



Using Contextual Security Policies for Threat Response



Yohann THOMAS - Hervé DEBAR

France Télécom R&D - Caen, France

Frédéric CUPPENS - Nora CUPPENS-BOULAHIA

ENST Bretagne - Rennes, France

Information systems service goals



- Performance
 - Response time
 - Number of users served
- Convenience
 - Ease of use
 - Automation
- Security
 - Confidentiality
 - Integrity
 - Availability

- Security is one of **many** adjustment variables
- Compromises are generally **static** at design time

But ...



- Security is not static
 - New vulnerabilities
 - New users and usages
 - New attackers
- Nor are the other variables
 - Reflect the evolution of the IS (new hardware & software)
 - Maintain a better balance between the different requirements
- The compromise between these variables needs to change
 - Respond to threat
 - **Dynamic** security policies

Threat response system



→ Reactivity

- Automated response process
- On-time deployment of response according to threats
- On-time withdrawal of response

→ Reliability

- Consistency of the threat characterization system (reliable alerts)
- Relevance of selected countermeasures
- Application of countermeasures to multiple enforcement points

→ Ease of use

- Ease of deployment (avoid or limit additional systems)
- Ease of countermeasures definition

How to fulfill the requirements?



- Clear identification of the threat, source and victim
- Policy-oriented approach
 - Adapt security level to the threat level (**dynamic** policy)
 - Compromise between security, performance, convenience, etc.
 - Avoid the deployment of additional systems
- Organization-based approach
 - **Abstract** vs **concrete** level of rules
 - Provide **local** reactions but responding to **global** constraints
- Context-based approach
 - Trigger security rules thanks to active contexts
 - In particular, **threat contexts** to trigger countermeasures

Access Control Policy (1)



→ Discretionary Access Control (DAC)

- Manage *privileges* of *subjects* on *objects*
- Definition of an *access matrix* to describe authorizations

(Subject, Object, Privilege)

Ex: (host1, file1, rw),

Means that host1 has the privilege of read and write on file1.

→ Limitations

- Many subjects and objects to describe
- Scalability issues (definition and administration)
- Poor expressiveness (*static* policy)

(Unrestricted)

Access Control Policy (2)



→ Role-Based Access Control (RBAC)

- Abstract *subjects* into *roles*
- Manage *permissions* of *actions*

```
Permission(Role, Action, Object)
UA  $\subseteq$  UxR, user-to-role assignment
```

```
Ex: Permission(group1, read, file1), with host1  $\in$  group1,
Means that group1, thus host1, is permitted to read file1.
```

→ Limitations

- Only provides means to group subjects, but not actions and objects
- Only manages permissions (no explicit prohibition)
- Limited expressiveness of the security rules (*static* policy)

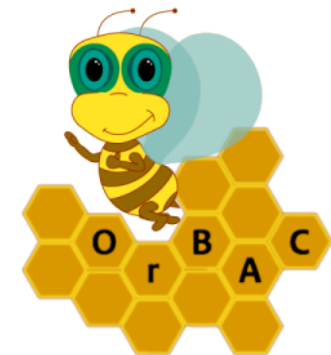
(Unrestricted)

Access Control Policy (3)



→ Organization-Based Access Control (Or-BAC)

- Manage entities through *organizations*
- Abstract *subjects* into *roles*
- Abstract *actions* into *activities*
- Abstract *objects* into *views*



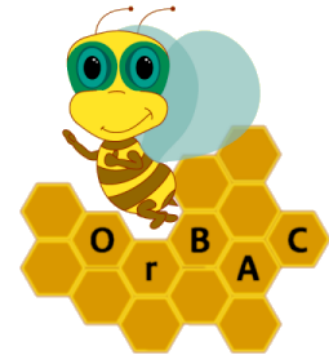
```
+ Empower(Organization, Subject, Role)
+ Consider(Organization, Action, Activity)
+ Use(Organization, Object, View)
```

Access Control Policy (3)



→ Organization-Based Access Control (Or-BAC)

- Manage entities through *organizations*
- Abstract *subjects* into *roles*
- Abstract *actions* into *activities*
- Abstract *objects* into *views*
- Provide not only *permissions*, but also *prohibitions/obligations*



```
Security_rule(Type, Organization, Role, Activity, View)  
+ Empower(Organization, Subject, Role)  
+ Consider(Organization, Action, Activity)  
+ Use(Organization, Object, View)
```

```
With Type={permission, prohibition, obligation}
```

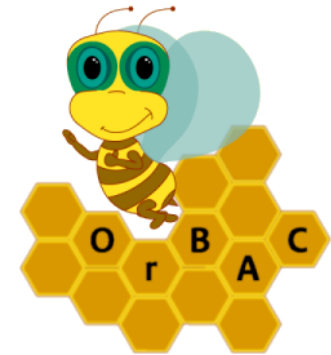
(Unrestricted)

Access Control Policy (3)



→ Organization-Based Access Control (Or-BAC)

- Manage entities through *organizations*
- Abstract *subjects* into *roles*
- Abstract *actions* into *activities*
- Abstract *objects* into *views*
- Provide not only *permissions*, but also *prohibitions/obligations*
- Trigger rules provided *contexts* (**dynamic** policy)

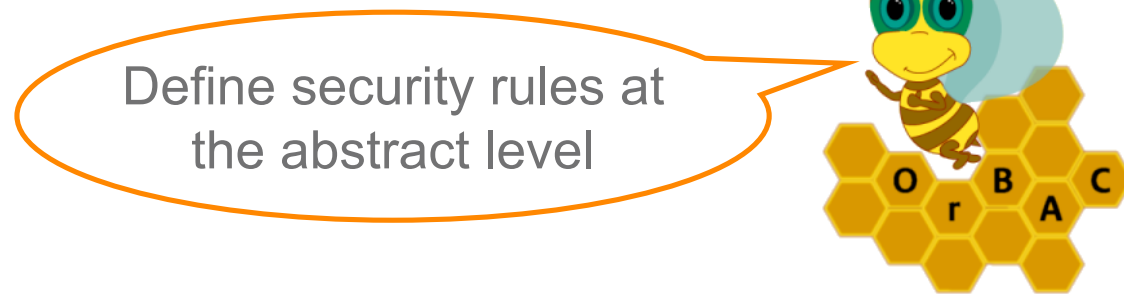


```
Security_rule(Type, Organization, Role, Activity, View, Context)
+ Empower(Organization, Subject, Role)
+ Consider(Organization, Action, Activity)
+ Use(Organization, Object, View)
+ Hold(Organization, Subject, Action, Object, Context)

With Type={permission, prohibition, obligation}
```

(Unrestricted)

Proposal (1): Use of Or-BAC



```
Security_rule(perm, corp, mail_user, read_mail, mailserver, normal)
```

- ➔ In the organization *corp*, the activity *read_mail* is permitted for the role *mail_user* on the view *mailserver* in a *normal* context.

Proposal (1): Use of Or-BAC



Contexts are activated at the concrete level



```
Security_rule(perm, corp, mail_user, read_mail, mailserver, normal)
```

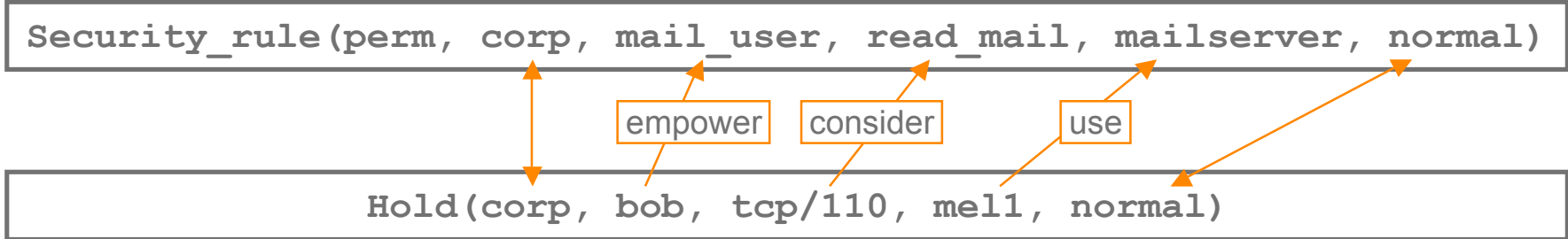
```
Hold(corp, bob, tcp/110, mel1, normal)
```

→ In the organization *corp*, the context *normal* is being held for user *bob* making action *tcp/110* on object *mel1*.

Proposal (1): Use of Or-BAC



Link hold facts with security rules

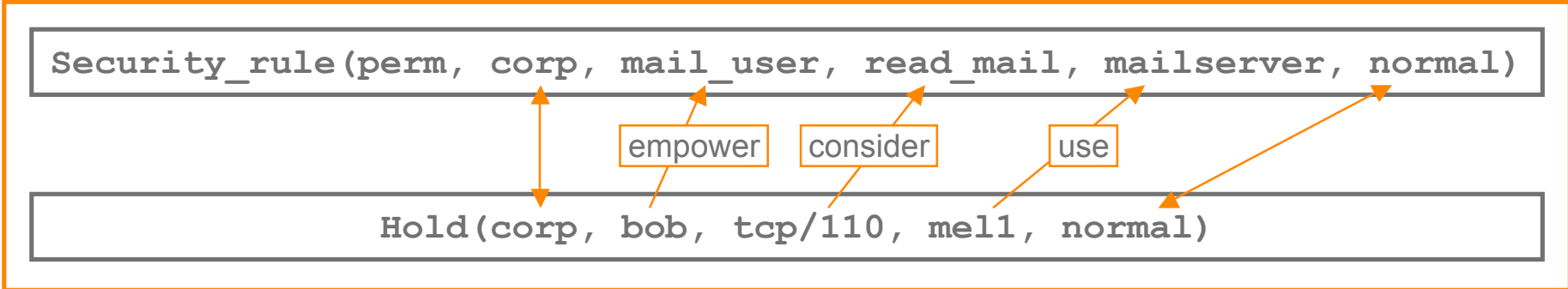
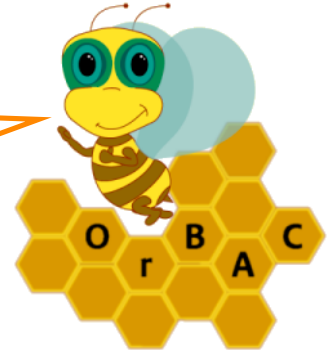


➔ In the organization *corp*, *bob* is a *mail_user* subject, *tcp/110* is a *read_mail* action and *mel1* is a *mailserver* object.

Proposal (1): Use of Or-BAC

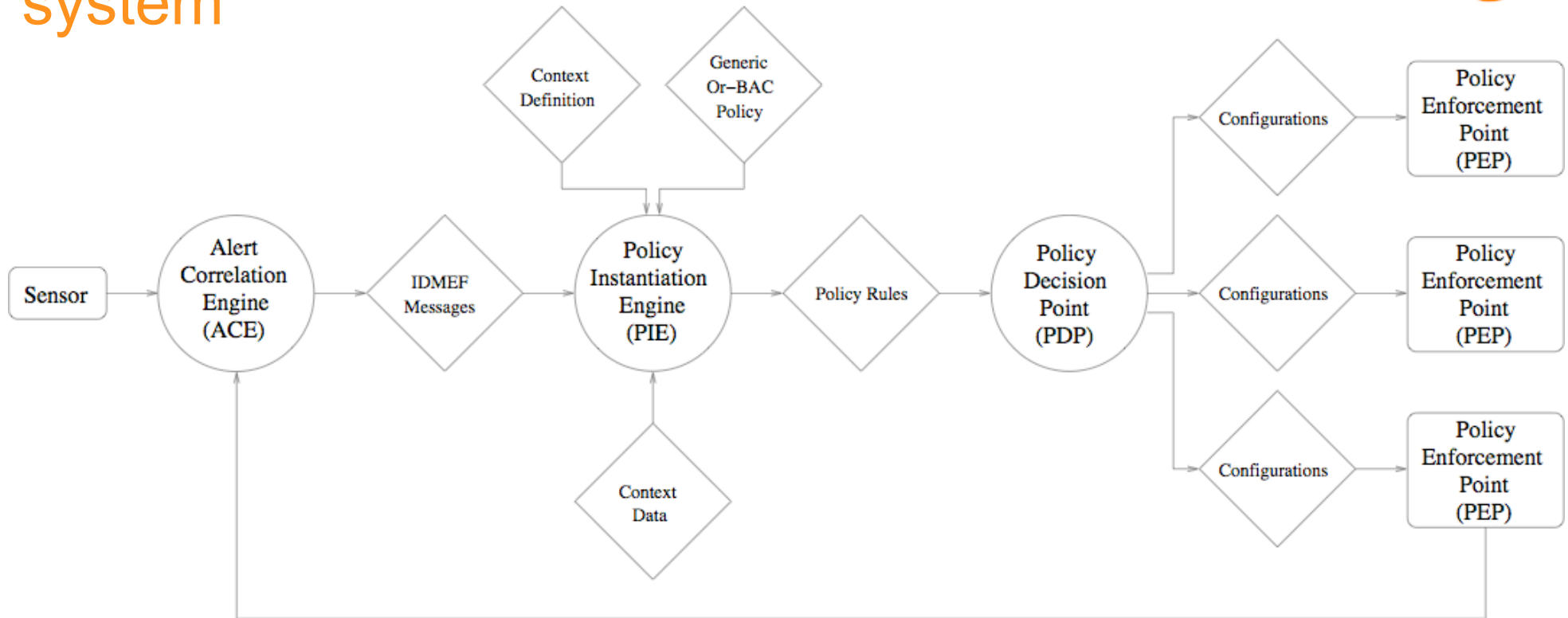


Derive concrete authorizations

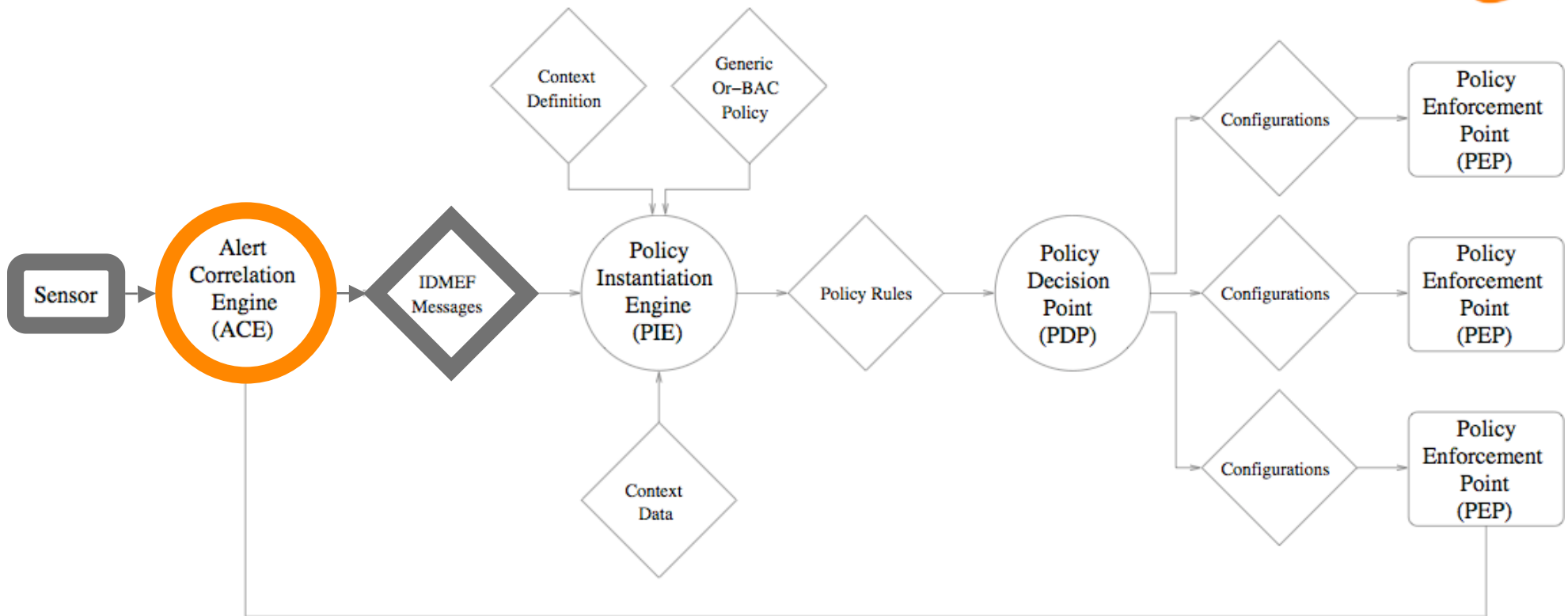


Bob is permitted to access tcp/110 port of mailserver mel1. Thus, he is allowed to read his mail in a normal context.

Proposal (2): Architecture for a threat response system



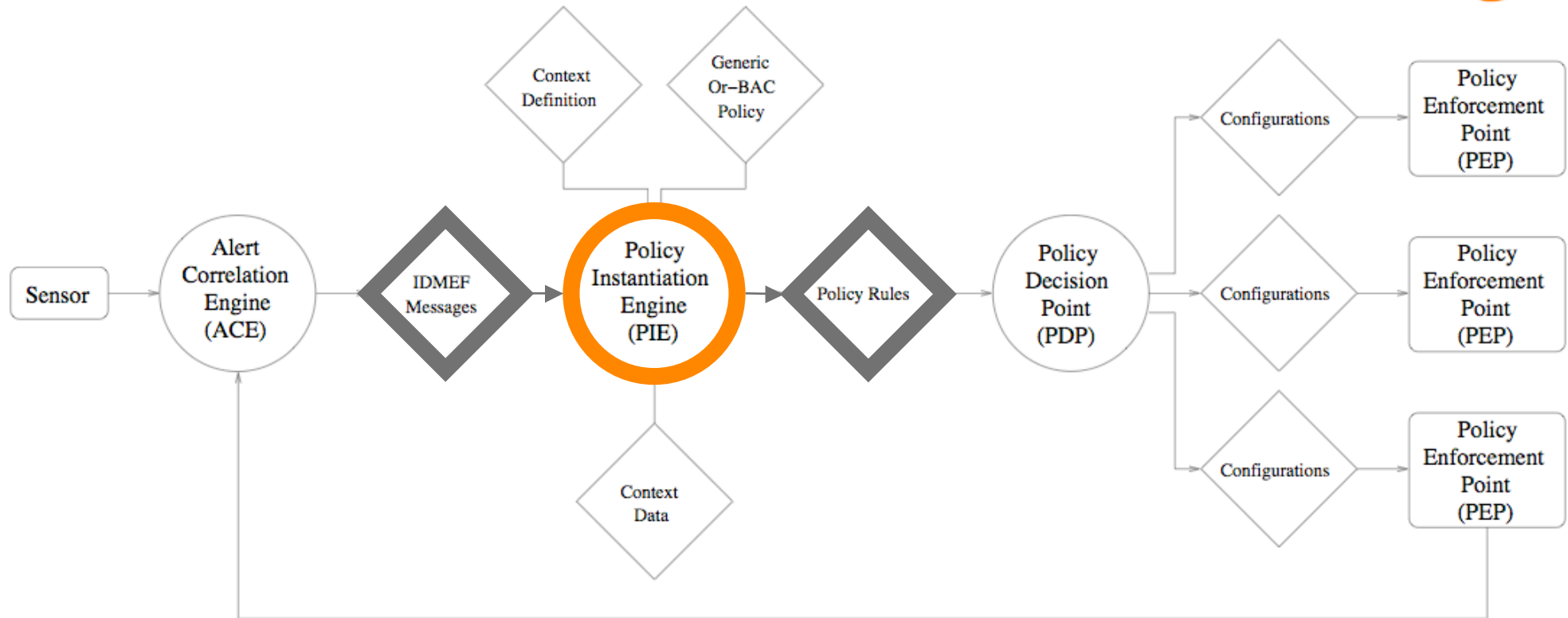
Proposal (2): Alert Correlation Engine (ACE)



- Input: Events/alerts from sensors (Snort, Prelude, firewall logs, etc.)
- Role: Provide reliable alerts reporting threats (existing tools are assumed reasonably accurate for the purpose of this work)
- Output: IDMEF messages (Intrusion Detection Message Exchange Format)

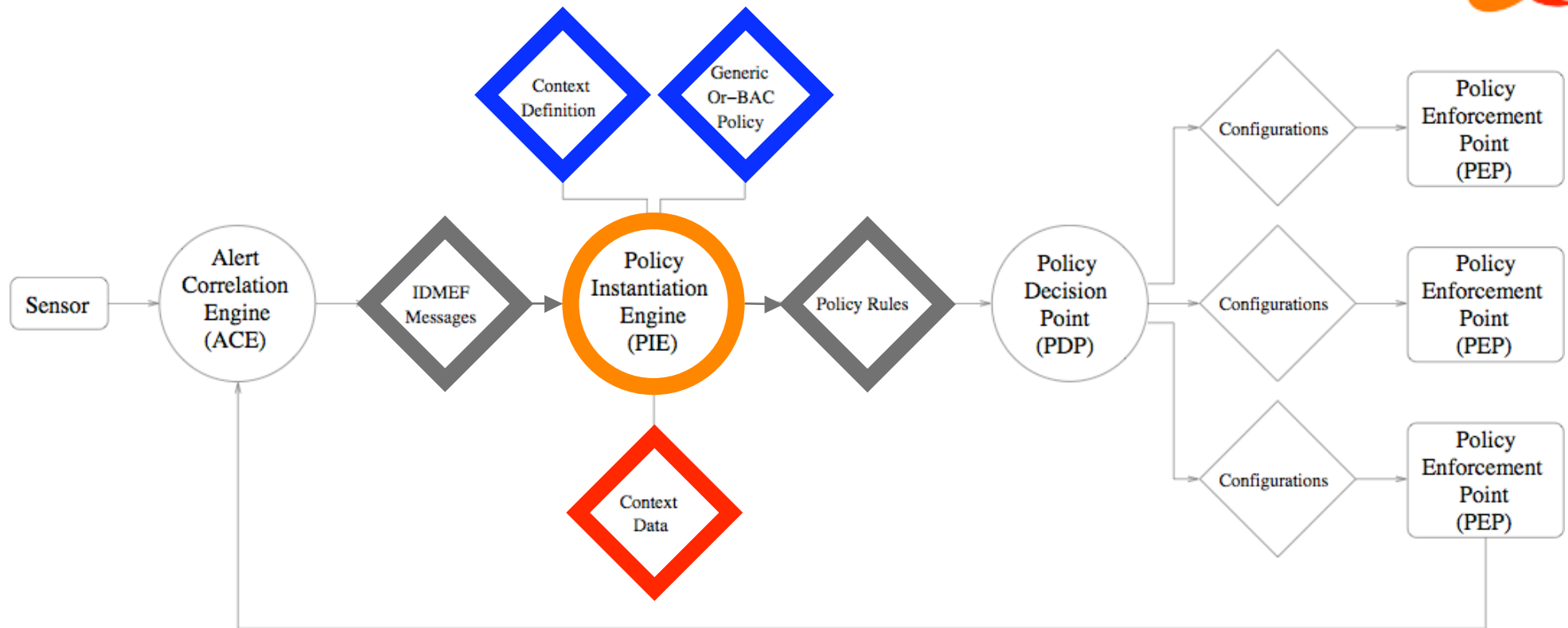
(Unrestricted)

Proposal (2): Policy Instantiation Engine (PIE)



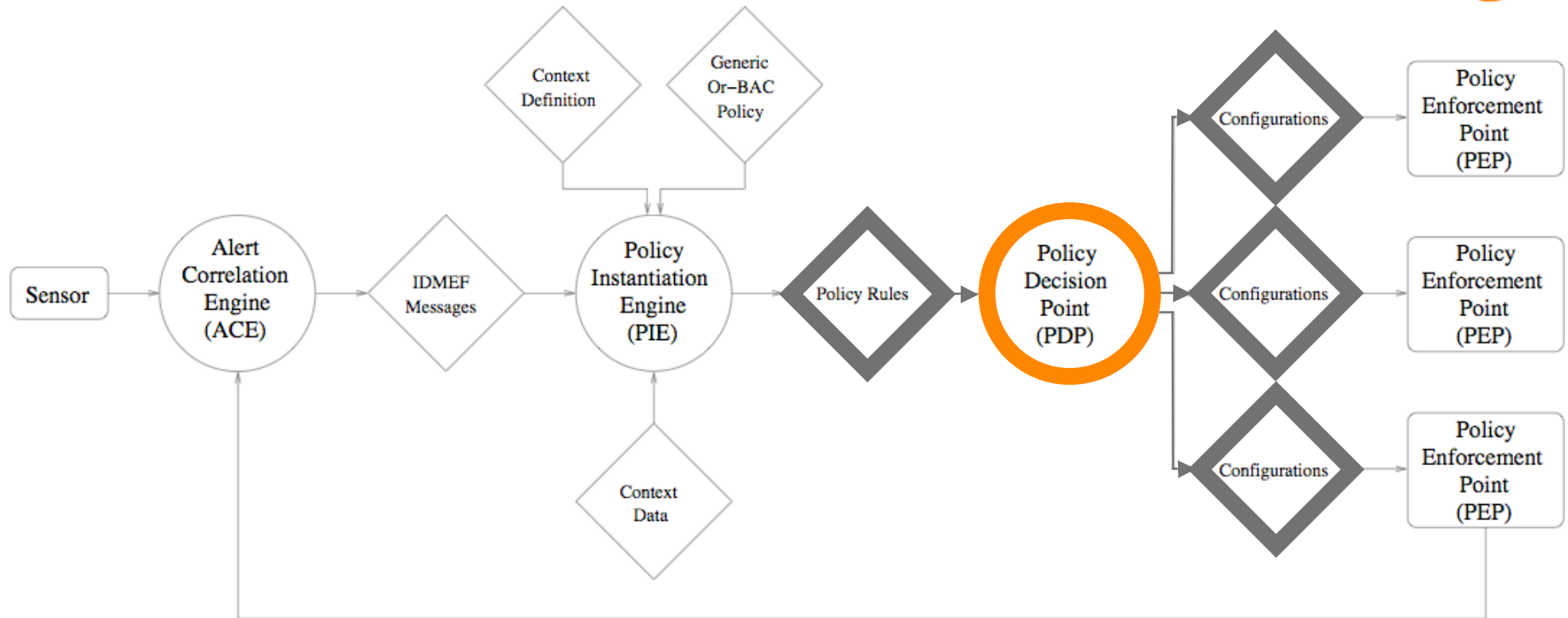
- Input: IDMEF messages (characterized threats)
- Role: Dynamically extract new policy rules considering threats
- Output: New policy rules (or instances)

Proposal (2): Policy Instantiation Engine (PIE)



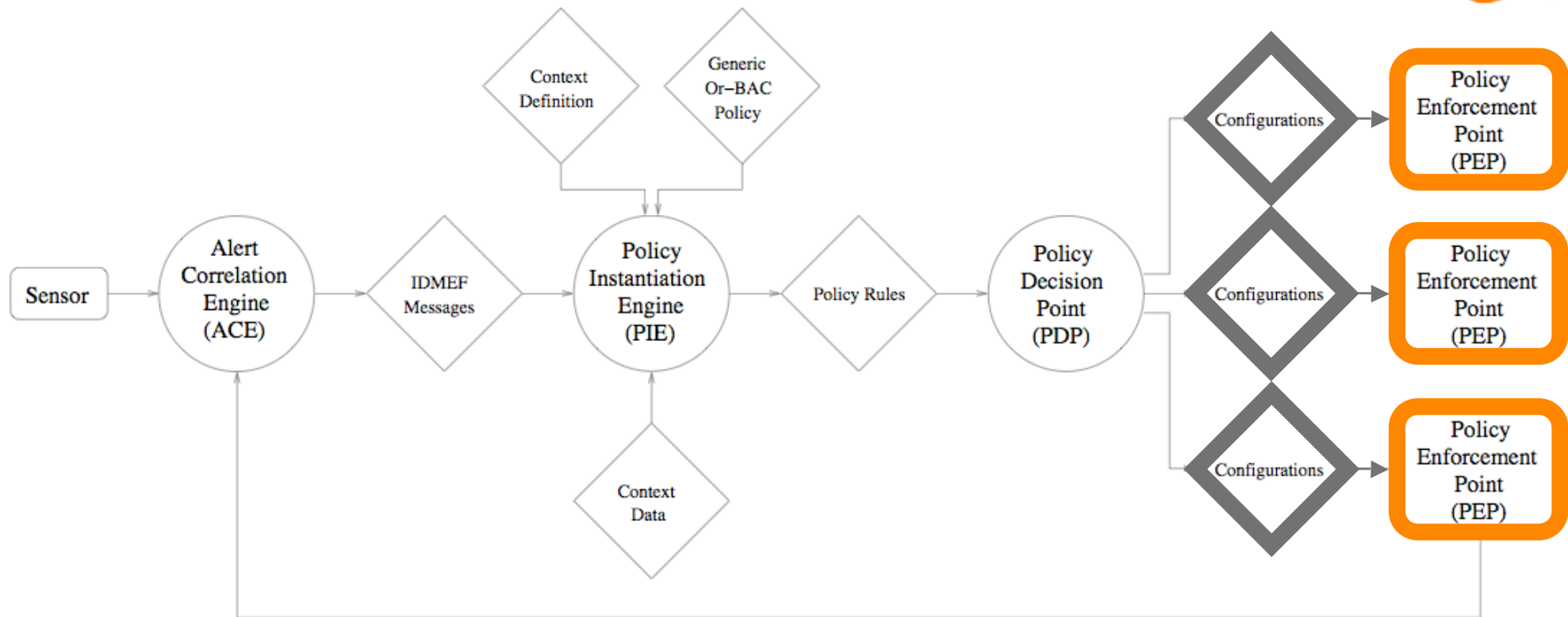
- Additional input: Policy definition
 - Generic Or-BAC policy (*security rules*, i.e. abstract policy)
 - Context definition (conditions to trigger contexts, i.e. *hold* predicates)
 - Context data (base of additional facts, apart from alerts, such as time, cartography, etc.)

Proposal (2): Policy Decision Point (PDP)



- Input: Policy rules
- Role: Prepare the policy for local enforcement
- Output: PEP configurations

Proposal (2): Policy Enforcement Point (PEP)



- Input: PEP configurations
- Role: Apply new configurations, i.e. enforce the policy
- Potential output: Events/alerts (PEPs acting as sensors)

From alerts to new policies (1)



- ➔ Alerts provide identification of source, victim and threat
 - IDMEF.Source: IP address, DNS name, network mask, etc.
 - IDMEF.Target: IP address, DNS name, network mask, etc.
 - IDMEF.Classification: Reference (ex: CVE-2005-1133)

➔ Mapping strategy

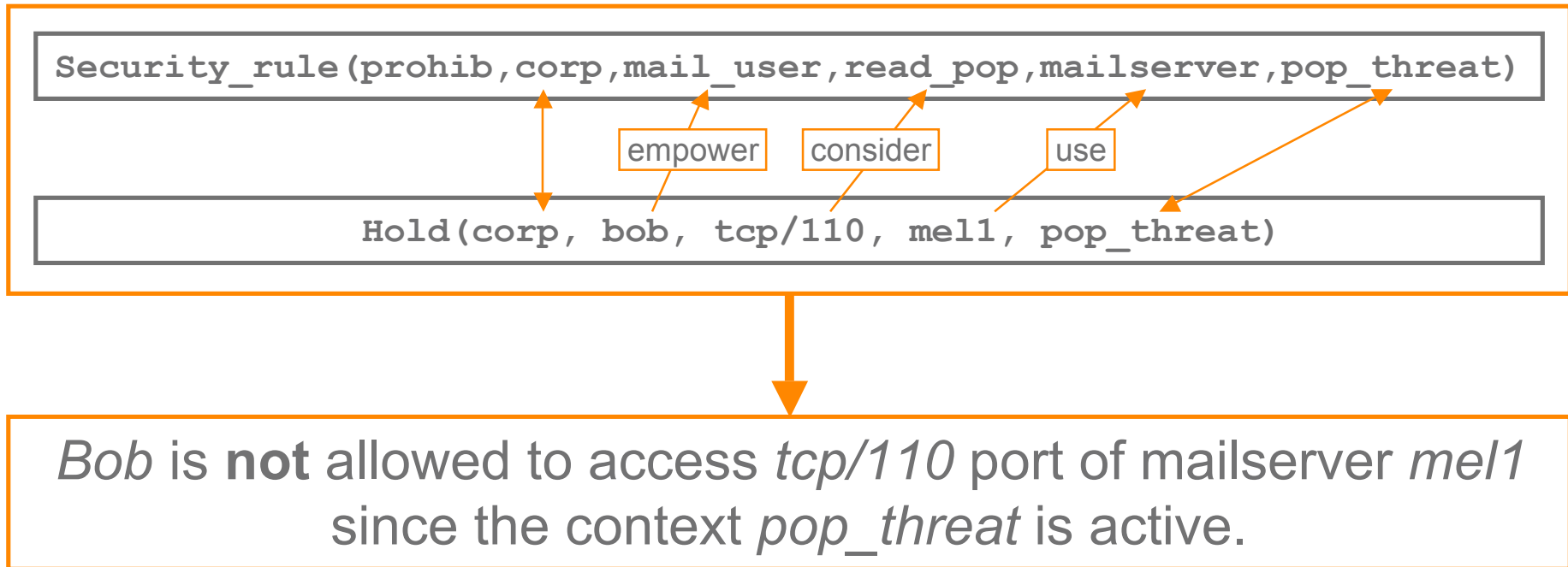
- Trigger a *hold(org, subject, action, object, context)* from alerts ensuring adequate response to the threat
- Example
 - `hold(corp, bob, tcp/110, mel1, pop_threat)`



From alerts to new policies (2)



- ➔ Derive concrete permissions/prohibitions (new policies) from *security_rules* and *hold* facts



From alerts to new policies (3)



- ➔ Concrete permissions/prohibitions managed by the PDP
 - Deployment: Adapt new policy instances into a concrete enforcement strategy
 - Block a port on a firewall,
 - Stop/reconfigure a service,
 - Etc.
 - Translation: Adapt policy rules to PEPs type and implementation
 - Type: “A firewall rule”
 - Implementation: “A Netfilter firewall rule”

- ➔ PEPs receive new configurations by the PDP to enforce the new policy

Minimal requirements



- Contexts allow expression of minimal requirements
 - Ex: 3 different paths to read mail (pop, imap and webmail)
 - During working hours, availability is considered more important than confidentiality and integrity for mail
 - If all paths to mail are threatened, re-open webmail to fulfill availability requirement, whatever the threat

```
Security_rule (perm, corp, mail_user, read_webmail, mailserver, minimal)
```

```
With minimal=pop_threat&imap_threat&webmail_threat&working_hours)
```

Conclusion



- ➔ Keeping a sensitive path open to maintain availability is questionable?
 - Availability is a crucial requirement
 - Other means can be deployed to ensure confidentiality and integrity
 - In particular, responses can be defined to switch between different requirements of authentication, ciphering, etc.
 - Provisional authorizations

- ➔ Results are encouraging
 - An implementation of the PIE/PDP in Prolog confirms the feasibility of the approach

Future work



- Mapping is a great part of the work in progress
 - Provide relevant hold facts to ensure adequate responses
 - Scale of the response

- Context lifetime
 - In a first time, static context lifetime based on expertise
 - Next step: characterizing the absence of threat (anti-alerts?)

- Experiments

Questions?

