Access Control via Belnap Logic: Intuitive, Expressive, and Analyzable Policy Composition

Michael Huth (joint work with Glenn Bruns, Bell Labs)
Outline of Access Control talk

• Motivation
• Belnap Logic
• Core Policy Language
• Expressiveness of Core Language
• Policy Analysis
• Conclusions
Access control in IT systems increasingly relies on ability to compose policies.

Composition framework should

- be intuitive, formal, expressive, and analyzable
- be independent of application domains but extendable to such domains
- support change management, separation of concerns, and reuse

We develop here such a framework based on Belnap Logic.

Belnap Logic used in the past for reasoning in Artificial Intelligence.
Belnap Logic

In the 1970ies, Belnap suggested the use of a four-valued logic

- Ordinary truth values for truth and falsity
- A third truth value that expresses lack of knowledge
- And a fourth truth value that expresses inconsistent knowledge

He developed a semantics and a sound and complete Hilbert style proof system for this logic
Belnap logic extends naturally to first- and higher-order logics

Key idea for us: Belnap’s evidence-based notion of truth, e.g.
- conjunction of “don’t know” and “false” is “false”
- but conjunction of “don’t know” and “true” is “don’t know”
Belnap Logic: four values as composition of two

\{..\} collects results of policies p and q, e.g. \{t,f\} as conflict

(\ldots\ldots\ldots) asks ‘‘does single policy p (grant?,deny?)’’, e.g. (t,t) means p grants and denies

Both interpretations captured abstractly
Belnap space \((4, \leq_t, \leq_k, \neg)\)

Belnap bilattice over \(4 = \{t, f, \top, \bot\}\)

Axioms:

\[x \leq_t y \Rightarrow \neg y \leq_t \neg x,\]
\[x \leq_k y \Rightarrow \neg x \leq_k \neg y, \text{ and}\]
\[\neg \neg x = x.\]

Truth negation \(\neg\) swaps denials and grants, and leaves other two values fixed.

x-axis: knowledge ordering

y-axis: truth ordering
Belnap space functionally complete

Implication:

\[ a \supset b = b \text{ if } a \in \{ t, T \} \]
\[ a \supset b = t \text{ otherwise} \]

Conjunction:

\[ \land \text{ meet (aka infimum) in truth ordering} \]
Core language PBel, pronounced “pebble”

\[ rp ::= \text{Request Predicate} \]

\[ a \quad \text{Atomic} \]

\[ \text{true} \quad \text{Truth} \]

\[ \text{false} \quad \text{Falsity} \]

\[ p, p' ::= \text{Policy} \]

\[ b \text{ if } rp \quad \text{Basic policy} \]

\[ \top \quad \text{Conflict} \]

\[ \neg p \quad \text{Logical negation} \]

\[ p \land p' \quad \text{Logical meet} \]

\[ p \supset p' \quad \text{Implication} \]

\[ \top \text{ overloaded: denotes policy and denotes element of Belnap space} \]

Atomic request predicates \( a \) denote sets of access requests.

\[ b \in \{t, f\} \]
Syntactic sugar

PBel is a core language, similar to byte code for higher-level languages.

Convenient policy composition operators map (syntactically and semantically) into this core language, e.g.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Syntactic Representation</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>t if true</td>
<td></td>
</tr>
<tr>
<td>f</td>
<td>¬t</td>
<td></td>
</tr>
<tr>
<td>p ∨ q</td>
<td>¬(¬p ∧ ¬q)</td>
<td></td>
</tr>
<tr>
<td>p ⊕ q</td>
<td>(p ∧ T) ∨ (q ∧ T) ∨ (p ∧ q)</td>
<td></td>
</tr>
<tr>
<td>p[f → q]</td>
<td>p ∨ (¬(p ∨ ¬p) ∧ q)</td>
<td></td>
</tr>
<tr>
<td>p[⊥ → q]</td>
<td>p ⊕ (¬(p ⊕ ¬p) ⊕ q)</td>
<td></td>
</tr>
<tr>
<td>p if rp</td>
<td>p ⊗ ((t if rp) ⊕ (f if rp))</td>
<td></td>
</tr>
<tr>
<td>p↓</td>
<td>p[T → f][⊥ → f]</td>
<td></td>
</tr>
</tbody>
</table>

Abbreviation for priority composition: \( p > q = p[⊥ → q] \)
Models

Access-control model \( M \)

consists of non-empty set of requests \( R_M \)

and interpretations of request predicates: \( rp^M \subseteq R_M \)

\[
\text{true}^M = R_M \text{ and } \text{false}^M = \{\}
\]

Example:

• set of requests as triples of form (subject, object, action, context)
• interpretation of atom manager is all triples whose subjects are managers
• interpretation of atom lowThreat is all triples whose context represents a low threat level
Semantics

\[
[b \text{ if } rp]_M(r) = \begin{cases} 
b & \text{if } r \in rp^M \\
\bot & \text{otherwise} \end{cases}
\]

\[
[\top]_M(r) = \top
\]

\[
[\neg p]_M(r) = \neg[p]_M(r)
\]

\[
[p \land q]_M(r) = [p]_M(r) \land [q]_M(r)
\]

\[
[p \supset q]_M(r) = [p]_M(r) \supset [q]_M(r)
\]

Meaning of policy p in model M maps requests r to element of Belnap space

Key point: policy composition is pointwise extension of Belnap operators
Support for composite request predicates

Propositional logic structure on request predicates compiles into PBel, e.g.

\((t \text{ if } (\text{Manager} \land \text{OnDuty} \land \neg \text{Weekend} \land \text{ReadPDF})) > f\)

is translated into PBel with function \(T\) below, for semantics on the left:

\[
\begin{align*}
\models a & \models_M = a^M \\
\models \text{true} & \models_M = R_M \\
\models \text{false} & \models_M = \{\} \\
\models \neg cp & \models_M = R_M \setminus \{ cp \} \\
\models cp \land cp' & \models_M = \{ cp \} \cap \{ cp' \} \\
\models cp \lor cp' & \models_M = \{ cp \} \cup \{ cp' \} \\
T(t \text{ if } \neg cp) & = T(t \text{ if } cp) \supset \bot \\
T(t \text{ if } cp \land cp') & = T(t \text{ if } cp) \land T(t \text{ if } cp') \\
T(t \text{ if } cp \lor cp') & = \neg(T(f \text{ if } cp) \land T(f \text{ if } cp')) \\
T(f \text{ if } \neg cp) & = \neg(T(t \text{ if } cp) \supset \bot) \\
T(f \text{ if } cp \land cp') & = \neg(T(t \text{ if } cp) \land T(t \text{ if } cp')) \\
T(f \text{ if } cp \lor cp') & = T(f \text{ if } cp) \land T(f \text{ if } cp')
\end{align*}
\]
Policy $p$ is **conflict-free** if it never returns $\top$ for any request in any model.

**Safe sublanguages**

Sublanguage in which all and only conflict-free policies can be written:

$$p, q ::= \text{Conflict-free policy}$$

<table>
<thead>
<tr>
<th>$b \text{ if } rp$</th>
<th>Basic policy</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\bot$</td>
<td>Gap</td>
</tr>
<tr>
<td>$\neg p$</td>
<td>Logical negation</td>
</tr>
<tr>
<td>$p \land q$</td>
<td>Logical meet</td>
</tr>
<tr>
<td>$r \supset q$</td>
<td>Implication</td>
</tr>
<tr>
<td>$p \lor q$</td>
<td>Logical Join</td>
</tr>
<tr>
<td>$p \otimes q$</td>
<td>Knowledge meet</td>
</tr>
</tbody>
</table>

| $r[\top \mapsto p]$ | Conflict resolution |
| $p[v \mapsto q]$   | Sequential composition |
| $p \text{ if } rp$ | Generalized basic policy |
| $p : q$            | Guard connective |
| $r \downarrow$     | Pessimistic wrapper |
| $r \uparrow$       | Optimistic wrapper |

*Boldface $r$ denotes any policy expression of PBel*
Policy $p$ is **gap-free** if it never returns $\bot$ for any request in any model.

Sublanguage in which all and only gap-free policies can be written:

$$p, q ::= \quad \text{Gap-free policy}$$

- $b \text{ if true}$  \quad \text{Gap-free basic policy}
- $\top$  \quad \text{Conflict}
- $\neg p$  \quad \text{Logical negation}
- $p \land q$  \quad \text{Logical meet}
- $r \supset q$  \quad \text{Implication}
- $p \lor q$  \quad \text{Logical join}
- $r \oplus q$  \quad \text{Left knowledge join}

$$p \oplus r$$  \quad \text{Right knowledge join}

$$r[\bot \rightarrow p]$$  \quad \text{Gap resolution}

$$p[v \leftrightarrow q]$$  \quad \text{Sequential composition}

$$p \text{ if true}$$  \quad \text{Generalized basic policy}

$$r \downarrow$$  \quad \text{Pessimistic wrapper}

$$r \uparrow$$  \quad \text{Optimistic wrapper}

**Boldface** $r$ denotes any policy expression of PBel
Safe sublanguage for conclusive decisions

Sublanguage in which only (and all) policies can be written that are gap-free and conflict-free:

\[ p, q ::= \begin{array}{l}
    Conclusive policy \\
    \quad b \text{ if true} \quad \text{Gap-free basic policy} \\
    \quad \neg p \quad \text{Logical negation} \\
    \quad p \land q \quad \text{Logical meet} \\
    \quad r \supset q \quad \text{Implication} \\
    \quad p \lor q \quad \text{Logical join}
\end{array} \]

\[ p[u \mapsto q] \quad \text{Sequential composition} \\
\quad p \text{ if } rp \quad \text{Generalized basic policy} \\
\quad r \downarrow \quad \text{Pessimistic wrapper} \\
\quad r \uparrow \quad \text{Optimistic wrapper} \]

Boldface \( r \) denotes any policy expression of PBel
Retrofitting policies

Recall abbreviation for priority composition: \[ p > q = p[\perp \leftrightarrow q] \]

Exceptional override: \((f \text{ if } rp_{exc}) > p\)

Behaves like policy \(p\) except at exceptional request set \(rp_{exc}\) at which it denies

Exclusive rights and exclusive prohibitions: \(((t \text{ if } rp_1) \oplus (f \text{ if } rp_2)) > p\)

Behaves like \(p\) except at two request sets that encode absolute rights and absolute prohibitions (respectively)

Retrofitted policy conflict-free iff two request sets are disjoint
Composing request predicates as policies

\((t \text{ if } ChW) \land (t \text{ if } RegisteredAnalyst) > f\)

*Might model that an access is granted if*

• the requester is a registered analyst
• and if the request is compliant with a Chinese Wall policy

*Otherwise, the request is denied.*

*Note: ChW is a request predicate that presumably stems from a policy.*

*PBel supports such demotions of policies to predicates* (see analysis part below).
Policy function for model $\mathcal{M}$ is total function $f : R_\mathcal{M} \to 4$

Policy $p$ expresses policy function $f$ if $\llbracket p \rrbracket_\mathcal{M} = f$

Policy functions of form $\llbracket p \rrbracket_\mathcal{M} = f$ have the same output for requests that cannot be distinguished by any request predicate in $p$

All functions of that form are expressible as meanings of policies in PBel

These results customize to the safe sublanguages for gap-free, conflict-free, and conclusive policies (not shown in this talk)
Expressing data-independent policy functions

Let $R$ be a set of request predicates, e.g. those occurring in a policy $p$.

On each model $\mathcal{M}$ we define equivalence relation $\equiv^R_{\mathcal{M}}$ to be \{(r, r') \in R_{\mathcal{M}} \times R_{\mathcal{M}} \mid \forall a \in R: r \in a_{\mathcal{M}} \iff r' \in a_{\mathcal{M}}\}\.

A policy function $f : R_{\mathcal{M}} \to 4$ is data-independent for $R$ in $\mathcal{M}$ iff $r \equiv^R_{\mathcal{M}} r'$ implies $f(r) = f(r')$.

We write $\text{PBel}^R$ for the set of PBel policies that contain only atoms from $R$.
In particular, $\text{PBel}^{\text{AP}}$ equals $\text{PBel}$.

E.g. $(t \text{ if } ChW) \land (t \text{ if } RegisteredAnalyst) > f$ has four equivalence classes

Result:

For each $p$ in $\text{PBel}^R$, policy function $[p]_{\mathcal{M}}$ is data-independent for $R$ in $\mathcal{M}$.
Conversely, let $f$ be a policy function that is data-independent for $R$ in $\mathcal{M}$ for finite set $R_{\mathcal{M}}$. Then there is some $p$ in $\text{PBel}^R$ with $f = [p]_{\mathcal{M}}$. 
Proof that data-independent function $f$ is expressible

Policy $p$ expressing policy function $f$ is knowledge join of a grant and a denial part:

$$p = p_t \oplus p_f$$

Each part is the $n$-ary knowledge join of (negations of) policies $p_r$

$$p_t = \sum_{r \in R_M \mid t \leq k \cdot f(r)} p_r$$
$$p_f = \sum_{r \in R_M \mid f \leq k \cdot f(r)} \neg p_r$$

Each building block $p_r$ is a characteristic function that grants on the equivalence class $\equiv^R_M$ of request $r$ and is undefined otherwise:

$$p_r = (\bigwedge_{a \in R \mid r \in a \cdot M} t \text{ if } a) \wedge (\bigwedge_{a \in R \mid r \not\in a \cdot M} (t \text{ if } a) \supset \bot)$$
Expressiveness of PBel: policy composition

Example: majority vote of three policies should make as decision the majority of decisions

\[ G(p_1, p_2, p_3) = (p_1 \land p_2) \lor (p_1 \land p_3) \lor (p_2 \land p_3) \]

Our pointwise composition means that any such function \( G \) is determined by a function

\[ g : 4^n \rightarrow 4 \]

\( G(p_1,\ldots,p_n) \) at request \( r \) decides \( g(v_1,\ldots,v_n) \) where \( v_i \) is decision of \( p_i \) on \( r \)
All composition functions are expressible

Free algebra \( \mathcal{A} \)
generated from operators \( \{ \top, \neg, \land, \lor \} \)
and variables \( X_1, X_2, \ldots \)

Result:

Let \( n \geq 0 \) and \( g \in 4^n \rightarrow 4 \)
Then there is a term \( t_g \in \mathcal{A} \) such that

\[
[t_g(p_1, \ldots, p_n)]_\mathcal{M}(r) = g([p_1]_\mathcal{M}(r), \ldots, [p_n]_\mathcal{M}(r))
\]

(\( \forall p_i \in \text{PBel}, \mathcal{M}, r \in \text{R}_\mathcal{M} \))

Example: term for knowledge join \( \oplus \) is

\[
\neg(\neg(X_1 \land \top) \land \neg(\neg((\top \land X_2) \land \neg(X_1 \land X_2))))
\]
We develop a simple query language.

Many important policy analyses are expressible as queries in this language.

Here, we ask whether queries hold in all models: validity checking.

Validity checking is appropriate, e.g. for gap and conflict analysis.

Queries can be assume-guarantee implications whose antecedents encode domain-specific or other assumptions.

This policy analysis reduces to validity checking of propositional logic.
Query Language

\[ cp, cp' ::= \text{Composite Request Predicate} \]

\[
\begin{array}{ll}
  a & \text{Atomic} \\
  \text{true} & \text{Truth} \\
  \text{false} & \text{Falsity} \\
  \neg cp & \text{Negation} \\
  cp \land cp' & \text{Conjunction} \\
  cp \lor cp' & \text{Disjunction}
\end{array}
\]

\[ \phi, \phi' ::= \text{Query} \]

\[
\begin{array}{ll}
  p \leq_t q & \text{Policy refinement in truth ordering} \\
  p \leq_k q & \text{Policy refinement in information ordering} \\
  \alpha \Rightarrow \phi & \text{Assume-guarantee query} \\
  \phi \land \phi' & \text{Conjunction}
\end{array}
\]

where assumption \( \alpha \) ranges over \text{composite} request predicates
**Query Examples**

Policy q is more defined and more permissive than p

\[(p \leq_k q) \land (p \leq_t q)\]

Policy q is more defined but less permissive than p

\[(p \leq_k q) \land (q \leq_t p)\]

Policies p and q are equivalent

\[(p \leq_t q) \land (q \leq_t p)\]

Policy p has no gaps

\[p \leq_t p[\perp \mapsto \text{f}]\]

Policy p has no conflicts

\[p \leq_k p[\top \mapsto \text{f}]\]
Query Semantics

\[ M \models p \leq_k q \iff \text{for all } r \in R_M \text{ we have } [p]_M(r) \leq_k [q]_M(r) \]
\[ M \models p \leq_t q \iff \text{for all } r \in R_M \text{ we have } [p]_M(r) \leq_t [q]_M(r) \]
\[ M \models \alpha \Rightarrow \phi \iff \alpha \not\models M \lor M \models \phi \]
\[ M \models \phi \land \psi \iff M \models \phi \text{ and } M \models \psi \]

Atomic queries interpreted \textit{universally}: policy \( p \) is below policy \( q \) for all requests

Assume-Guarantee Implications modeled \textit{universally}:

If all requests of the model satisfy assumption, then the guarantee query has to be true in the model

Conjunction has standard interpretation
Query Analysis

For each query $\phi$ we generate a composite request predicate $C(\phi)$ such that

Query $\phi$ is valid in all models

Iff

$C(\phi)$ is valid when interpreted as formula in propositional logic

Definition of $C(\phi)$ proceeds in two steps:

1. Capture constraints for grants and denials of policies contained in query
2. Capture the logical structure of the query in terms of these policy constraints
Constraints for PBel policies

Each request $r$ in model $M$ determines model of propositional logic:

$$\rho^M_r(a) = t \text{ if } r \in a^M \quad \rho^M_r(a) = f \text{ if } r \notin a^M$$

Semantics of that model matches that of request model $M$: $\rho^M_r$ satisfies those composite request predicates that contain $r$ in their interpretation.

Define, below, propositional logic formula $p \uparrow b$ such that

$$\rho^M_r \models p \uparrow b \iff b \leq_k [p]^M(r)$$

$$(b' \text{ if } rp) \uparrow b = \begin{cases} rp & \text{if } b = b' \\ \text{false} & \text{otherwise} \end{cases}$$

$$\top \uparrow b = \text{true}$$

$$(p \land q) \uparrow f = p \uparrow f \lor q \uparrow f$$

$$(p \lor q) \uparrow f = p \uparrow t \land q \uparrow f$$

$$(\neg p) \uparrow b = p \uparrow \neg b$$

$$(p \land q) \uparrow t = p \uparrow t \land q \uparrow t$$

$$(p \lor q) \uparrow t = \neg(p \uparrow t) \lor q \uparrow t$$
Constraints for Queries

Query for knowledge ordering uses that truth and falsity are prime elements of the distributive lattice in the knowledge ordering

\[ C'(p \leq_k q) = (p \uparrow f \rightarrow q \uparrow f) \land (p \uparrow t \rightarrow q \uparrow t) \]
\[ C'(\alpha \Rightarrow \phi) = \alpha \rightarrow C'(\phi) \]

Query for assume-guarantee reasoning translates this into a propositional implication (as done in assume-guarantee reasoning for linear-time temporal logic)

Query for truth ordering uses characterization of truth ordering in terms of knowledge ordering

Query for conjunction is interpreted compositionally (sound for validity checking)

\[ C'(p \leq_t q) = (q \uparrow f \rightarrow p \uparrow f) \land (p \uparrow t \rightarrow q \uparrow t) \]
\[ C'(\phi \land \psi) = C'(\phi) \land C'(\psi) \]
Example Policy Analysis

\[
p = (t \text{ if } rd) \oplus (f \text{ if } wr) \quad \quad q = p[\top \mapsto f]
\]

Valid query (assumes that no read action is also a write action):

\[
\neg(rd \land wr) \Rightarrow (p \leq_t q)
\]

Propositional constraints for policies:

\[
p \uparrow t = rd \lor \text{false} = rd
\]
\[
p \uparrow f = \text{false} \lor wr = wr
\]
\[
q \uparrow t = p \uparrow t \land (\neg(p \uparrow f) \lor f \uparrow t) = rd \land (\neg wr \lor \text{false}) = rd \land \neg wr
\]
\[
q \uparrow f = p \uparrow f \land (\neg(p \uparrow t) \lor t \uparrow t) = wr \land (\neg rd \lor \text{true}) = wr
\]

Propositional constraint for above query is equivalent to valid formula

\[
C(\neg(rd \land wr) \Rightarrow p \leq_t q) = \neg(rd \land wr) \rightarrow (q \uparrow f \rightarrow p \uparrow f) \land (p \uparrow t \rightarrow q \uparrow t)
\]
\[
= \neg(rd \land wr) \rightarrow (wr \rightarrow wr) \land (rd \rightarrow (rd \land \neg wr))
\]
Some Related Work

- **Halpern and Weissman** 2003: policies specified in first-order logic, access granted if formula logically entails formal permission predicate
- **XACML** standard: has no formal semantics, its semantic values and policy combination algorithms can be interpreted within PBel for suitable interpretation
- **Craven et al.** 2009: policy analysis that takes account of obligations, authorizations, and system state
- **Ni et al.** 2009: D-algebras as functionally complete value spaces with algebraic operators, result spaces rather ad-hoc in nature
- **Bauer et al.** 2005: Polymer, access-control language for untrusted Java applications, policy is query method that returns one of six values for code execution request
- **Moffett and Sloman** 1994: early work on policy conflict analysis
- **Ribeiro et al.** 2001: access-control language SPL, three-valued, can be cleanly embedded into PBel
Things we did but didn’t mention here

- Support for attribute language for request predicates
- Clean encoding of policy language SPL in PBel
- Expressiveness results specialized to safe sub-languages, both for policy functions and for policy composition
- Symbolic query analysis, where policies are parameters
- Methods and interface specifications
- Request mappings, e.g. "grant read access whenever write access is given"
Conclusions

We developed an access-control policy language based on

- abstract request predicates that encapsulated domain-specific aspects of sets of access requests
- the 4-valued Belnap bilattice whose operators were extended pointwise to our language

This gave us a very expressive core language over which common policy combination idioms can be expressed.

The core language (and so any idioms compiling into it) has a simply query analysis that supports many of the desired policy analyses and reduces them to validity checking of propositional logic.
Belnap, N. D.
A useful four-valued logic.
In “Modern Uses of Multiple-Valued Logic”, J. M. Dunn & G. Epstein (eds)
D. Reidel, Dordrecht, pages 8--37, 1977

Arieli, O. and Avron, A.
The value of the four values.
Thanks to Jason Crampton and Daniel Dantas.