries on Policies

- $p \leq_t q$       “ $q$  grants at least as much as  $p$ ”
- $p \leq_k q$       “ $q$  is at least as defined as  $p$ ”

Firewall policy example:

$$p_1 = (\mathbf{t} \text{ if } protocol = tcp)$$

$$p_2 = (\mathbf{f} \text{ if } protocol = tcp \wedge port = 80) \oplus (\mathbf{t} \text{ if } protocol = tcp)$$

$$p_3 = (\mathbf{f} \text{ if } protocol = tcp \wedge port = 80) > (\mathbf{t} \text{ if } protocol = tcp) > f$$

We have:

- $p_2 \leq_t p_1$
- $p_1 \leq_k p_2$

# Query Syntax and Semantics

$\phi, \psi ::=$	<i>Query</i>
$p \leq_t q$	Truth ordering ( $p, q$ are PBel expressions)
$p \leq_k q$	Information ordering
$\phi \wedge \psi$	Conjunction
$\neg \phi$	Negation

# Query Syntax and Semantics

$\phi, \psi ::=$	<i>Query</i>
$p \leq_t q$	Truth ordering ( $p, q$ are PBel expressions)
$p \leq_k q$	Information ordering
$\phi \wedge \psi$	Conjunction
$\neg \phi$	Negation

$\mathcal{M} \models p \leq_* q$  *Iff* for all  $a \in A_{\mathcal{M}}$  we have  $\llbracket p \rrbracket_{\mathcal{M}}(a) \leq_* \llbracket q \rrbracket_{\mathcal{M}}(a)$

$\mathcal{M} \models \neg \phi$  *Iff* not  $\mathcal{M} \models \phi$

$\mathcal{M} \models \phi \wedge \psi$  *Iff*  $\mathcal{M} \models \phi$  and  $\mathcal{M} \models \psi$

## More Policy Query Examples

- is policy  $p$  “gap-free”?

$$p \leq_t p[\perp \mapsto \mathbf{f}]$$

- is policy  $p$  “conflict-free”?

$$p \leq_k p[\top \mapsto \mathbf{f}]$$

- is policy  $p'$  more lenient than  $p$  only on ftp packets?

$$p' \leq_t (p \oplus (\mathbf{t} \text{ if } protocol = ftp))$$

# Policy Analysis

We might ask:

- does a query hold on a particular access-control state?

Example: policy  $p = (\mathbf{f}$  if  $is\_doctor$ )  $\oplus$  ( $\mathbf{t}$  if  $is\_nurse$ )

$$A_{\mathcal{M}_1} = \{(doctor, write, record1), (doctor, read, record2), (nurse, read, record2)\}$$

$$A_{\mathcal{M}_2} = \{(doctor, write, record1), (doctor, read, record1), (nurse, read, record2), (admin, read, record3)\}$$

$p$  is gap-free on  $\mathcal{M}_1$ , but not on  $\mathcal{M}_2$

# Policy Analysis

We might ask:

- does a query hold on a particular access-control state?

Example: policy  $p = (\mathbf{f}$  if  $is\_doctor$ )  $\oplus$  ( $\mathbf{t}$  if  $is\_nurse$ )

$$A_{\mathcal{M}_1} = \{(doctor, write, record1), (doctor, read, record2), (nurse, read, record2)\}$$

$$A_{\mathcal{M}_2} = \{(doctor, write, record1), (doctor, read, record1), (nurse, read, record2), (admin, read, record3)\}$$

$p$  is gap-free on  $\mathcal{M}_1$ , but not on  $\mathcal{M}_2$

- does a query hold on all access-control states?

# Policy Analysis

We might ask:

- does a query hold on a particular access-control state?

Example: policy  $p = (\mathbf{f}$  if  $is\_doctor$ )  $\oplus$  ( $\mathbf{t}$  if  $is\_nurse$ )

$$A_{\mathcal{M}_1} = \{(doctor, write, record1), (doctor, read, record2), (nurse, read, record2)\}$$

$$A_{\mathcal{M}_2} = \{(doctor, write, record1), (doctor, read, record1), (nurse, read, record2), (admin, read, record3)\}$$

$p$  is gap-free on  $\mathcal{M}_1$ , but not on  $\mathcal{M}_2$

- does a query hold on all access-control states?
- does a query hold on a specified class of access-control states?

## Answering Policy Analysis Questions

To answer policy analysis questions, we translate queries to FOL on access predicates. Example:

$$p = (f \text{ if } protocol = tcp \wedge port = 80) \oplus (t \text{ if } protocol = tcp)$$
$$\phi_{cf} = p \leq_k p[\top \mapsto \mathbf{f}] \text{ (is } p \text{ conflict-free?)}$$

- translate  $\phi_{cf}$  to  $\forall x.(protocol(x) = tcp \Rightarrow port(x) \neq 80)$ .  
 $p$  is conflict-free at  $\mathcal{M}$  iff  $\mathcal{M}$  satisfies this two-valued formula.
- $p$  would be conflict-free in all access states if the formula for  $\phi_{cf}$  were valid.

The last question can be reduced to a question of propositional satisfiability, and answered with SAT solvers or BDDs.

## Translating Queries to First-Order Logic

Core idea: translate policy expressions in the query to quantifier-free FOL

- $p \uparrow \mathbf{t}$  is the condition under which  $p$  gives  $\mathbf{t}$  or  $\top$
- $p \uparrow \mathbf{f}$  is the condition under which  $p$  gives  $\mathbf{f}$  or  $\top$

## Translating Queries to First-Order Logic

Core idea: translate policy expressions in the query to quantifier-free FOL

- $p \uparrow \mathbf{t}$  is the condition under which  $p$  gives  $\mathbf{t}$  or  $\top$
- $p \uparrow \mathbf{f}$  is the condition under which  $p$  gives  $\mathbf{f}$  or  $\top$

Example. Let

$$p_1 = (\mathbf{f} \text{ if } protocol = tcp \wedge port = 80)$$

$$p_2 = (\mathbf{t} \text{ if } protocol = tcp)$$

## Translating Queries to First-Order Logic

Core idea: translate policy expressions in the query to quantifier-free FOL

- $p \uparrow \mathbf{t}$  is the condition under which  $p$  gives  $\mathbf{t}$  or  $\top$
- $p \uparrow \mathbf{f}$  is the condition under which  $p$  gives  $\mathbf{f}$  or  $\top$

Example. Let

$$p_1 = (\mathbf{f} \text{ if } protocol = tcp \wedge port = 80)$$

$$p_2 = (\mathbf{t} \text{ if } protocol = tcp)$$

Then we get:

$$p_1 \uparrow \mathbf{t} = ff$$

$$p_1 \uparrow \mathbf{f} = (protocol(x) = tcp \wedge port(x) = 80)$$

$$\begin{aligned}(p_1 \oplus p_2) \uparrow \mathbf{t} &= p_1 \uparrow \mathbf{t} \vee p_2 \uparrow \mathbf{t} \\ &= (protocol(x) = tcp)\end{aligned}$$

## Translating Queries to First-Order Logic (con't.)

Translating policy expressions:

$$(b' \text{ if } ap_i) \uparrow b = \begin{cases} ap_i(x) & \text{if } b = b' \\ \text{ff} & \text{otherwise} \end{cases}$$

$$(p \oplus q) \uparrow b = p \uparrow b \vee q \uparrow b$$

$$(p \wedge q) \uparrow \mathbf{f} = p \uparrow \mathbf{f} \vee q \uparrow \mathbf{f}$$

$$(p \wedge q) \uparrow \mathbf{t} = p \uparrow \mathbf{t} \wedge q \uparrow \mathbf{t}$$

...

Translating queries:

$$\|p \leq_t q\| = \forall x. ([q \uparrow \mathbf{f} \supset p \uparrow \mathbf{f}] \wedge [p \uparrow \mathbf{t} \supset q \uparrow \mathbf{t}])$$

$$\|p \leq_k q\| = \forall x. ([p \uparrow \mathbf{f} \supset q \uparrow \mathbf{f}] \wedge [p \uparrow \mathbf{t} \supset q \uparrow \mathbf{t}])$$

...

# Static Analyses for Conflict-Freedom and Gap-Freedom

An inference system for conflict-freedom:

$$\frac{}{CF(b \text{ if } ap_i)} \quad \frac{CF(p)}{CF(\neg p)} \quad \frac{CF(p) \quad CF(q)}{CF(p \wedge q)}$$
$$\frac{CF(q)}{CF(p \supset q)} \quad \frac{CF(q)}{CF(p[\top \mapsto q])} \quad \frac{CF(p) \quad CF(q)}{CF(p[\perp \mapsto q])}$$

If the judgement  $CF(p)$  can be inferred, then  $p$  is conflict-free.

Example:

$$(\mathbf{f} \text{ if } protocol = tcp \wedge port = 80) > (\mathbf{t} \text{ if } protocol = tcp)$$

is conflict-free.

Any policy not containing  $\oplus$  is conflict-free – thus  $\oplus$  cannot be expressed using the other operators of PBel.

# Expressiveness

- If every access request in a access-control state can be distinguished by access predicates, then any function from access requests to the Belnap space can be expressed by a policy.

# Expressiveness

- If every access request in a access-control state can be distinguished by access predicates, then any function from access requests to the Belnap space can be expressed by a policy.
- If not, then any function can be expressed modulo the distinctions made by the access predicates.

# Expressiveness

- If every access request in a access-control state can be distinguished by access predicates, then any function from access requests to the Belnap space can be expressed by a policy.
- If not, then any function can be expressed modulo the distinctions made by the access predicates.
- It is difficult to express in PBel that a policy should grant on one access request if it grants on some related access request, because PBel interprets Belnap operators on policies in a pointwise fashion.

## Conclusions and Future Work

- PBel is a policy language based on Belnap logic that provides a small, natural set of policy composition operators.

## Conclusions and Future Work

- PBel is a policy language based on Belnap logic that provides a small, natural set of policy composition operators.
- PBel policy analysis is accomplished with propositional model checking and satisfiability checking.

# Conclusions and Future Work

- PBel is a policy language based on Belnap logic that provides a small, natural set of policy composition operators.
- PBel policy analysis is accomplished with propositional model checking and satisfiability checking.
- We are currently studying extensions to PBel to support richer features, such as delegation and closure.