Policy Aware Systems
Some open research issues

Piero A. Bonatti

Università di Napoli Federico II and REWERSE

Pittsburgh, Feb 27, 2006
Outline I

1. Introduction
2. Formulating credential requests
3. Negotiations
4. A first set of open issues
5. Explanations
Credentials for Open Systems

Digital credentials constitute the main approach to access control for open systems

- **Reliable**
  - Unforgeable (cryptographic techniques)
  - Ownership can be checked (with challenges)
  - ...

- **Scalable**
  - There can be many domain-specific certification authorities...

- **Privacy-oriented**
  - Can represent properties of individuals
  - Without necessarily disclosing their identity

Widely adopted in basic tools such as SSL. Researchers are more ambitious
Scenario

Heterogeneous Agent Systems
(Hardcover)

by V. S. Subrahmanian, Pierre Bonatti, Jürgen Dix, Thomas Eiter, Sarit Kraus, Fatma Ozcan, Robert Ross

Browse: Front Cover | Table of Contents | Excerpt | Index | Back Cover

List Price: $68.78 & this item ships for FREE with Super Saver Shipping. See details

You Save: $11.22 (14%)

Availability: Usually ships within 24 hours. Ships from and sold by Amazon.com.

Only 1 left in stock—order soon (more on the way).

18 used & new available from $26.98

More Buying Choices

Quantity: 1

To turn on 1-Click ordering, sign in or create an account.

A9.com users save 1.57% on Amazon. Learn how.

Add to Wish List

Add to Wedding Registry

P.A. Bonatti (Univ. di Napoli & REWERSE)
Introduction

Scenario
Introduction

Scenario

Ordering from Amazon.com is quick and easy

Enter your e-mail address: [blank]

- [ ] I am a new customer.
  (You'll create a password later)

- [ ] I am a returning customer,
  and my password is:

  [blank]

  Sign in using our secure server

Forgot your password? Click here

Has your e-mail address changed since your last order?

The secure server will encrypt your information. If you received an error message when you tried to use our secure server, sign in using our standard server.

You are buying this item from Amazon.com, Inc.

The only way to place an order at Amazon.com is via our Web site. (Sorry—no phone orders. However, if you prefer, you may phone in your credit card number, after filling out the order form online.)

Redeeming a gift certificate? We’ll ask for your claim code when it’s time to pay.

Having difficulties? Please visit our Help pages to learn more about placing an order.

Page loaded.
Introduction

Scenario: Scalability and usability issues

Similar considerations hold for systems based on

- MyProxy, Kerberos, CAS
- oriented to “localized” navigation

In the absence of more flexible identification methods:

- Web services have to keep accounts for all customers
- Users have to create accounts all the time
- Articulated business policies are discouraged
Introduction

Scenario: Scalability and usability issues

You can get this book:
1. by logging in
2. by supplying an ID and a credit card
3. by providing an Amazon card

Please choose a number or click on a link for more information.
Scenario: Scalability and usability issues

What one would really want:

- Suppose the Amazon card gives you free access to some products
- If you have it, you want to use it automatically
  - click on the purchase button, and that's it
- If you don’t you may want to see something like the next figure
Introduction

Scenario: Scalability and usability issues

WARNING
You are about to pay $10 for paper0123.pdf using your VISA card
Similar desiderata:

- Travellers connect to airport lounge services
  - such as network, printers, content services, ...
  
  using

  - frequent flier cards
  - pre-payed cards
  - credit cards
  - employee credentials (government, airlines, ...)
  - ...

- In a transparent way (well, as much as possible)
How to ask for credentials

One by one (e.g. PeerTrust)
- slow (more messages)
- unnecessary disclosures
  - after sending off your credit card you realize that you should also send an id credential that you don’t have
- unnecessary messages (even slower)
How to ask for credentials

One by one (e.g. PeerTrust)

- slow (more messages)
- unnecessary disclosures
  - after sending off your credit card you realize that you should also send an id credential that you don’t have
- unnecessary messages (even slower)

All the alternatives at once

- less messages, less unnecessary disclosures
How to ask for credentials

One by one (e.g. PeerTrust)

- slow (more messages)
- unnecessary disclosures
  - after sending off your credit card you realize that you should also send an id credential that you don’t have
- unnecessary messages (even slower)

All the alternatives at once

- less messages, less unnecessary disclosures
- combinatorial explosion: an id and a credit card becomes
  - passport and VISA
  - passport and Mastercard
  - ...
  - student-card and VISA
  - ...
Send the policy

As a compact but exhaustive request formulation (e.g. Protune)

**Informal policy**

1. allow purchase if the customer sends an *id* and a *valid credit card* or...
2. an *id* can be a passport, a student-card, ... issued by a *recognized CA*
3. a *valid credit card* is issued by VISA or ... and it is not *expired*
4. ...

The client then searches its portfolio for credentials that - together with the (formal) policy - entail *allow purchase* (an *abduction problem*)

Proposed for the first time in [CCS 2000]
Formulating credential requests

Formal policy in Protune

Something similar to:

\[
\text{allow}(\text{purchase}, \text{Item}) \leftarrow \\
\quad \text{id}(\text{ID}), \\
\quad \text{credit\_card}(\text{CC}), \\
\quad \text{ID}\text{.name} = \text{CC}\text{.holder}.
\]

\[
\ldots \\
\text{credit\_card}(X) \leftarrow \\
\quad \text{credential}(X), \\
\quad \text{accepted\_cc}(X\text{.issuer}).
\]

\[
\text{accepted\_cc}(\text{'VISA'}). \\
\text{accepted\_cc}(\text{'Mastercard'}). \\
\ldots
\]
Formulating credential requests

Formal policy in Protune

Something similar to:

```prolog
allow(purchase, Item) ← (decision predicate)
    id(ID),
    credit_card(CC),
    ID.name = CC.holder.
...
credit_card(X) ←
    credential(X),
    accepted_cc(X.issuer).

accepted_cc('VISA').
accepted_cc('Mastercard').
...
```
Formal policy in Protune

Something similar to:

allow(purchase,Item) ←
id(ID),
credit_card(CC),
ID.name = CC.holder.
...
credit_card(X) ←
    credential(X),
    accepted_cc(X.issuer).

accepted_cc('VISA').
accepted_cc('Mastercard').
...
Formulating credential requests

Formal policy in Protune

Something similar to:

```
allow(purchase, Item) ←
id(ID),
credit_card(CC),
ID.name = CC.holder.
...
credit_card(X) ←
  credential(X),
  accepted_cc(X.issuer).

accepted_cc('VISA').
accepted_cc('Mastercard').
...
```

Flora-like O.O. syntax
Formulating credential requests

Relationships with Semantic Web

Informal policy

1. allow purchase if the customer sends an id and a valid credit card or...
2. an id can be a passport, a student-card, ... issued by a recognized CA
3. a valid credit card is issued by VISA or ... and it is not expired
4. ...

The definitions of id, valid credit card, recognized CA etc. constitute a simple ontology

The server shares its ontology with the client
  - basic shared knowledge: rule semantics and X.509
  - underlying logic: function-free Horn clauses
  - complex shared domain ontologies are not a prerequisite
  - feasible today
Privacy policies

Credentials may contain sensitive information

- users should not explicitly authorize each disclosure
- release policies are needed
- that can be treated like access control policies [CCS 2000]

Informal privacy policy

1. allow credit card disclosure if the server joins the Better Business Bureau program
2. allow student-id disclosure (always)
3. ...

In response to a credential request the client may issue a counter-request

⇒ Trust Negotiation
A negotiation scenario

**Step 1:** Alice requests a service from Bob

**Step 2:** Bob discloses his policy for the service

**Step 3:** Alice discloses her policy for VISA

**Step 4:** Bob discloses his BBB credential

**Step 5:** Alice discloses her VISA card credential

**Step 6:** Bob grants access to the service
Multi-party negotiations

Third parties may be needed to:

- check credit card validity
- store credentials
- give special permissions
- ...

Protune metapolicies may be used to specify whom is responsible for what, e.g.

\[
\text{credential}(C).\text{actor:serverXY} \leftarrow \text{C.type:student\_id}
\]

means that serverXY is to provide student ids
Some technical issues

- Policy protection
- Negotiation length
- Negotiation success
- Minimizing information disclosure
- Provisional policies (actions)
A first set of open issues

Policy protection

The policy itself is confidential

- it may reveal agreements between companies
- it may reveal private information
  - only my best friend can see my pictures
  - my best friends are ...
- definition of correct user-password pairs...

Policies have to be protected

- by hiding some rules
- by sanitizing others

⇒ Policy Filtering (before each disclosure)
A first set of open issues

Policy protection in Protune

The sensitivity of policy rules and predicates is declared with suitable metapolicies:

- A rule with name \([r]\) can be protected by asserting \([r].\text{sensitivity:private}\)

- Sensitivity may depend on further conditions, as in \([r].\text{sensitivity:public} \leftarrow \text{authenticated}(\text{User})\)

In this way, more rules can be disclosed as the level of trust increases during negotiation.

- Predicates can be protected in a similar way, e.g. \(\text{passwd}(\text{User}, \text{Pwd}).\text{sensitivity:private}\)

Further features are described in REWERSE report I2-D2
Sanitizing credential requests

Private rules can be **applicable** or **non-applicable**
- applicable rules are evaluated
- only their results are sent off
- non-applicable rules are discarded
- rules with a private predicate in the head are private

Private *state* predicates are **blurred**
- private atoms are replaced with a fresh propositional symbol
Policy filtering

- Filter non-applicable and irrelevant rules
  - Partial evaluation of public rules
  - Execute immediate actions
  - Evaluate local provisional literals
- Compile applicable, non-public rules
- Blur deferred state conditions
- Filter irrelevant policies due to blurring
- Replace provisional state predicates with actions
- Anonymize abbreviation predicates
A first set of open issues

Negotiation length

In general, difficult to predict

- the server may issue a counter-counter-request, and so on
- protected policies are disclosed incrementally
  - as the other peer sends more credentials

? Techniques for estimating max length

? Useful bounded protocols

? Useful restricted *policies*
  - 2-step disclosure [CCS 2000]
  - unilateral policies (the server releases no credentials)
  - transparent (public) policies
  - too restrictive in many cases
  - REWERSE is working on more general cases
Negotiation success

Negotiations may fail because the peers hide part of their policies

- peers do not know how to fulfill the access control conditions

- any local conditions that guarantee success? (if the policies allow)
  - little hope of being able to check global conditions on the policies of the involved peers
  - current results: “if such & such disclosure sequence exists then…”
  - when does it exist?
  - REWERSE is working at improving these results
Minimizing information (sensitivity) disclosure

- some credentials are more sensitive than others
  - Safeway’s discount card ≤ student-id ≤ credit card ≤ SSN ...
- even if all the policies are published, finding an optimal choice is computationally hard
  - precise characterization in the next REWERSE deliverable
- in general, when policies are protected no strategies guarantee optimality
- design languages for expressing preferences
- study reasonable negotiation strategies
- identify useful restricted cases that admit optimal strategies
  - and efficient algorithms, possibly approximate algorithms
  - some preliminary results in a forthcoming REWERSE report
Provisional policies (actions)

Sometimes policies have to execute actions
- log a request for audit purposes
- activate a workflow (e.g. for manual registration)
- ...

Credential themselves involve an action
- they can be requested and released and verified

In Protune further actions include
- **declarations** (unsigned)
  - accept a copyright/license agreement
  - login and password
  - ...
- **application dependent action**
  - e.g. connect to a URL
Example of declaration

Traditional authentication:

\[
\text{allow(access\_site) } \leftarrow \quad \text{declaration(username = N, password = P),}\n\] 
\[
\text{has\_passwd(N,P).}\n\]

Declarations are treated like credentials during negotiation

- Declarations are *not* signed
- they are included in the current state without any cryptographic verification

Declarations can be supplied

- automatically, if the client’s policy allows
- by filling in a form on a pop-up window
Metapolicies for actions

Specifying who is in charge of an action

\[
\text{credential(ID).actor:cmu\_CA} \leftarrow \text{ID.type:student\_id}.
\]
\[
\text{log(Request).actor:self}
\]

Specifying application-specific actions

\[
\text{log(_).type:provisional}.
\]
\[
\text{log(M).action: 'echo' + M + ' > log\_file'}.
\]

Specifying when an action should be executed

\[
\text{log(_).evaluation: immediate}.
\]

other values: deferred, concurrent

Plus some extra features (see REWERSE report I2-D2)
A first set of open issues

Interplay with filtering

- Filter non-applicable and irrelevant rules
- Compile applicable, non-public rules
- Partial evaluation of public rules
- Execute immediate actions
- Evaluate local provisional literals
- Blur deferred state conditions
- Filter irrelevant policies due to blurring
- Replace provisional state predicates with actions
- Anonymize abbreviation predicates
A first set of open issues

Execution module

Client

- Access Control Queries
- Why Queries
- How-to Queries

Inference Engine

Negotiation Controller

Execution Handler

Action Policies

Query

Policy Set

Decision

Filtered Policies + Proof

Answer

Inference Engine

Metadata

Query

Annotated Policies

(filtered from last client's request)

Annotated Policies
Event-Condition-Action rules

Current action semantics is vaguely Prolog-like

- when a predicate with actor `self` and evaluation immediate is enclosed in the filtered policy, it is evaluated

  \[
  \text{allow}(\text{Usr}, \text{Op}, \text{Obj}) \leftarrow \text{log}(\text{Usr}+\text{Op}+\text{Obj}), \ldots
  \]

- a bit less procedural than Prolog (parallel action execution)
- it fits well the abductive nature of negotiation

However many actions would be more naturally specified as ECA rules

- “... And by the way, whenever you get a request, log it”
- *incremental* policy formulation style
- not clear how to harmonize abductive and ECA semantics
Explanations
Users and policies

- Common users have little awareness and understanding of security and privacy policies
  - applied by their own system and by remote services

- this is a major source of security problems
  - a typical PC with default security settings is violated in < 5min
  - with a careful setting the same machine resists for weeks

- there may be service usability issues
  - many first-time and occasional users in web and pervasive environments
Challenges

- a tradeoff is needed between protection and functionality
  - based on user’s needs
  - generic policies typically won’t work
    \[\Rightarrow\text{users should be able to personalize their policies}\]
- similar arguments apply to privacy policies and credential release policies
  - risks are to be balanced with functionality and value
  \[\Rightarrow\text{help users get better understanding of and control on policies}\]
Strategies

- Education and dissemination through mass media
- Let users formulate their policies
  - user-friendly languages
  - based on simple concepts (no cookies)
- Explain policies and decisions
  - never say (only) no
  - negative answers should come with explanations and suggestions
Formulating policies

Graphical languages
- so far, not expressive enough [Winslett et al.]
- still interesting for *part* of the specifications e.g. user and object hierarchies

Controlled natural language

*A user can browse directory “internal docs” if he provides a REWERSE credential*

- to be translated into Protune rules
- REWERSE is extending the Attempto system [Fuchs et al.]
Automated Explanations - Goals

Rich query set

- *how-to, why/why-not, what-if*

Quality comparable to 2nd generation explanation facilities

- remove irrelevant details
- high-level object identification
- ...

With improved failure explanations (*why not* queries)

- handling infinite failures

And low framework instantiation cost

- for every new application domain
Protune’s explanations in a nutshell

- a hypertext
  - nodes corresponds to the entries of *tabled* LP engines (subgoal calls)
    ⇒ can explain *infinite failure*

- **local and global** proof info to improve navigation ease
  - rules applicable to the current goal
  - answer substitutions for each of them

- **intra- and inter-proof** info
  - users can match anticipated proof outcomes with their own expectations and expand only the interesting parts of the proof

- explanations are focussed on what the user *can do/should do/should have done*

- irrelevant details are omitted using *generic heuristics*

- objects are denoted by means of their attributes (*clusters*)
to make sure that
it is allowed to download Resource
nothing needs to be done if Resource is public
alternatively

please make sure that for some User
User is authenticated
where for some Subscription
User subscribed Subscription
and
Resource is available for Subscription

alternatively ...
Example: Why-not query

I can't prove that it is allowed to download paper012.pdf because:

allow(download(Resource)) ← public(Resource).

allow(download(Resource)) ← authenticated(User), has_subscr(User,Subscription), available(Resource,Subscription).

allow(download(Resource)) ← authenticated(User), paid(User,Resource).
Example: Why-not query

I can't prove that it is allowed to download paper012.pdf because:

Rule [2] is not applicable:

allow(download(Resource)) ←
public(Resource).

allow(download(Resource)) ←
authenticated(User),
has_subscr(User,Subscription),
available(Resource,Subscription).

allow(download(Resource)) ←
authenticated(User),
paid(User,Resource).

- Rule [1] removed by filtering
Example: Why-not query

I can’t prove that it is allowed to download paper012.pdf because:

Rule [2] is not applicable: there is no User such that User is authenticated

Furthermore, there is no User such that User paid for paper012.pdf

allow(download(Resource)) ← public(Resource).

allow(download(Resource)) ← authenticated(User), has_subscr(User,Subscription), available(Resource,Subscription).

allow(download(Resource)) ← authenticated(User), paid(User,Resource).

- Rule [1] removed by filtering
- Rule [2] partially omitted
Example: Why-not query

I can't prove that it is allowed to download paper012.pdf because:

Rule [2] is not applicable: there is no User such that User is authenticated and

Rule [3] is not applicable: there is no User such that User is authenticated moreover there is no User such that User paid for paper012.pdf

allow(download(Resource)) ←

public(Resource).

allow(download(Resource)) ←

authenticated(User), has_subscr(User,Subscription), available(Resource,Subscription).

allow(download(Resource)) ←

authenticated(User), paid(User,Resource).

- Rule [1] removed by filtering
- Rule [2] partially omitted
- Rule [3] involves 2 user-dependent conditions
Predicate authenticated/1 depends on valid_id/1 ...

I can’t find any Cred such that Cred is a valid id because:

Rule [6] is not applicable: c321 is a credential with type student-id and issuer Open University, student-id is an id but it is not the case that Open University is trusted for id

valid_id(Cred) ← credential(Cred), Cred.type : T, Cred.issuer : CA, isa(T,id), trusted_for(CA,id).
Example: Why-not query

Predicate authenticated/1 depends on valid_id/1 ...

I can’t find any Cred such that Cred is a valid id because:

Rule [6] is not applicable:
c321 is a credential with type student-id and issuer Open University, student-id is an id but it is not the case that Open University is trusted for id

Here you see an example of a cluster
Explanations need not be built on the server:

- the “server” sends its filtered policy together with predicate verbalization rules (and possibly the outcome of local predicates)

  authenticated(X).explanation : [X,is,authenticated]
  not authenticated(X).explanation : [X,is,not,authenti...  

- the “client” constructs the tabled explanation structure and verbalizes the explanations

  ⇒ the computational cost of explanations can be moved to the clients
Final observations

- Explanations with a reasonable quality can be built with little instantiation effort
- and without overloading the server
- we are planning to assist the creation of literal verbalization by means of the natural language front-end for policy formulation
- some experimentation is needed to evaluate and refine the current heuristics
- there is space for improvements...
Final observations

- Explanations with a reasonable quality can be built with little instantiation effort
- and without overloading the server
- we are planning to assist the creation of literal verbalization by means of the natural language front-end for policy formulation
- some experimentation is needed to evaluate and refine the current heuristics
- there is space for improvements...

NB: there are several other interesting TM issues that could not be discussed in this talk...

Questions?