Cassandra: Distributed Access Control Policies with Tunable Expressiveness

Moritz Y. Becker and Peter Sewell

Computer Laboratory, University of Cambridge, U.K.

Cassandra: Yet Another PSL?

Cassandra

- distributed Trust Management
- rule-based policy specification language (PSL)
- role-based: activation, deactivation, actions
- distributed: credential management

Cassandra: Yet Another PSL?

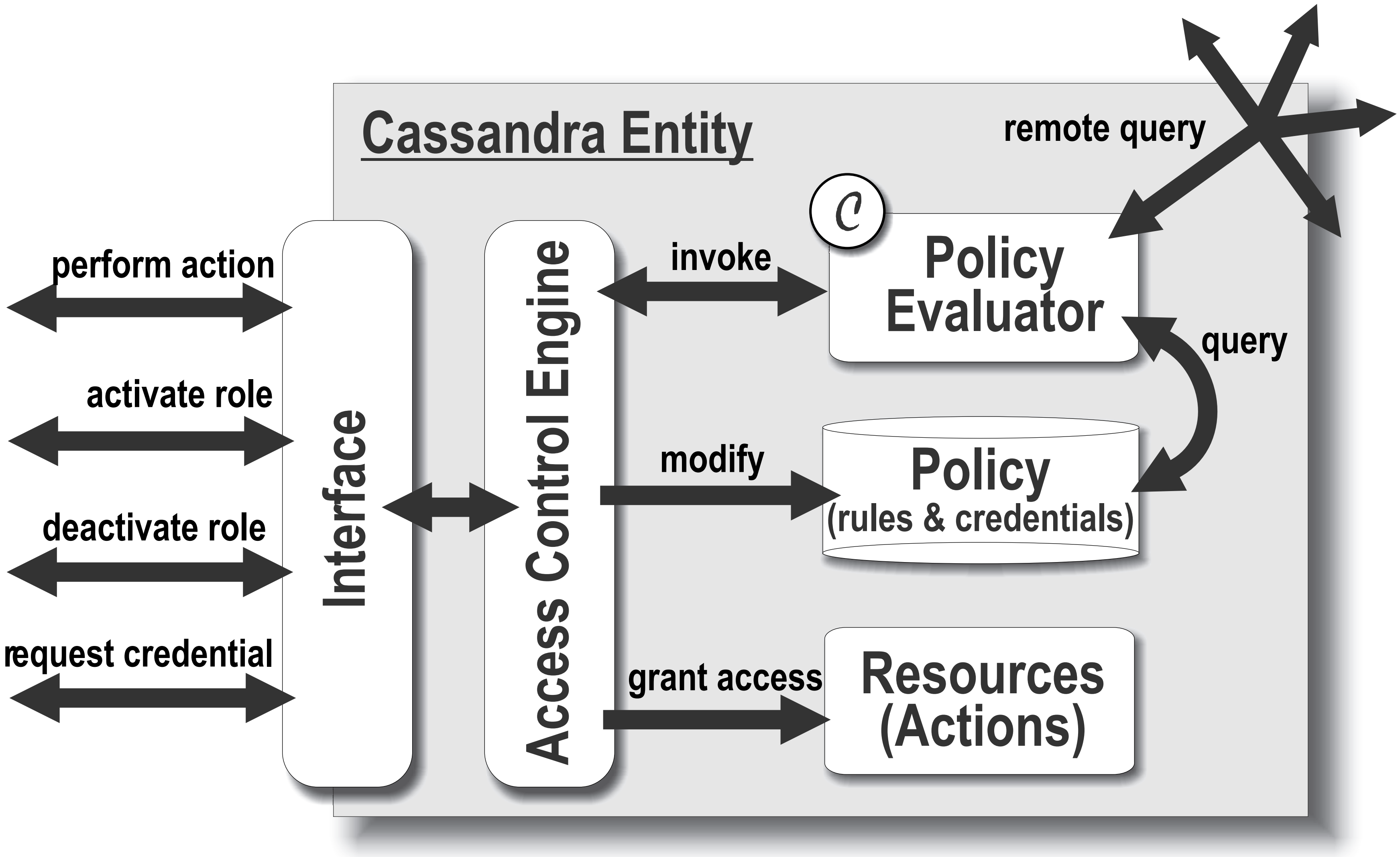
Cassandra

- distributed Trust Management
- rule-based policy specification language (PSL)
- role-based: activation, deactivation, actions
- distributed: credential management

Why YAPSL?

- wide range of applications $\longrightarrow$ need tunable expressiveness
- formal semantics: language **and** dynamics
- distributed query evaluation with guaranteed termination
- practical foundation: real-life case study

Cassandra Overview



Access Control Semantics (1/2)

- What: specifies dynamic meaning of 4 requests
- Why: makes subtle design decisions explicit

Access Control Semantics (1/2)

- What: specifies dynamic meaning of 4 requests
- Why: makes subtle design decisions explicit

- can P **perform** action A on S's service?
  - deduce $\mathsf{permits}(P, A)$

Access Control Semantics (1/2)

- What: specifies dynamic meaning of 4 requests
- Why: makes subtle design decisions explicit

- can P **perform** action A on S's service?
  - deduce $\mathsf{permits}(P, A)$

- can P **activate** role R on S's service?
  - deduce $\mathsf{canActivate}(P, R)$
  - add $\mathsf{hasActivated}(P, R)$ to S's policy

Access Control Semantics (2/2)

- can P **deactivate** V's role R on S's service?
  - deduce canDeactivate(P, V, R)
  - under the assumption isDeactivated(V, R), deduce all isDeactivated$(?, ?)$ on S
  - remove all corresponding hasActivated$(?, ?)$ from S's policy

Access Control Semantics (2/2)

- can P **deactivate** V's role R on S's service?
  - deduce canDeactivate(P, V, R)
  - under the assumption isDeactivated(V, R), deduce all isDeactivated$(?, ?)$ on S
  - remove all corresponding hasActivated$(?, ?)$ from S's policy
- can P **request credential** $E_{iss}.p(\vec{x}) \leftarrow c$ from S?
  - deduce canReqCred$(P, E_{iss}.p(\vec{x}) \leftarrow c)$ to get c'
  - deduce $E_{iss}.p(\vec{x}) \leftarrow c'$

Policy Specification

- entities control access to their resources with a **Cassandra** policy
- a *policy* is a set of rules based on Datalog$_{\mathcal{C}}$
- *rules* are of the form

$$p_0(\vec{e}_0) \leftarrow loc_1@iss_1.p_1(\vec{e}_1), .., loc_n@iss_n.p_n(\vec{e}_n),\ c.$$

(where loc_i, iss_i are entities and c is a constraint from the constraint domain)

Policy Specification

- entities control access to their resources with a **Cassandra** policy
- a *policy* is a set of rules based on Datalog$_\mathcal{C}$
- *rules* are of the form

$$p_0(\vec{e}_0) \leftarrow loc_1@iss_1.p_1(\vec{e}_1), .., loc_n@iss_n.p_n(\vec{e}_n), \; c.$$

  (where loc_i, iss_i are entities and c is a constraint from the constraint domain)

- predicates with special access control meaning:
  permits, hasActivated, canActivate, canDeactivate, isDeactivated, canReqCred

Policy Specification

- entities control access to their resources with a **Cassandra** policy
- a *policy* is a set of rules based on Datalog$_{\mathcal{C}}$
- *rules* are of the form

$$p_0(\vec{e}_0) \leftarrow loc_1@iss_1.p_1(\vec{e}_1), .., loc_n@iss_n.p_n(\vec{e}_n),\ c.$$

(where loc_i, iss_i are entities and c is a constraint from the constraint domain)

- predicates with special access control meaning:
  permits, hasActivated, canActivate, canDeactivate, isDeactivated, canReqCred

- Example: suppose a hospital's policy contains
  canActivate$(x, \texttt{Doctor}(spcty)) \leftarrow$
  $x@\texttt{NHS}$.canActivate$(x, \texttt{CertifiedDoctor}(spcty)),\ x \neq \texttt{Alice}$

Constraint Domains for Tuning Expressiveness

- $\mathcal{C}_{min}$, The simplest constraint domain:

$$
\begin{aligned}
e &::= x \mid E \quad \in \mathit{Entities} \\
c &::= \text{true} \mid \text{false} \mid (e_1 = e_2) \mid c_1 \wedge c_2 \mid c_1 \vee c_2
\end{aligned}
$$

Constraint Domains for Tuning Expressiveness

- $\mathcal{C}_{min}$, The simplest constraint domain:

  $$\begin{array}{lll} e & ::= & x \mid E \quad \in \mathit{Entities} \\ c & ::= & \text{true} \mid \text{false} \mid (e_1 = e_2) \mid c_1 \wedge c_2 \mid c_1 \vee c_2 \end{array}$$

- $\mathcal{C}_0$, a useful one for complex policies:

  $$\begin{array}{lll} e & ::= & ... \mid n \mid c \mid (e_1, .., e_n) \mid \pi_i(e) \mid f(e) \mid R(e_1, .., e_n) \mid \\ & & A(e_1, .., e_n) \mid \emptyset \mid \Omega \mid \{e_1, .., e_n\} \mid e_1 \cup e_2 \mid e_1 \cap e_2 \mid e_1 \setminus e_2 \\ c & ::= & ... \mid e_1 \neq e_2 \mid e_1 < e_2 \mid e_1 \subseteq e_2 \end{array}$$

Constraint Domains for Tuning Expressiveness

- $\mathcal{C}_{min}$, The simplest constraint domain:
  $e \quad ::= \quad x \mid E \qquad \in Entities$
  $c \quad ::= \text{true} \mid \text{false} \mid (e_1 = e_2) \mid c_1 \wedge c_2 \mid c_1 \vee c_2$

- $\mathcal{C}_0$, a useful one for complex policies:
  $e \quad ::= \; ... \mid n \mid c \mid (e_1, .., e_n) \mid \pi_i(e) \mid f(e) \mid R(e_1, .., e_n) \mid$
  $\qquad A(e_1, .., e_n) \mid \emptyset \mid \Omega \mid \{e_1, .., e_n\} \mid e_1 \cup e_2 \mid e_1 \cap e_2 \mid e_1 \setminus e_2$
  $c \quad ::= \; ... \mid e_1 \neq e_2 \mid e_1 < e_2 \mid e_1 \subseteq e_2$

- Constraint domains must support
  - satisfiability checking
  - projection
  - subsumption checking
- For guaranteed termination, constraint domains have to be *constraint compact*

Policy Idioms in Cassandra (1/2)

- **appointment**

  $\mathsf{canActivate}(mgr, \texttt{AppointEmployee}(emp)) \leftarrow$
  $\quad \mathsf{hasActivated}(mgr, \texttt{Manager}())$

  $\mathsf{canActivate}(emp, \texttt{Employee}(appointer)) \leftarrow$
  $\quad \mathsf{hasActivated}(appointer, \texttt{AppointEmployee}(emp))$

- **appointment revocation**

  $\mathsf{isDeactivated}(emp, \texttt{Employee}(appointer)) \leftarrow$
  $\quad \mathsf{isDeactivated}(appointer, \texttt{AppointEmployee}(emp))$

Policy Idioms in Cassandra (2/2)

- **grant-dependent** vs **grant-independent** appointment revocation

  canDeactivate$(x, appointer, \texttt{AppointEmployee}(emp)) \leftarrow$
  $x = appointer$

  canDeactivate$(x, appointer, \texttt{AppointEmployee}(emp)) \leftarrow$
  hasActivated$(x, \texttt{Manager}())$

- **cascading appointment revocation**

  isDeactivated$(mgr, \texttt{AppointEmployee}(emp)) \leftarrow$
  isDeactivated$(supermgr, \texttt{AppointManager}(mgr))$

- **others:** role hierarchy, role delegation, separation of duties, role validity dates, cardinality/manifold constraints, trust negotiation, ...

National EHR in UK

- NHS planning ICRS with online EHR for clinicians and patients
- Difficulties:
  - huge: 100m records, 400m episodes/yr, 1bn accesses/yr
  - changing requirements
  - distributed policies
  - patient confidentiality requirements
  - access control can be configured by patients/clinicians
- Our three layer approach:
  Master Patient Index (1), EHR servers (100s), health orgs (1000s)
- **Cassandra** policies for all layers: 310 rules, 58 roles, 10 actions
- patient consent, third-party consent, personal AC configuration, legal agents, staff appointment, clinician certification

An Example from the EHR Policy

Prerequisite for Treating-clinician

$$\text{canActivate}(cli, \texttt{Treating-clinician}(pat, org, spcty)) \leftarrow$$
$$org.\text{canActivate}(cli, \texttt{Group-treating-clinician}(pat, group, spcty)),$$
$$org@ra.\text{hasActivated}(x, \texttt{NHS-health-org-cred}(org, start, end)),$$
$$ra \in \texttt{NHS-registration-authorities}(),$$
$$\texttt{Current-time}() \in [start, end]$$

Conclusion

- **Cassandra**'s expressiveness is tunable; very expressive with $\mathcal{C}_0$
- high-level enough for concise and readable policies
- low-level enough to express wide range of policies
- formal foundation
- substantial case study
- prototype implementation