# Policy Languages for User Managed Access

Michael Huth
Imperial College London

25 February 2011

# Acknowledgements

Glenn Bruns
Bell Labs, Naperville
collaborator

Jason Crampton
Information Security Group, Royal Holloway
collaborator

Ben Laurie
Google
His 2009 note “Access Control (v0.1)”

# Aims of my talk

We study policy-based user managed access control (UMA):

- we identify its opportunities and challenges
- we discuss its language design and analysis issues
- we hint at some solution approaches to these challenges.

# Outline of lecture

Non-Technical Drivers

Domain-Specific Languages

Design Example

Outlook

# Non-Technical Drivers

# Emergent user behavior

- arguably the largest non-technical driver for Web 2.0
- Web 2.0 tools' fate: get adopted by users or perish
- but Web 2.0 tools can also manipulate adopters
- we discuss briefly relevant issues

# Post-Twitter users

- interested in neither platforms nor operating systems

Access control needs to be independent from the choice of platform or system.

# Post-Twitter users

- interested in cheap, always available services

Access control needs to be resilient to failure.

Users may rate availability higher than security!

# Post-Twitter users

- change services periodically or frequently

Access control needs

- to adjust to such changes
- support for management of resource lifecycle.

# Post-Twitter users

- want to manage access at varying depth and rate

Access control at

- various levels of granularity
- varying rates of change: from lazy bum to control freak

# Post-Twitter users

- have short attention spans

Easy appoaches win.

Access control automatically computed from general user goals and principles?

E.g. Design Synthesis.

# Post-Twitter users

- expect to use Web 2.0 technologies at work place

Access control needs to enforce organizational rules but be consistent with Web 2.0 usage models.

Web 2.0 may be disruptive technology in terms of security or established work practices.

# Post-Twitter users

- can program/configure simple interfaces

Access control needs to be simple yet expressive and extensible.

# Post-Twitter users

- like to think about access control in different ways

Access control in terms of

- degrees of trust
- groups, such as friends and “near” friends as in `mixpool.com`
- capabilities

# How do policies for access control fit in here?

- User may opt for hosts' or apps' default policies
- But hosts have vested and conflicting interests
- User may express her own access control
-
    - formalize expression without compromising user intent
    - bridging informal and formal worlds is hard (e.g. requirements capture in Software Engineering)
- Even if users and tools have set of rules, composition of such sets cannot always rely on union
- Solution:
    - formal policies as modular programs with scope
    - enables analysis, implementation, and optimization

Imperial College
London

# Adaptor between user and UMA architecture

- user frontend: manage access via views, edits, analyses
- “middleware” as glue between frontend and systems
- domain-specific languages (DSLs) and their analyses

User
view
view
view
DSLs
Analyses
model based
UMA
policies

# Domain-Specific Languages

# The next 700 access-control policy languages

- 2009 paper "The next 700 access control models or a unifying meta-model?" by Steve Barker
- based on influential 1966 paper by Peter Landin:
  The next 700 programming languages
- we want to investigate use of domain-specific languages (DSLs) for UMA policies
- these DSLs may be "generic" in that they have plugs for administration, composition, and application specifics
- DSLs may be embedded (e.g. Scala), new or mix thereof
- e.g. capabilities as objects controlling "normal" objects

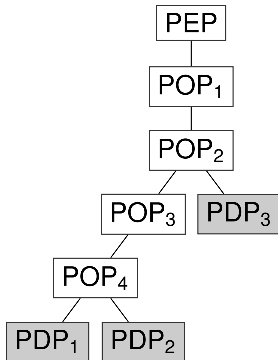
# Design Principles

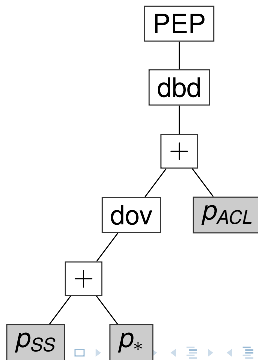
- programming language design is mature research area
- offers proven principles
- general consensus is
  (1) to build small core language
  (2) extend core language with syntactic sugar ...
  (3) ... and with important, orthogonal design principles
- we now illustrate this using some of our past work

# Design Example

# Policy composition or orchestration

- XACML inspired by work on policy algebra
- policy composition example (left: abstract, right: concrete):



# An XACML type core language

- expressive enough for any XACML-style composition
- contains a grammar clause as plug for base policies
- base policies contain also application specifics

| | |
|---|---|
| `pol ::=` | (Policy) |
| `allow | deny | na | conflict` | Constant Policy |
| % | Dead Code |
| `base` | Base Policy |
| `switch {pol: pol;pol;pol;pol}` | Policy Switch |

# Constant policies

`allow` | `deny` | `na` | `conflict`     Constant Policy

- represent possible access-control "decisions"
- allows for propagation of inconsistencies (`conflict`) and of gaps (`na` for "not applicable")
- has elegant information-theoretic representation in Belnap's bilattice

# Dead code

% Dead Code

- static safety check: portion of policy code unreachable
- useful for testing and for verification
- for example:

$$\texttt{switch}\ \{\texttt{P} : \%; \texttt{Q}; \texttt{Q}; \%\}$$

is safe for policy P that is free of gaps and conflicts

# Base policies

`base` Base Policy

- atomic policies that, say, are conflict free
- atomic policies as stubs for richer DSL
- e.g. `base` may contain a generic constraint language
- and `base` may have plugs for application specifics

# Policy Switch

$\texttt{switch}\ \{\texttt{pol}\colon \texttt{pol}; \texttt{pol}; \texttt{pol}; \texttt{pol}\}$ Policy Switch

- sole composition operator
- intuitive meaning of $\texttt{switch}\ \{p\colon q_\perp, q_0, q_1, q_\top\}$
  - identify $\perp$, 0, 1, and $\top$ with `na`, `deny`, `allow` and `conflict` (respectively)
  - evaluate switch policy $p$, renders value $v$
  - continue with evaluation of $q_v$

# Example switch: deny overrides operator `dov`

```
switch { P : na; deny; allow; deny}
```

- operator `dov` defined via above policy switch
- `dov` converts `conflict` into `deny`
- and won't change other decisions `na`, `deny`, and `allow`
- want to be able to name policy switches for reuse
- want to know input and output types
- extend core language to achieve both

# Typed, modular policies

- extend core language with types and methods
- types for input and output are
  - `pol` for any of the four decisions
  - `base` for any decision but `conflict`
  - `prd` for any decision `allow` or `deny`

e.g. can implement operator `dov` as

```
base deny-overrides(P:pol) {
      switch { P : na; deny; allow; deny }
}
```

# Two basic constraints for analysis

- Typed, modular policies enable rich set of analyses
- Analysis can abstract base policies as atomic expressions
- Many analyses can be build from two basic ones:
  - $p \Uparrow 1$ condition for when policy $p$ makes decision `allow` or `conflict`
  - $p \Uparrow 0$ condition for when policy $p$ makes decision `deny` or `conflict`

# Example: $\texttt{switch}\{p\colon q_\bot; q_0; q_1; q_\top\} \Uparrow 1$

- this constraint is a "DNF" of similar constraints for sub-policies:

$$
\begin{array}{ll}
(\neg(p \Uparrow 0) \wedge \neg(p \Uparrow 1) \wedge (q_\bot \Uparrow 1)) & \vee \\
((p \Uparrow 0) \wedge \neg(p \Uparrow 1) \wedge (q_0 \Uparrow 1)) & \vee \\
(\neg(p \Uparrow 0) \wedge (p \Uparrow 1) \wedge (q_1 \Uparrow 1)) & \vee \\
((p \Uparrow 0) \wedge (p \Uparrow 1) \wedge (q_\top \Uparrow 1)) &
\end{array}
$$

- constraint $\texttt{switch}\{p\colon q_\bot; q_0; q_1; q_\top\} \Uparrow 0$ similar "DNF"

Imperial College London

# Semantic types

- some types cannot be inferred statically, e.g.

```
prd foo(P:pol) { switch {P : deny; P; P; deny }
```

- type inference only concludes `pol` as return type for `foo`
- combine inference with policy analysis for more precision
- here: compute `foo` ⇑ 1 as `<BodyOf_foo>` ⇑ 1 using semantic copy rule
- show `foo` ⇑ 1 is logical dual of `foo` ⇑ 0
- therefore, `foo` only can output `deny` or `allow`
- so return type `prd` is safe

# Example language for base policies

```
base ::=                          (Base Policy)
   allow when constraint            Permit Rule
   deny when constraint             Deny Rule
```

- agnostic to application specifics at this level
- promotes constraints to base policies
- constraints may also demote policies, as we see next

# Example for constraint language

| | |
|---|---|
| `con ::=` | (Constraint) |
| `atomic` | Atomic Constraint |
| `!con` | Negated Constraint |
| `con && con` | Conjoined Constraint |
| `pol@deny` | Deny or Conflict $(p \Uparrow 0)$ |
| `pol@allow` | Allow or Conflict $(p \Uparrow 1)$ |

- pushes application specifics into grammar for `atomic`
- for `pol` of type `prd`, its implementation is `pol@allow`
- e.g. `pol@allow` represented as binary decision diagram

# Outlook

# What DSLs can bring to UMA

- generic policy patterns combined with application specifics
- formal meanings and analysis
- implementation and optimization principles
- interactive user experience, e.g. “For my resources, give me the access view of my Uncle Bob.”
- ability to
  - transform policies between languages
  - develop interfaces and adaptors between users, hosts, etc.

# Conclusions

We studied policy-based user managed access control (UMA):

- we identified its opportunities and challenges
  e.g. support of user management through policy analysis
- we discussed its language design and analysis issues
  e.g. to provide users with views that encapsulate DSLs
- we hinted at some solution approaches to these challenges
  e.g. the use of semantic types and modular programming

# References

Crampton J., and Huth M.
A Framework for the Modular Specification and Orchestration of Authorization Policies
Proc. 15th Nordic Conf. on Secure IT Systems

Bruns G., and Huth M.
Access Control via Belnap Logic: Intuitive, Expressive, and Analyzable Policy Composition
ACM Transactions on Information and System Security
accepted for publication in August 2010, ACM Press

Crampton J., and Huth M.
An Authorization Framework Resilient to Policy Evaluation Failure
Proc. 15th ESORICS, 2010

Imperial College
London

## Thank You for Your Kind Attention

# Questions?

Imperial College
London

# Q & A

# Q & A

# Q & A

Imperial College London

# Q & A

Imperial College
London

# Q & A

# Q & A

# Q & A

# Q & A

# Q & A

# Q & A