

A Simple Language For Policy Composition and Analysis

Michael Huth

`imperial.ac.uk/quads/`

Schloß Dagstuhl, 23 July 2008



Acknowledgment

Glenn Bruns, Bell Labs, Naperville, Illinois



Outline

Motivation

A Simple Policy Language

Countering Insider Threats

Conclusion



Motivation



My relevant background

- ▶ Lived in three territories so far: Germany (26 years), USA (11), and UK (9).
- ▶ Mostly taught and did research in mathematics, computer science, software engineering, engineering ethics.
- ▶ Currently do research in software verification, software modeling, and access control.
- ▶ Do some consulting in security, notably identity management.
- ▶ Goal: maximize impact of – while minimizing use of – formal methods.



Context of work reported here

Policies:

- ▶ become more important in many domains, not just security
- ▶ may not be explicitly documented
- ▶ may be too general, too specific, too ambiguous
- ▶ need to be modifiable, comparable to other policies
- ▶ need to support different degrees of granularity
- ▶ etc.

⇒ A simple but expressive target language for policies that supports their elaboration, exploration, and analysis should have value.

Some requirements for policy language

- ▶ policies can specify single aspects, may be silent on some requests
- ▶ intuitive yet expressive composition of such policies
- ▶ efficient policy analysis (think SAT solvers or BDDs)
- ▶ gap and conflict analysis supported
- ▶ ability to analyze policy hierarchies and change impact
- ▶ ability to specialize or partially evaluate policies
- ▶ provide clean interface for specific application domains (e.g. role hierarchies)



What does this have to do with Countering Insider Threats?

- ▶ Policies normally don't make explicit who is an insider.
- ▶ Policies normally don't support impact analysis of insider abuse.
- ▶ Policies normally adapt to threats without pondering their degree of insiderness.
- ▶ If insiders are those with special access or trust, we may think of insiders as being able to have granted special requests under special conditions.

⇒ So a policy language that can partially evaluate or specialize policies with respect to specific sets of requests may address the issues above.



A Simple Policy Language



PolCore: a core policy language

dec ::= (Decision)
grant
deny

reqs ::= (Requests)
ff Falsity
tt Truth
reqsAtom Atom
!reqs Neg.
reqs & reqs Conj.
reqs || reqs Disj.
pol.dec Demote

pol ::= (Policy)
dec Constant
pol if reqs Promote
pol merge pol Merge

Example policy (Halpern & Weissman)

Consider policy

$$p = p1 \text{ merge } p2 \text{ merge } p3$$

Its sub-policies each model an aspect of a campus policy:

- ▶ policy $p1$ says “faculty has permission to assign grades”
- ▶ policy $p2$ says “students must not assign grades”
- ▶ policy $p3$ says “non-faculty has permission to enroll in courses”.

We formalize these policies in PolCore:

- ▶ $p1 = \text{grant if faculty \& grades \& assign}$
- ▶ $p2 = \text{deny if student \& grades \& assign}$
- ▶ $p3 = \text{grant if !faculty \& courses \& enroll}$

Queries and their evaluation

We want to know what p decides on the query "Can students enroll in courses?"

- ▶ $p.\text{grant}$ expresses those requests that policy p grants
- ▶ $p.\text{deny}$ expresses those requests that policy p denies
- ▶ fundamental to this approach: $p.\text{grant}$ is logical negation of $p.\text{deny}$ if, and only if, policy p has no gaps or conflicts

In PolCore expressions $p.\text{ol.dec}$ are simply expanded into expressions of the form reqs .



Expansion of `pol.dec`

$\text{Expd}(\text{dec.dec}) = \text{tt}$

$\text{Expd}(\text{dec.!dec}) = \text{ff}$

$\text{Expd}((\text{pol if reqs}).\text{dec}) = \text{Expd}(\text{reqs}) \ \& \ \text{Expd}(\text{pol.dec})$

$\text{Expd}((\text{p1 merge p2}).\text{dec}) = \text{Expd}(\text{p1.dec}) \ \| \ \text{Expd}(\text{p2.dec})$

Where $\text{Expd}(\text{reqs})$ only expands sub-terms of form `pol.dec` recursively.



Revisiting campus policy p

- ▶ $\text{Expd}(p.\text{grant}) = (\text{faculty} \& \text{grades} \& \text{assign})$
 $\parallel (!\text{faculty} \& \text{courses} \& \text{enroll})$
- ▶ "Can students enroll in courses?" turns into
 $\text{Expd}(p.\text{grant}) \& \text{student} \& \text{courses} \& \text{enroll}$
- ▶ making domain assumption that courses are never grades,
 assignments are never enrollments, this simplifies to
 $\text{student} \& !\text{faculty} \& \text{courses} \& \text{enroll}$, i.e. to
 $!\text{faculty}$
- ▶ so a student can enroll in courses iff she is not a faculty
 member
- ▶ similarly, using $\text{Expd}(p.\text{deny})$ we conclude that no student
 is being denied to enroll in courses: exposes policy gap for
 students who are also faculty members



Policy analysis through SAT solving

- ▶ policy p_{ol} has no gaps if the expansion of $p_{ol}.grant \parallel p_{ol}.deny$ is valid
- ▶ policy p_{ol} has no conflict if the expansion of $!(p_{ol}.grant \ \& \ p_{ol}.deny)$ is valid
- ▶ policy p_2 refines policy p_1 if the expansion of $(p_1.grant \rightarrow p_2.grant) \ \& \ (p_1.deny \rightarrow p_2.deny)$ is valid
- ▶ policy p_{ol} consistently blacklists set of requests $reqs$ iff the expansion of $reqs \rightarrow p_{ol}.deny$ is valid
- ▶ etc.

Defining policy combinators

- ▶ just define `po1.dec` when `po1` is the desired policy combination
- ▶ illustration: priority composition `p1 > p2` makes those decisions that `p1` makes; if `p1` is silent, it makes whatever decision that `p2` may make
- ▶ `(p1 > p2).dec` is defined to be
`p1.dec || (!p1.deny & !p1.grant & p2.dec)`
- ▶ implementation of `p1 > p2` in PolCore:

```
(grant if (p1 > p2).grant) merge (deny if (p1 > p2).deny)
```

Countering Insider Threats



Insiders as sets of requests

- ▶ if insider is one who has special privileges, we may classify certain requests as coming from an insider
- ▶ a faculty who wants to assign a grade is an insider, as is a student who wants to enroll in a course
- ▶ e.g. we may define `insider` to be

```
(faculty & grades & assign) ||  
(student & courses & enroll)
```

Insider policies

- ▶ if policy p is meant to only apply to insiders, we can check this by asking whether decisions of that policy only apply to insiders
- ▶ e.g. if we want that grants apply only to insiders, we check validity of expansion of

`p.grant → insider`

- ▶ note that policy p may represent all kinds of trust, e.g. it may have form

`grant if WriteFile & trustedRequest`



Outsiders

- ▶ we present an outsider in the same manner as an insider, as an expression `outsider` denoting a set of requests
- ▶ using expressions `outsider` and `insider` we can derive other sets of requests of interest, e.g.:
- ▶ `pol.grant & insider & outsider` denotes those requests that policy `pol` grants to insiders and to outsiders
- ▶ `pol.grant & pol.deny & insider` denotes those requests on which policy `pol` is inconsistent for insiders
- ▶ `(grant if pol.grant & insider)` modifies policy `pol` so that it only grants, and only grants to insiders



Degree of "insiderness"

- ▶ `rogueRequest` denotes set of requests that should absolutely not be granted by a policy `p`
- ▶ `insider1` and `insider2` two insiders
- ▶ if the expansion of

`!(insider1 & rogueRequest & pol.grant)`

is valid but

`!(insider2 & rogueRequest & pol.grant)`

isn't, then `insider1` is weaker than `insider2` for this particular threat

Conclusion



Conclusion

- ▶ We should have a special interest group on policy elaboration, specification, and analysis with a focus on insiders and their threats.

