

# Policy-Based Response to Intrusions Through Context Activation

Yohann THOMAS

Orange Labs - ENST Bretagne

November 23rd, 2007

# Outline

- 1 ■ Introduction
- 2 ■ Response Component
- 3 ■ Case Study
- 4 ■ Architecture
- 5 ■ Conclusion and Perspectives

# Context: Information System Security

- Operational security: 3 properties to ensure
  - Confidentiality, Integrity and Availability.
- Properties ensured with respect to a Security Policy
  - Set of requirements, more or less formal, describing *which subject is authorized to do which action on which object*,
  - Ex: `is_permitted(pc-bob,tcp/110,mel1)` means that *pc-bob* is permitted to access port *tcp/110* on server *mel1*.
- Violations of the security policy detected by Intrusion Detection Systems (IDS).
  - Successful attacks lead to intrusions,
  - Report **alert** in case of intrusion.

# IDS: State-of-the-Art

- **Sensors:** provide alerts,
  - Host-based or network-based,
  - Two detection methods
    - Misuse: detect unauthorized activity (mainly signature-based),
    - Anomaly: detect behavior deviating from normal use.
- **SIM platforms (Security Information Management):**  
relevance of the alerts?
  - Alert correlation,
  - Vulnerability assessment,
  - Actionable diagnosis? Which actions?

## Objective

Focus on response.

# Base Data: Relevant IDMEF Alert

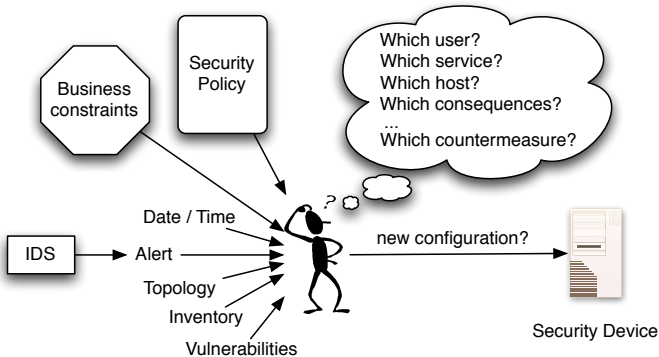
```
<IDMEF-Message><Alert messageid="5433859457"><Analyzer
analyzerid="2372811654396406" name="prelude-manager"
manufacturer="http://www.prelude-ids.com" model="Prelude
Manager" version="0.9.1" class="Concentrator" ostype="Linux"
osversion="2.6.10-1-686"><Process><name>prelude-
manager</name><pid>17649</pid><path>/usr/local/bin/
prelude-manager</path></Process><Analyzer
analyzerid="801489279220101" name="prelude-lml"
manufacturer="http://www.prelude-ids.com" model="Prelude
LML" version="0.9.1" class="Log Analyzer" ostype="Linux"
osversion="2.6.10-1-686"><Process><name>prelude-lml</
name><pid>17652</pid><path>/usr/local/bin/prelude-lml</
path></Process><Analyzer name="PAM"
class="Authentication"><Node
category="unknown"><name>localhost.localdomain</
name><Address category="ipv4-addr"><address>127.0.0.1</
address></Address></Node><Process><name>sshd</
name><pid>17656</pid></Process></Analyzer></Analyzer>
<CreateTime ntpstamp="0xc78c9c1a.
0x0604f000">2006-02-02T15:56:26.23513+01:00</
CreateTime><DetectTime
ntpstamp="0xc78c9c19.0x00000000">2006-02-02T15:56:25.00
+01:00</DetectTime><AnalyzerTime ntpstamp="0xc78c9c1a.
0x0644f000">2006-02-02T15:56:26.24489+01:00</
AnalyzerTime><Source spoofed="unknown"><Node
category="unknown"><name>pc-charlie</name><Address
category="ipv4-addr"><address>192.168.1.13</address></
Address></Node><Source><Target decoy="unknown"><Node
category="unknown"><Address category="ipv4-
addr"><address>192.168.2.50</address></Address></
Node><Process><name>popd</name><path>/etc/popd</
path></Process></Target><Classification text=""><Reference
origin="o" meaning="m"><name>CVE-1999-0822</name></
Reference></Classification><Assessment><Impact
severity="high" completion="succeeded" type="admin"></
Impact></Assessment></Alert></IDMEF-Message>
```

- This one is “simple”,
- Many information provided:
  - CreateTime: date of emission,
  - Source: here, the attacker,
  - Target: here, the victim,
  - Classification: vulnerability reference,
  - etc.
- In the following:  
alert(CreateTime,Source,Target,Classif)

# Which Countermeasure?

- Having a look at the alert, we observe that:
  - **Source:** reports that `pc-charlie` is the attacker,
  - **Target:** port `tcp/110` on server `mel1` is the victim,
  - **Classification:** `CVE-1999-0822` is a reference indicating an attack towards the pop service.
- Many possible countermeasures (response strategy?):
  - Forbid access to port `tcp/110` for:
    - `pc-charlie` on server `mel1` (specific countermeasure),
    - any host on server `mel1` (victim-centric),
    - `pc-charlie` on any mail server (attacker-centric),
    - any host on any mail server (larger scale),
  - Shutdown pop process associated to port `tcp/110`,
  - Block access for user to mail repository,
  - Possibly others?

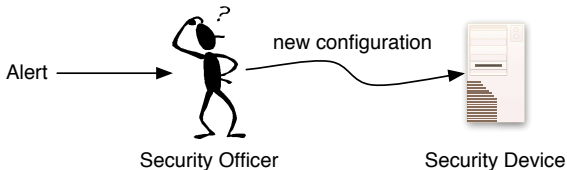
# Response: Complex Decision



- Security officer responsible for threat analysis and choice of response.
  - Tremendous analysis task leads to a lack of **reactivity**,
  - Response **relevancy** issue.

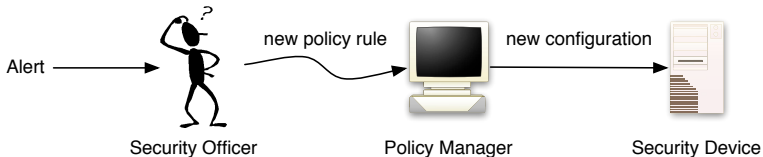
# Ad-hoc Response

- No formal security policy, informal recommendations only,
- Security Officer manually deploys new configurations over security devices (*e.g. filtering rule on a firewall*),
- Inadapted technique
  - Patch'n pray approach,
  - Lack of coherence and consistency.



# Policy-Based Response

- Formal security policy,
- Security Officer adds/deletes policy rules *w.r.t.* threat,
- Requires the corresponding architecture so that policy instances are correctly enforced over security devices.
- Better approach to ensure coherence and consistency,
- **But:** security officer still in charge of response.



# Automating Policy-Based Response

- Most access control models are inadapted (**static** policies),
  - *If bob is permitted to read mail through POP, he is always permitted to do so.*
- **But:** changes are needed in case of **threat!**
- **Moreover:** many other parameters to consider in addition to security
  - Performance, convenience, business constraints, etc.
- **State-of-the-art:** trade-off between these multiple adjustment variables is generally **static**.

## Objective

Render this trade-off **dynamic**, so that security policy requirements automatically adapt to **context** (including threat).

# Organization-Based Access Control

- Or-BAC is a model allowing the definition of a **contextual** security policy (**dynamic** policy).
  - Model entities to manage (e.g. *pc-bob*, *tcp/110*, *mel1*),
  - Define contextual policy rules,
  - Activate contexts,
  - Derive policy instances.

## Proposal

Make use of Or-BAC to automate threat response.

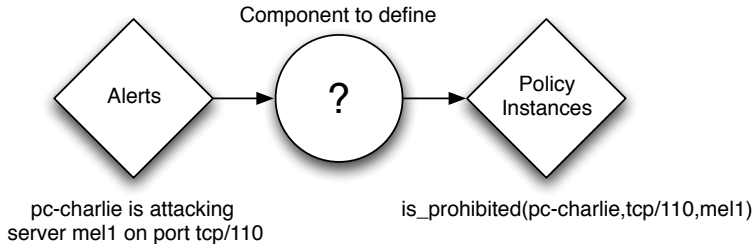
- Define contextual policy rules describing what should be authorized or not in case of threat,
- Activate **threat contexts** thanks to **alerts** to derive **policy instances** in response to threat.

# Outline

- 1 ■ Introduction
- 2 ■ Response Component
- 3 ■ Case Study
- 4 ■ Architecture
- 5 ■ Conclusion and Perspectives

# From Alerts to Policy Instances

- Design a component which replaces the security officer.
  - Assess current context (especially threat),
  - Decide which response is appropriate,
  - Provide corresponding new policy instances to deploy.



# Model Entities to Manage

- In Or-BAC, a policy is relevant within an **organization**,
- **Concrete level**
  - **subjects** make **actions** on **objects**.
- **Abstract level**
  - *subjects* abstracted into **roles**,
  - *actions* abstracted into **activities**,
  - *objects* abstracted into **views**,

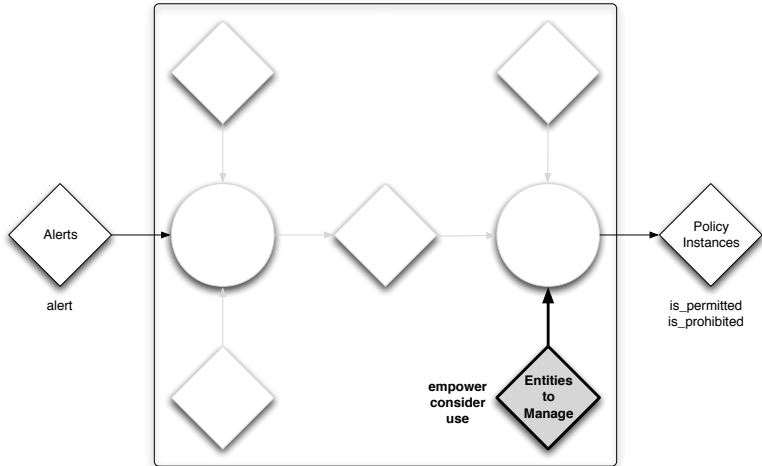
## Formalism

```
Empower(Org,Subject,Role),  
Consider(Org,Action,Activity),  
Use(Org,Object,View).
```

## Example

```
Empower(corp,pc-bob,mail_station),  
Consider(corp,tcp/110,read_pop),  
Use(corp,mel1,mail_server).
```

# Model Entities to Manage



# Define Contextual Policy Rules

- Permissions and prohibitions at the abstract level,
- Context parameter renders the policy dynamic.

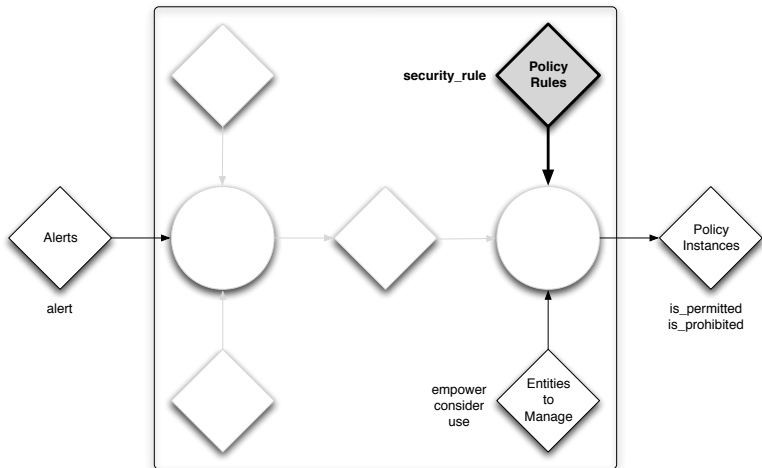
## Formalism

Security\_rule(Type, Organization, Role, Activity, View, **Context**),  
With Type  $\in$  {permission, prohibition}

## Example

```
sr(perm,corp,mail_station,read_pop,mail_server,working_hours)
```

# Define Contextual Policy Rules



# Activate Contexts

- **Context** is **held** for a given **subject** making a given **action** on a given **object**.

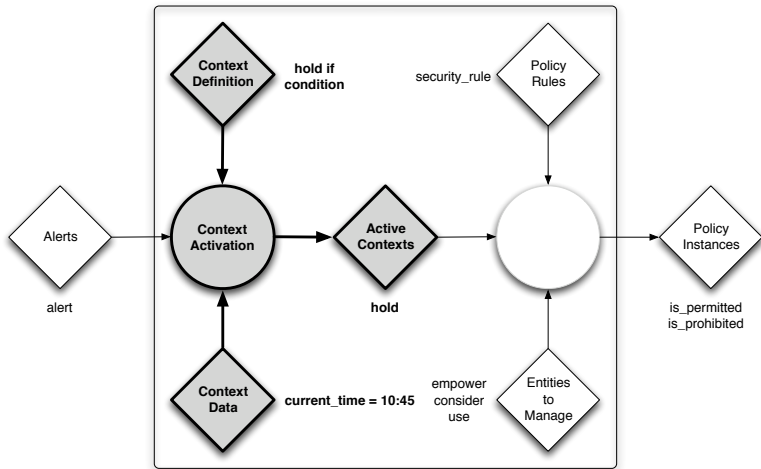
## Formalism

Hold(Organization, Subject, Action, Object, **Context**)

## Example

```
hold(corp, _, _, _, working_hours) :-  
  globalclock(DayClock, TimeClock),  
  TimeClock >= '07:00:00',  
  TimeClock < '20:00:00',  
  DayClock != 'saturday',  
  DayClock != 'sunday'.
```

# Activate Contexts



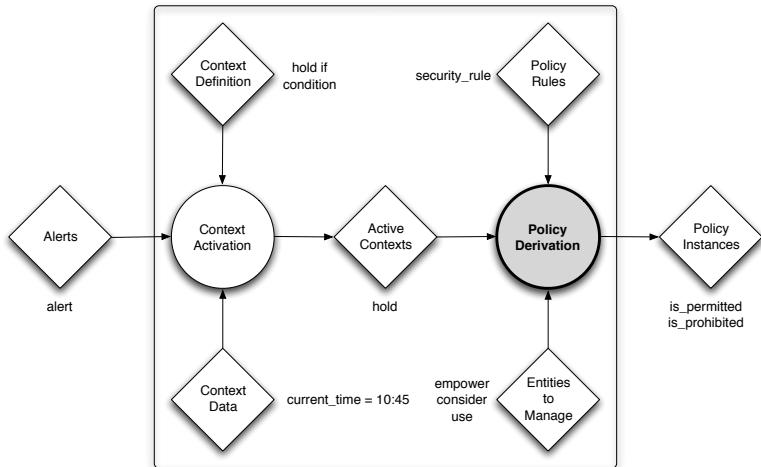
# Derive Policy Instances

- Matching **security rules** with currently activated **hold** facts,
- Derive **concrete authorizations**.

## Example

```
sr(perm,corp,mail_station,read_pop,mail_server,working_hours),  
empower(corp, pc-bob, mail_station),  
consider(corp, tcp/110, read_pop),  
use(corp, mel1, mail_server),  
hold(corp, pc-bob, tcp/110, mel1, working_hours)  
→ is_permitted(pc-bob, tcp/110, mel1).
```

# Derive Policy Instances



# Dealing with Threat

- We can define contextual policy rules to deal with threat.

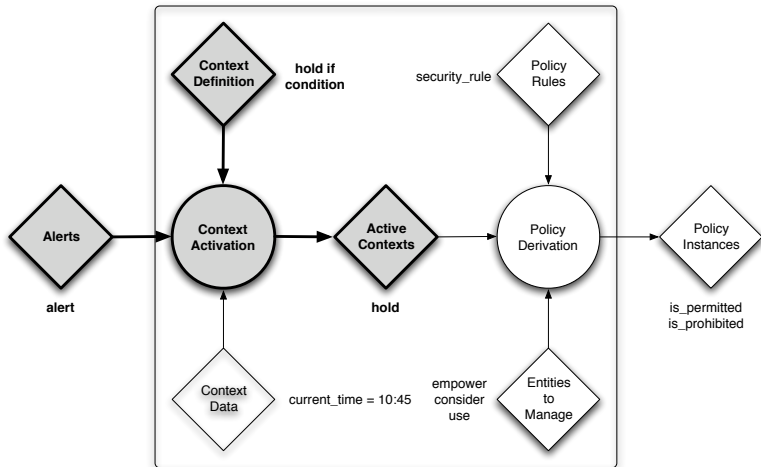
## Another example

```
sr(prohib,corp,mail_station,read_pop,mail_server,pop_attack),
empower(corp, pc-charlie, mail_station),
consider(corp, tcp/110, read_pop),
use(corp, mel1, mail_server),
hold(corp, pc-charlie, tcp/110, mel1, pop_attack)
→ is_prohibited(pc-charlie, tcp/110, mel1).
```

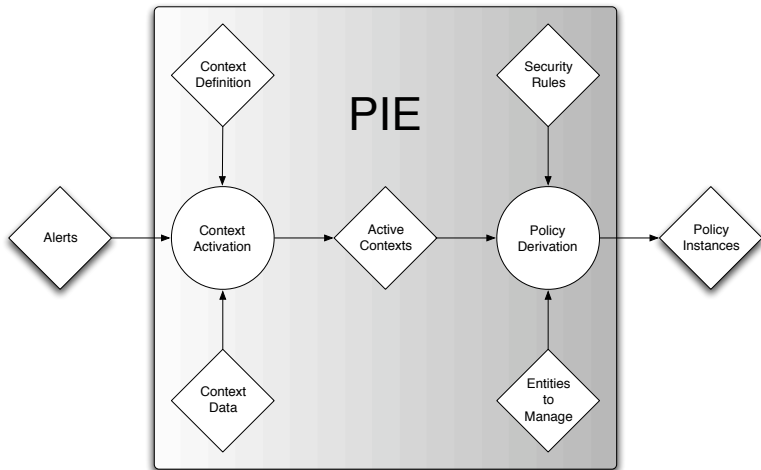
## Core of our proposal

If we can build **threat contexts** from **alerts**, we can provide new **policy instances** in response to threat.

# Activate Threat Contexts



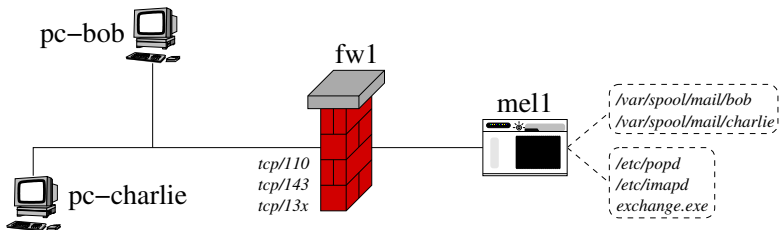
# Result: Policy Instantiation Engine



# Outline

- 1 ■ Introduction
- 2 ■ Response Component
- 3 ■ Case Study**
- 4 ■ Architecture
- 5 ■ Conclusion and Perspectives

# Email Application



- Users Bob and Charlie access their mailboxes on server mel1,
- Server provides 3 services for mail reading: pop, imap, exchange,
- Users may use various mail clients for mail reading.

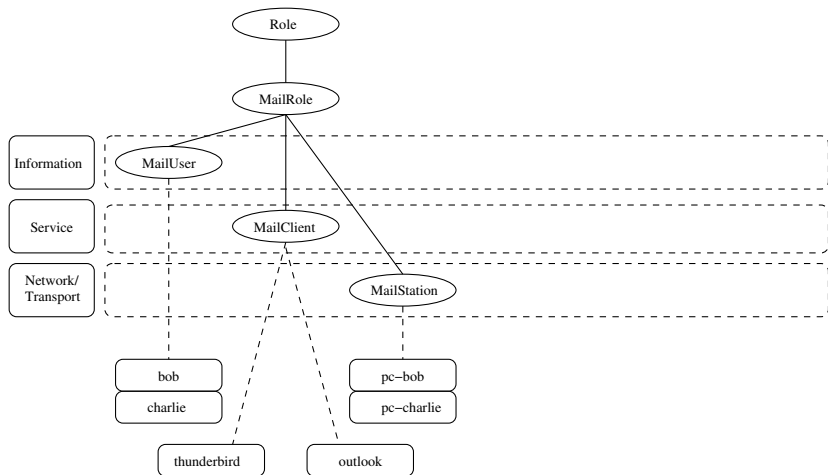
# Managing Structured Entities

- Alerts report many information to map on subjects, actions and objects:
  - ip address, dns name, netmask, port number,
  - client and server processes,
  - user, resource, file permission.
- Expected countermeasures:
  - Port filtering on firewall fw1,
  - Process terminating on server mel1,
  - File access right modification on server mel1 (mail repository).

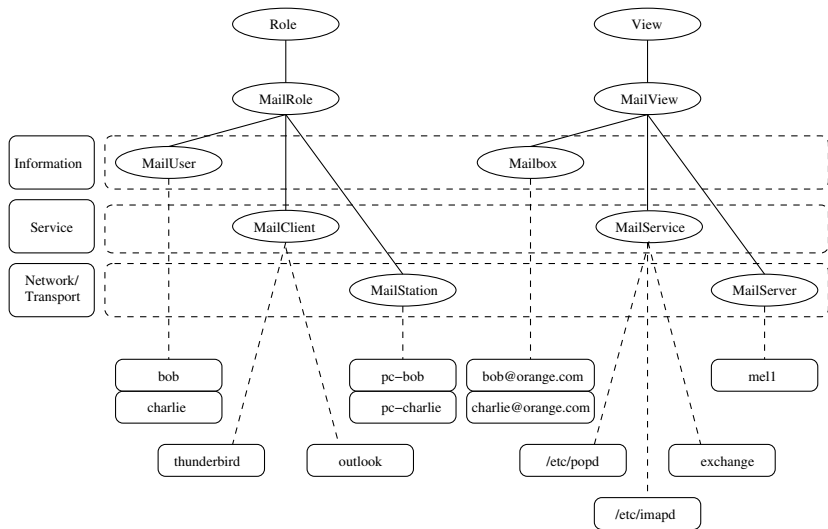
## Need for Structured Entities

Three layers appear: network/transport, service, information.

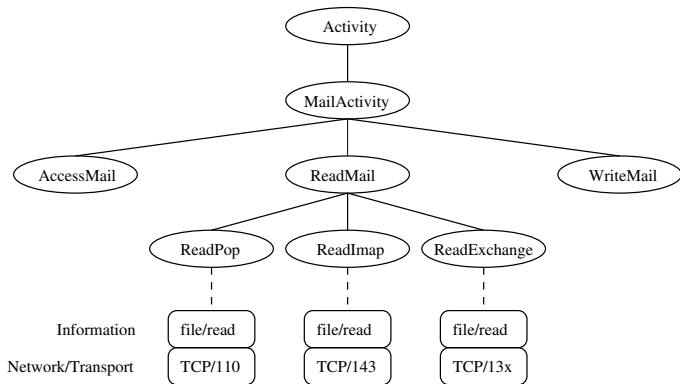
# Modeling Roles and Views



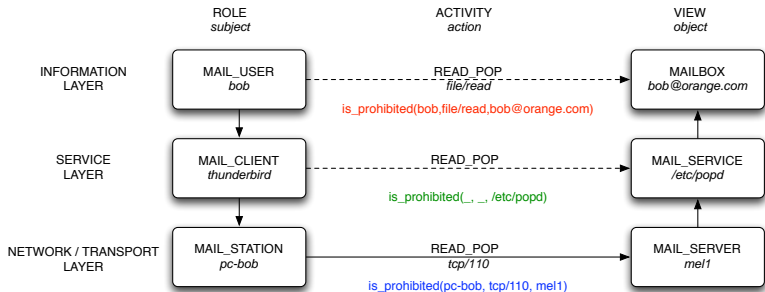
# Modeling Roles and Views



# Modeling Activities



# Our Three-layer Model



- Multi-level response allowing fine-grained countermeasures
  - Information: *e.g. file permission modification,*
  - Service: *e.g. process termination,*
  - Network/Transport: *e.g. port filtering.*

# Defining a Reaction Policy

## ■ Operational policy

- What is authorized or not by default in the absence of threat,
  - *Ex: a mail user can read his mailbox.*

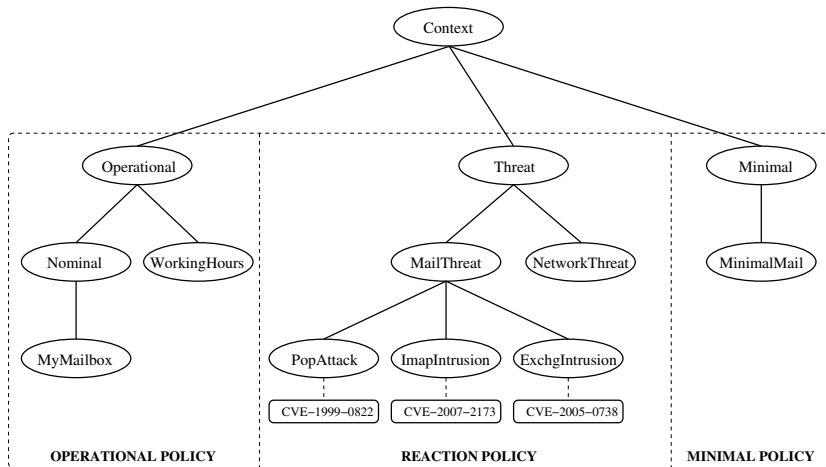
## ■ Reaction policy

- What is authorized or not in case of threat.
  - *Ex: in case of pop threat, a mail user cannot read his mailbox through POP.*

## ■ Minimal policy

- What must be ensured whatever the threat.
  - *Ex: in case of threat against all mail reading services, preserve access to one of them.*

# Modeling Corresponding Contexts



# Operational Policy

- Nominal requirements (default policy),
- Constraint at the information layer: a mail user must not access other mailboxes (*myMailbox* context).

## Policy Rules Definition

```
sr(permission,corp,mail_role,mail_activity,mail_view,nominal).  
sr(prohibition,corp,mail_user,mail_activity,mailbox,not(myMailbox)).
```

## Context Definition

```
hold(corp,_,_,_,nominal).  
  
hold(corp,Subject,Action,Object,myMailbox) :-  
    empower(corp,Subject,mail_user),  
    consider(corp,Action,mail_activity),  
    use(corp,Object,mailbox),  
    active_directory_entry(Subject,Object).
```

# Reaction Policy

- Response requirements.

## Policy Rules Definition

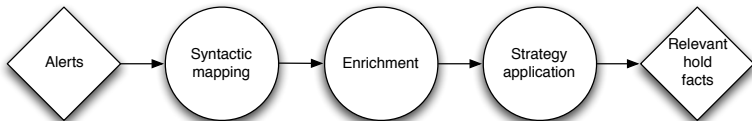
```
sr(prohibition,corp,mail_role,read_pop,mail_view,pop_attack).  
sr(prohibition,corp,mail_role,read_imap,mail_view,imap_intrusion).  
sr(prohibition,corp,mail_role,read_exchange,mail_view,exch_intrusion).
```

## Context Definition

```
hold(corp,Subject,Action,Object,Context) :-  
    alert(CreateTime,Source,Target,Classification),  
    map_subject(Source,Target,Classification,Subject),  
    map_action(Source,Target,Classification,Action),  
    map_object(Source,Target,Classification,Object),  
    map_context(Source,Target,Classification,Context).
```

# From Alerts to hold Facts

- Three step process:
  - **Syntactic mapping:** Get information available in alerts to build subjects, actions and objects,
  - **Enrichment:** Possibly add information missing in alerts,
  - **Strategy application:** Activate **relevant hold facts** to obtain appropriate countermeasures.



# Response Strategy

- Multiple hold facts may match a same **security rule**,
  - `sr(prohibition,corp,mail_role,read_pop,mail_view,pop_attack)`
- Which **hold** fact(s) to retain for **most appropriate** response?
  - Network/Transport layer:
    - Specific : `hold(corp,pc-charlie,tcp/110,mel1,pop_attack)`
    - Victim-centric : `hold(corp,_,tcp/110,mel1,pop_attack)`
    - Attacker-centric: `hold(corp,pc-charlie,tcp/110,_,pop_attack)`
    - Larger scale : `hold(corp,_,tcp/110,_,pop_attack)`
  - Service layer:
    - `hold(corp,_,_,etc/popd,pop_attack)`
  - Information layer:
    - `hold(corp,charlie,file/read,charlie@orange.com,pop_attack)`
- Considered elements:
  - Vulnerability reference (expert knowledge),
  - Alert severity,
  - Spoofed source.

# Minimal Policy

- Minimal requirement: preserve POP access in case of threat towards all three mail reading services.

## Policy Rules Definition

```
sr(permission,corp,mail_role,read_pop,mail_view,minimal_mail).
```

## Context Definition

```
hold(corp,Subject,_,Object,minimal_mail) :-  
    hold(corp,Subject,_,Object,pop_attack),  
    hold(corp,Subject,_,Object,imap_intrusion),  
    hold(corp,Subject,_,Object,exchg_intrusion).
```

# Conflict Resolution

We can derive policy instances from the 3 sub-policies, but:

- Some authorizations may not be compatible.
  - **Nominal:** `is_permitted(pc-charlie,tcp/110,mel1)`,
  - **Pop attack:** `is_prohibited(pc-charlie,tcp/110,mel1)`.
- **Conflict:** A subject is both permitted and prohibited to make an action on an object.
  - At the **concrete** level: **actual conflicts**,
  - At the **abstract** level: **potential conflicts**.
- Potential conflicts may lead to actual conflicts at policy derivation.

## Objective

Solve **potential conflicts**, so that no actual conflict can occur at policy derivation.

# Conflict Resolution

## Prioritized Or-BAC model

Provide a partial order between potentially conflicting rules to assess rules precedence in case of conflicts.

## Sample potential conflicts

```
sr(corp,permission,mail_station,read_pop,mail_server,nominal,p1).  
sr(corp,prohibition,mail_station,read_pop,mail_server,pop_attack,p2).  
sr(corp,permission,mail_station,read_pop,mail_server,minimal_mail,p3).  
higherLevel(p2,p1), higherLevel(p3,p2).
```

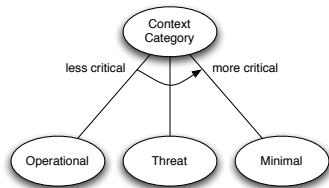
## Conflict Resolution Strategy (CRS)

- Conditions allowing the system to decide which rule takes precedence in case of conflict,
- Automate assessment of *higherLevel* facts.

# Conflict Resolution

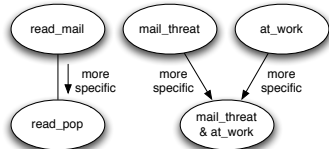
## Criticality

- Order context categories:  
 $operational <_c threat <_c minimal$

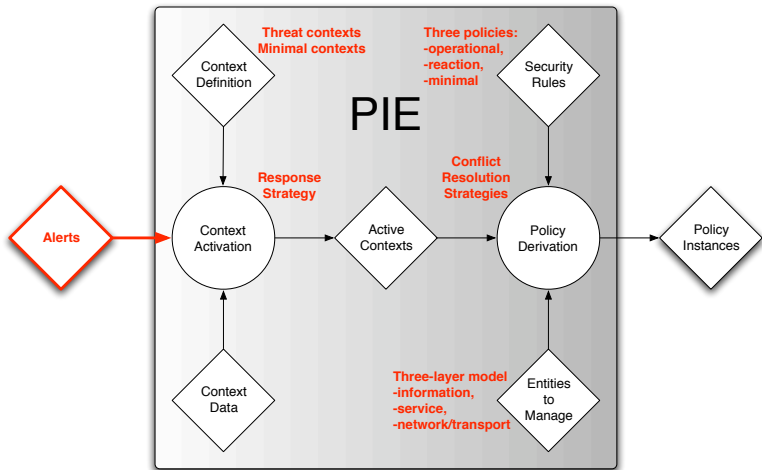


## Specificity

- An inherited entity is more specific than its parent:  
 $read\_mail <_s read\_pop,$
- A composed context is more specific than its composing contexts:  
 $mail\_threat <_s mail\_threat \& at\_work,$   
 $at\_work <_s mail\_threat \& at\_work.$



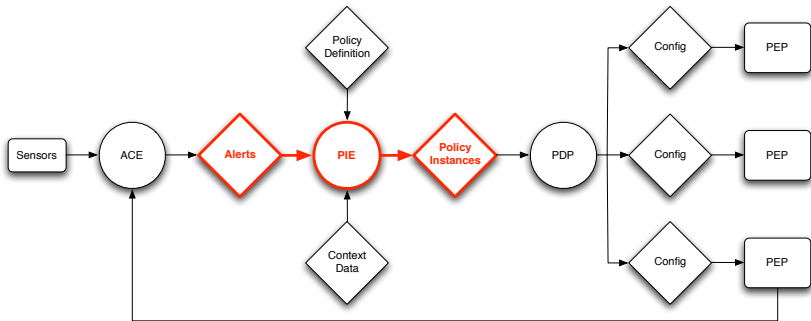
# Result: Our Contributions



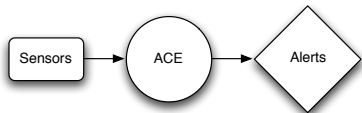
# Outline

- 1 ■ Introduction
- 2 ■ Response Component
- 3 ■ Case Study
- 4 ■ Architecture**
- 5 ■ Conclusion and Perspectives

# Resulting Architecture



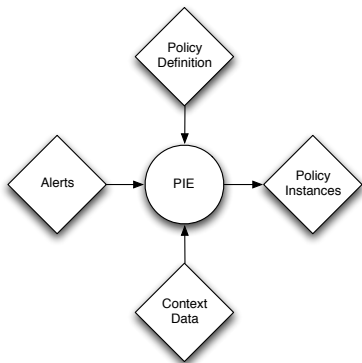
# Alert Correlation Engine (ACE)



SIM Component.

- Centralize sensor outputs,
- Improve intrusion detection diagnosis,
- Normalize output (IDMEF),
- Ensure alert reporting.

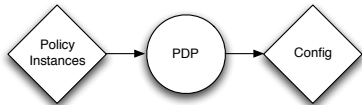
# Policy Instantiation Engine (PIE)



Our response component to replace the security officer.

- Make global decisions,
- Characterize threat & response,
- Apply response strategy,
- Update policy instantiation.

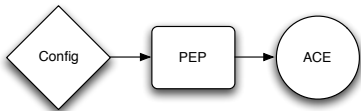
# Policy Decision Point (PDP)



Component in charge of policy deployment.

- Make local decisions,
- Translate policy instances into configurations,
- Apply potential local strategy,
- Push configurations to PEPs.

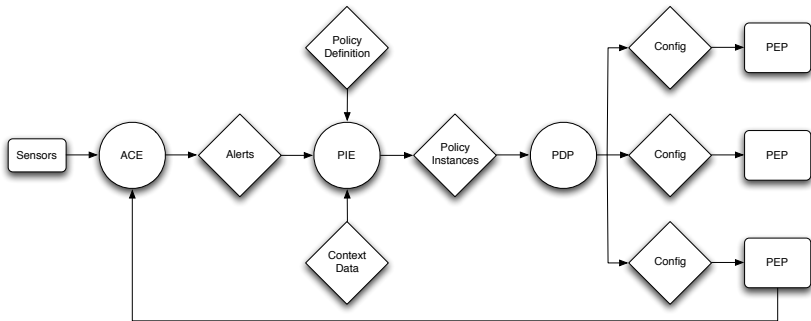
# Policy Enforcement Point (PEP)



Component in charge of policy enforcement.

- Receive configurations to enforce,
- Implement current policy instantiation,
- Act as sensors for ACE.

# Resulting Architecture



# Prototype

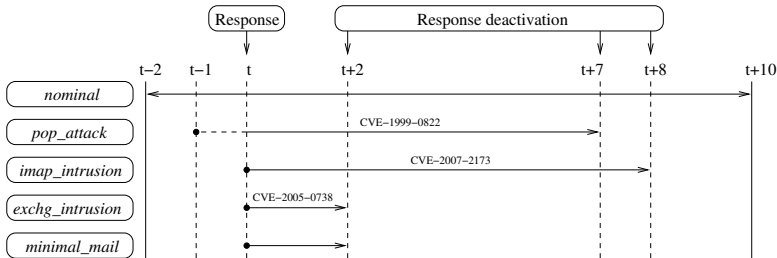
- Policy Instantiation Engine (PIE)
  - First-order logic-based implementation (Prolog),
  - Perl sequencer to ensure temporal consistency and updates.
- Policy Decision Point (PDP)
  - Ad-hoc translation of policy instances into configurations,
- Policy Enforcement Points (PEPs)
  - Local agent running for command enforcement.

# Experiment

- Injection of three well-chosen alerts,
  - Results of the exploit of Exchange, POP and IMAP vulnerabilities,
  - Host pc-charlie always seen as the attacker,
  - Host mel1 always seen as the victim,
  - Different response strategies.
- Alerts asserted for a given lifetime, computed through expert knowledge (severity and type of attack).
  - Threat contexts are thus active for a given duration,
  - Threat contexts are re-activated if threat has not disappeared.

# Experiment

- Alert sequencement for pc-charlie attacking server mel1:



- We obtain expected policy instances,
- Proof-of-feasibility of our proposal.

# Outline

- 1 ■ Introduction
- 2 ■ Response Component
- 3 ■ Case Study
- 4 ■ Architecture
- 5 ■ Conclusion and Perspectives

# Contributions

- Definition of a security policy including reaction and minimal requirements,
- Activation of threat contexts based on incoming alerts and allowing threat response,
- Definition of a CRS automating conflict resolution,
- Proposal of an architecture of a threat response system taking into account the required constraints.

# Conclusion

## ■ Automated Response

- A component which allows to gain in reactivity and which is able to apply a response strategy matching the threat.

## ■ Dynamic Trade-off

- A continuous assessment of the best trade-off between multiple adjustment variables, including security.

## ■ Policy-based Approach

- Connection between security policies and monitoring (IDS).

# Conclusion

- Results are encouraging (proof-of-feasibility),
- Approach developed in the CELTIC RED european project,
- Many perspectives identified.

# Future Work

Many perspectives identified to improve and extend the proposal, including:

- Extension of the approach to other mechanisms,
  - Other services: file (*NFS/CIFS*), database (*grant/revoke*), etc.
  - Cryptographic means: constrain more or less authentication and encryption.
- Model dependencies,
  - Dependencies between layers (*network, service, information*),
  - Infrastructure service dependencies (e.g. *DHCP, DNS*),
  - Application service dependencies (e.g. *dynamic web service requires HTTP+DB+PHP*).
- Test the approach in operational environment.

Thank you.