LISA '11
Fine-grained access-control for the Puppet configuration language

Bart Vanbrabant, Joris Peeraer and Wouter Joosen

DistriNet, Dept. of Computer Science,
K.U.Leuven, Belgium

December 7, 2011

Outline

Systems configuration

Context

Problems

Our solution: ACHEL

Authorising Puppet

Conclusion

Outline

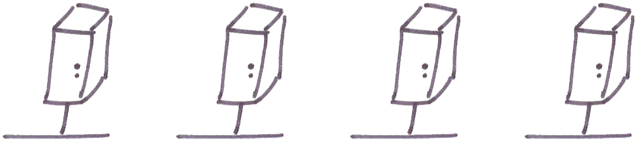
Systems configuration
- Context
- Problems

Our solution: ACHEL

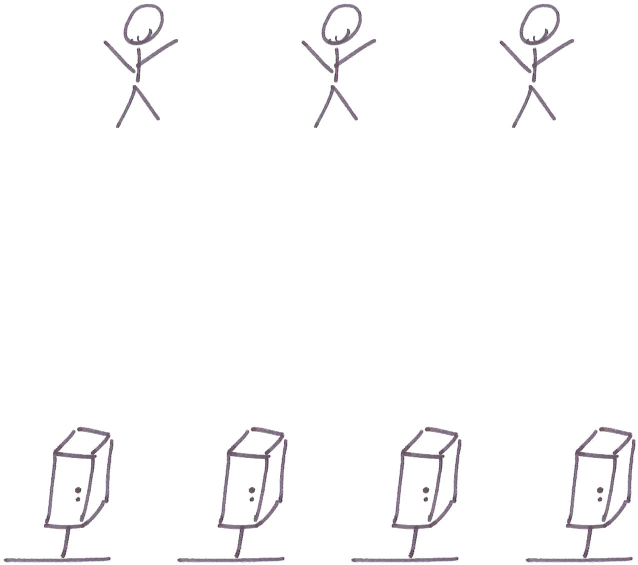
Authorising Puppet

Conclusion

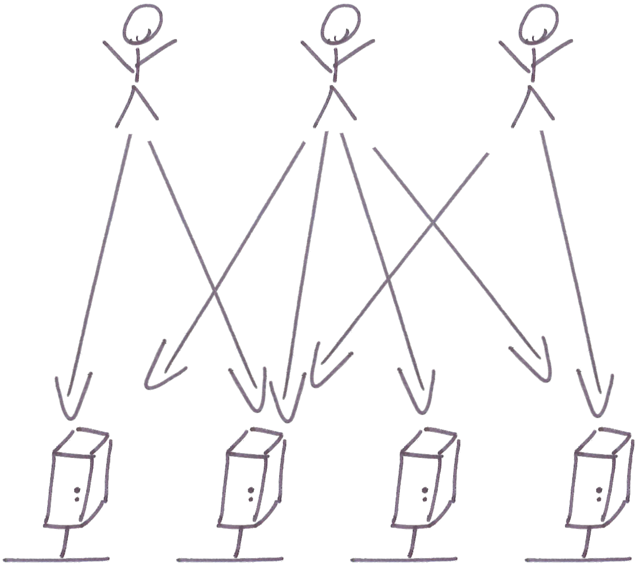
System configuration tools



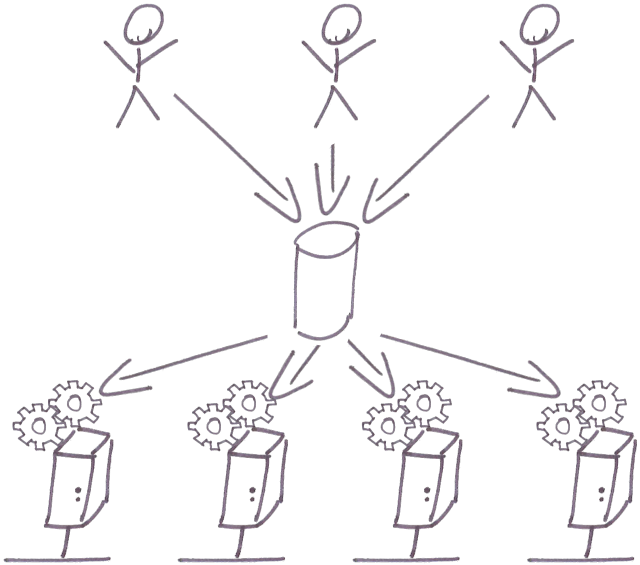
System configuration tools



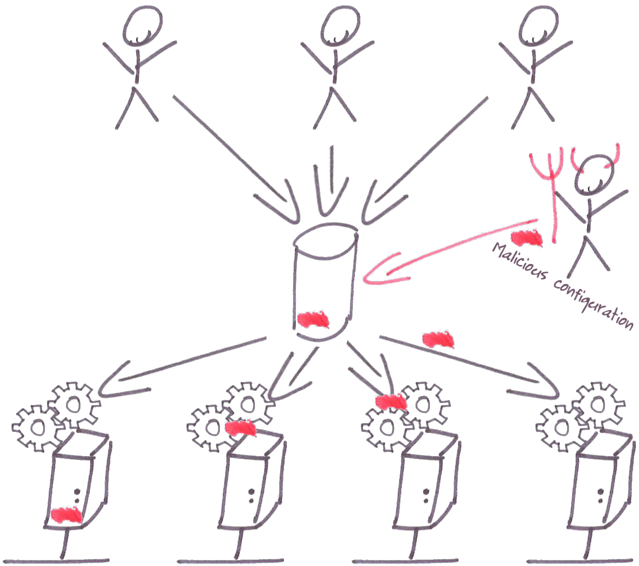
System configuration tools



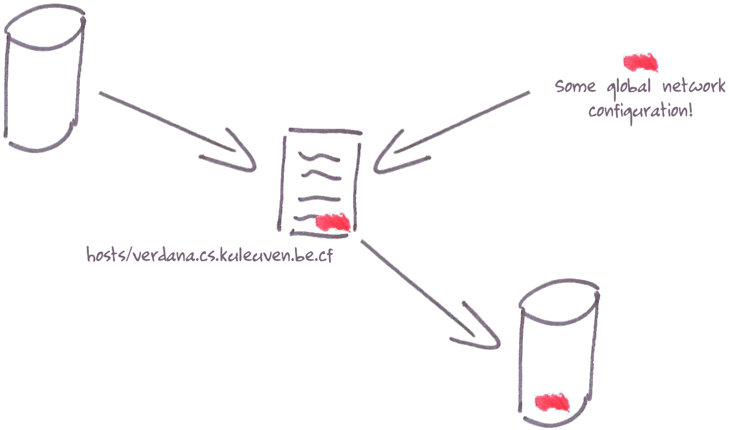
System configuration tools



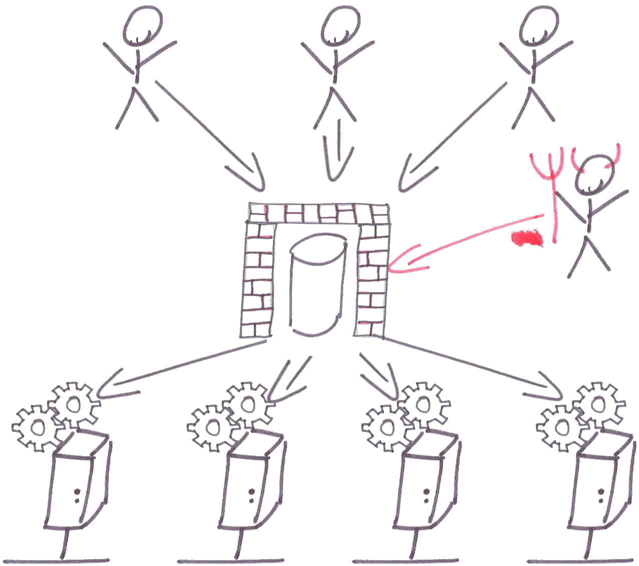
System configuration tools



System configuration tools


Some global network configuration!
hosts/verdana.cs.kuleuven.be.cf

System configuration tools



Outline

Systems configuration

- Context
- Problems

Our solution: ACHEL

Authorising Puppet

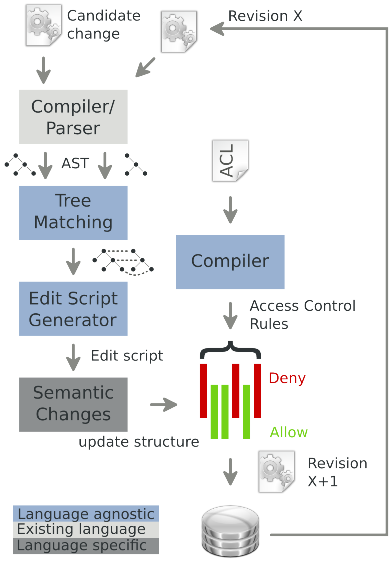
Conclusion

What is ACHEL?

ACHEL manages access to *repositories* of *configuration specification* by implementing *access control* and enforcing *workflows*

- *fine-grained* access control interpreting the *semantics* of *changes*
- The actions that needs authorisation are derived automatically
- *access control* is applied at the *abstraction level* of the configuration specification
- support for workflow in *federated* infrastructures
- a (configuration) *language agnostic* solution

Generating meaningful changes with ACHEL



Outline

KATHOLIEKE UNIVERSITEIT LEUVEN DistriNet RESEARCH GROUP

Systems configuration
- Context
- Problems

Our solution: ACHEL

Authorising Puppet

Conclusion

Puppet

Authorise changes to the configuration model of a real tool:

- System management tool used in production environment
- Puppet has an expressive and *complex* configuration language
- Manifests organised in modules
- Authorisation based on modules and their file path
- Link between contents of module and its name is not enforced

Applying ACHEL to Puppet

Steps to authorise changes the ACHEL way:

- Aquire the AST from Puppet
- AST contains syntax so normalisation is required
- Derive to be authorised actions
- Submit request to XACML policy engine
- Report result of authorisation

AST normalisation

Define three users with one statement:

```
user {["bart", "joris", "wouter"]:
}
```

Define three users with three statements:

```
user {"bart":
}
user {"joris":
}
user {"wouter":
}
```

Prototype

Challenges for prototype:

- Not all language features supported, some are impossible to support
- Prototype extracts AST from Puppet compiler and normalises it
- The AST is serialised to XML so XPath can be used in policies
- Prototype is integrated in a DVCS (Bazaar) to enforce access control

Example: Adding vhosts

Puppet manifest:

```
# Apache-class
class apache {
...
}

# vhost definition
define apache::vhost ($document_root) {
    file {"/etc/apache2/vhosts-available/${name}":
        ensure  => present,
        docroot => $document_root,
    }
}

node a {
    include apache
}
```

Example: Adding vhosts

User Jdoe adds a virtual host:

```
# Apache-class
class apache {
    apache::vhost {"www.example.com":
        docroot => "/home/jdoe/public_html",
    }
...
}

# vhost definition
define apache::vhost ($document_root) {
    file {"/etc/apache2/vhosts-available/${name}":
        ensure  => present,
        docroot => $document_root,
    }
...
```

Example: Adding vhosts

Result from matching:

```
* Updated: none

* Inserted:
  Add member: Resource (title:www.example.com,
        type:apache::vhost)
  Add parameter: ResourceParam (param:docroot)
  Add value: String () => /home/jdoe/public_html

* Removed: none
```

Example: Adding vhosts

KATHOLIEKE UNIVERSITEIT LEUVEN DistriNet RESEARCH GROUP

XAMCL policy extract (without the namespace clutter)

```
<Policy>
 <Description>Apache permissions for webuser</Description>
  <Target><Subjects><Subject><SubjectMatch>
   <AttributeValue>webuser</AttributeValue>
   <SubjectAttributeDesignator AttributeId="subject:role" />
  </SubjectMatch></Subject></Subjects></Target>
 <Rule Effect="Permit">
  <Description>Add or remove a vhost</Description>
  <Target><Resources><Resource><ResourceMatch>
   <AttributeValue>//pup:*[@type="apache::vhost"]</AttributeValue>
   <ResourceAttributeDesignator AttributeId="resource-id" DataType="xpath-expression" />
  </ResourceMatch></Resource></Resources></Target>
 </Rule>
 <Rule Effect="Permit">
  <Target><Resources><Resource><ResourceMatch>
   <AttributeValue>//pup:*[@type="apache::vhost"]/pup:*[@param="docroot"]</AttributeValue>
   <ResourceAttributeDesignator AttributeId="resource-id" DataType="xpath-expression" />
  </ResourceMatch></Resource></Resources></Target>
  <Condition>
   <Apply FunctionId="string-starts-with"><Apply FunctionId="string-one-and-only">
    <AttributeSelector RequestContextPath="//pup:*[@param='docroot']/pup:value/text()" />
   </Apply>
   <Apply FunctionId="string-concatenate">
    <AttributeValue>/home/</AttributeValue>
    <Apply FunctionId="string-one-and-only">
     <SubjectAttributeDesignator AttributeId="subject-id" />
    </Apply>
   </Apply></Apply>
  </Condition>
 </Rule>
</Policy>
```

Example: Adding vhosts

First rule from extract:

```
<Policy>
 ...
 <Rule Effect="Permit">
  <Description>Add or remove a vhost</Description>
  <Target><Resources><Resource><ResourceMatch>
   <AttributeValue>//pup:*[@type="apache::vhost"]
    </AttributeValue>
   <ResourceAttributeDesignator AttributeId="resource-id"
    DataType="xpath-expression" />
  </ResourceMatch></Resource></Resources></Target>
 </Rule>
 ...
</Policy>
```

Example: Adding vhosts

Second rule from extract:

```
<Policy>
 ...
 <Rule Effect="Permit">
  <Target><Resources><Resource><ResourceMatch>
   <AttributeValue>//pup:*[@type="apache::vhost"]/pup:*[@param="docroot"]</AttributeValue>
   <ResourceAttributeDesignator AttributeId="resource-id" DataType="xpath-expression" />
  </ResourceMatch></Resource></Resources></Target>
  <Condition>
   <Apply FunctionId="string-starts-with"><Apply FunctionId="string-one-and-only">
    <AttributeSelector RequestContextPath="//pup:*[@param='docroot']/pup:value/text()" />
   </Apply>
   <Apply FunctionId="string-concatenate">
    <AttributeValue>/home/</AttributeValue>
    <Apply FunctionId="string-one-and-only">
     <SubjectAttributeDesignator AttributeId="subject-id" />
    </Apply>
   </Apply></Apply>
  </Condition>
 </Rule>
</Policy>
```

Use unsupported language constructions

KATHOLIEKE UNIVERSITEIT LEUVEN DistriNet RESEARCH GROUP

- Policy defines what is allowed
- Usage of defines or classes can be authorised
- Encapsulate unsupported or complex Puppet constructions
- Authorise on the container of the unsupported statements

Outline

Systems configuration
- Context
- Problems

Our solution: ACHEL

Authorising Puppet

Conclusion

Conclusion

- ACHEL method supports complex languages
- Unsupported languages features using encapsulation
- Clean AST required
- XACML is powerful but hard to use