Policy-driven Reflective Enforcement of Security Policies
Ian Welch and Fan Lu
Want to separate out enforcement code from other code

In context of Java, allow application-level enforcement to be handled as transparently as system-level enforcement

But solution requires two policy files, one of which is imperative

Want to write one policy using a DSL and have everything else taken care of automatically
Example – chat client

IRC chat client (3rd party component). Standalone or used as part of another application. Aim is to enforce a local policy. Restrict access by user to given chat rooms or channels. Security implemented using a Kava metaobject protocol.
Kava

Metaobject redefines object behaviour.
Binding achieved at loadtime.
Similar to Aspects except crosscutting code is distributed across multiple metaobjects.
Reflective enforcement

```java
public class Lirc {...};
(new Lirc).createChannel(channelName);

public class EnforcementMetaObject extends MetaObject {
    ...
    public void beforeExecuteMethod(...) {
        ...
        sm.checkPermission(new ChannelPermission(
            joinChannel, "join");
    }
}

<intercept><execute><cnamelLirc</cname>
<method>createChannel</method></execute></intercept>

grant signedby WhiteHat {
    permission ChannelPermission allowedChannel, join;
}
```
Reflective enforcement

<table>
<thead>
<tr>
<th>Chat client</th>
<th>Program using client</th>
</tr>
</thead>
<tbody>
<tr>
<td>public class Lirc {...};</td>
<td>(new Lirc).createChannel(channelName);</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>public class EnforcementMetaObject extends MetaObject {</td>
<td>public class EnforcementMetaObject extends MetaObject {</td>
</tr>
</tbody>
</table>
| ... | ...
| public void beforeExecuteMethod(...) { | public void beforeExecuteMethod(...) { |
| ... | ...
| } } sm.checkPermission(new ChannelPermission( | } } sm.checkPermission(new ChannelPermission( |
| joinChannel, "join"); | joinChannel, "join"); |
| } } |
| <intercept><execute><cname>Lirc</cname> | <intercept><execute><method>createChannel</method></execute></intercept> |
| <method>createChannel</method></execute></intercept> |
| grant signedby WhiteHat { | grant signedby WhiteHat { |
| permission ChannelPermission allowedChannel, join; | permission ChannelPermission allowedChannel, join; |
| } } | } } |
### Reflective enforcement

<table>
<thead>
<tr>
<th>Chat client</th>
</tr>
</thead>
<tbody>
<tr>
<td>public class Lirc {...}</td>
</tr>
<tr>
<td>(new Lirc).createChannel(channelName);</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Program using client</th>
</tr>
</thead>
<tbody>
<tr>
<td>public class EnforcementMetaObject extends MetaObject {</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>public void beforeExecuteMethod(...) {</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>sm.checkPermission(new ChannelPermission(</td>
</tr>
<tr>
<td>joinChannel, “join”);</td>
</tr>
<tr>
<td>}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Metaobject</th>
</tr>
</thead>
</table>
| <intercept><execute><cname>Lirc</cname>
| <method>createChannel</method></execute></intercept> |

<table>
<thead>
<tr>
<th>Grant signed by WhiteHat</th>
</tr>
</thead>
<tbody>
<tr>
<td>grant signedby WhiteHat {</td>
</tr>
<tr>
<td>permission ChannelPermission allowedChannel, join;</td>
</tr>
<tr>
<td>}</td>
</tr>
</tbody>
</table>

© Victoria University of Wellington, 2006
Reflective enforcement

```java
public class Lirc {...};

(new Lirc).createChannel(channelName);

public class EnforcementMetaObject extends MetaObject {
    ...
    public void beforeExecuteMethod(...) {
        ...
        sm.checkPermission(new ChannelPermission(
            joinChannel, "join");
    }
}

<intercept><execute><cname>Lirc</cname>
    <method>createChannel</method></execute></intercept>

grant signedby WhiteHat {
    permission ChannelPermission allowedChannel, join;
}
```

**Chat client**

**Program using client**

**Metaobject**

**Binding specification**
## Reflective enforcement

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>public class Lirc {...};</code></td>
<td>Chat client</td>
</tr>
<tr>
<td><code>(new Lirc).createChannel(channelName);</code></td>
<td>Program using client</td>
</tr>
<tr>
<td><code>public class EnforcementMetaObject extends MetaObject {</code></td>
<td></td>
</tr>
<tr>
<td><code>...</code></td>
<td></td>
</tr>
<tr>
<td><code>public void beforeExecuteMethod(...) {</code></td>
<td></td>
</tr>
<tr>
<td><code>...</code></td>
<td></td>
</tr>
<tr>
<td><code>sm.checkPermission(new ChannelPermission(</code></td>
<td></td>
</tr>
<tr>
<td><code>joinChannel, “join”);</code></td>
<td></td>
</tr>
<tr>
<td><code>}</code></td>
<td></td>
</tr>
<tr>
<td><code>&lt;intercept&gt;&lt;execute&gt;&lt;cname&gt;Lirc&lt;/cname&gt;</code></td>
<td>Binding specification</td>
</tr>
<tr>
<td><code>&lt;method&gt;createChannel&lt;/method&gt;&lt;/execute&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>grant signedby WhiteHat {</code></td>
<td>Java security policy</td>
</tr>
<tr>
<td><code>permission ChannelPermission allowedChannel, join;</code></td>
<td></td>
</tr>
<tr>
<td><code>}</code></td>
<td></td>
</tr>
</tbody>
</table>
### Interdependencies

```java
public class Lirc {...};
```

```java
(new Lirc).createChannel(channelName);
```

```java
public class EnforcementMetaObject extends MetaObject {
    ...
    public void beforeExecuteMethod(...) {
        ...
        sm.checkPermission(new ChannelPermission(
            joinChannel, "join");
    }
}
```

```xml
<intercept><execute><cname>Lirc</cname><method>createChannel</method></execute></intercept>
```

```java
grant signedby WhiteHat {
    permission ChannelPermission allowedChannel, join;
}
```

---

**Chat client**

**Program using client**

**Choice of permissions related to abstract resource**
### Interdependencies

```java
public class Lirc {...};
(new Lirc).createChannel(channelName);

public class MetaObject {
    ...
    public void beforeExecuteMethod(...) {
        ...
        executePermission(new ChannelPermission(
            subject, Join``
        ));
    }
}

<intercept><execute><cname>Lirc</cname>
<method>createChannel</method></execute></intercept>

grant signedby WhiteHat {
    permission ChannelPermission allowedChannel, join;
}
```

**Chat client**

**Program using client**

**Choice of metaobject related to abstract resource that is subject of policy**
### Interdependencies

**Chat client**

```java
public class Lirc {
...

(new Lirc).createChannel(channelName);
```

**Program using client**

```java
public class EnforcementMetaObject extends MetaObject {
...

    public void beforeExecuteMethod(...) {
...

        sm.checkPermission(new ChannelPermission(
            joinChannel, "join");
    }
}
```

**Binding specification determined by resource implementation detail and policy being enforced**

```xml
<intercept>
    <execute>
        <method>createChannel</method>
    </execute>
</intercept>
```

```xml
grant signedby Whitelat {
    permission
}
```
Interdependencies

grant signed by WhiteHat {
    permission

}<intercept><execute><cname>Lirc</cname><method>createChannel</method></execute></intercept>

Program using client

public class Lirc {
    
    (new Lirc).createChannel(channelName);

    public class EnforcementMetaObject extends MetaObject {
        ...
        public void beforeExecuteMethod(...) {
            ...
            sm.checkPermission(new joinChannel, “join”);
        }
    }

Chat client

Permission checked should be related to positive authorisation
Model

Abstract Policy

Transformer

Java Policy

Kava Binding Spec

Kava Meta objects
Ponder policy language

Designed to specify wide range of policies. Declarative rather than imperative. Positive, negative and constraint based policies. Envisaged for enterprise-level enforcement so used LDAP to store policies and perform mappings.

```
inst auth+ rpc_chat {
    subject /staff/securityAdmin;
    target <Channel> /channels/support;
    action join, speak;
}
```

One policy file!
Subject

Ponder objects -> Java protection domains

Targets and actions

Ponder target is an object, actions relate to interface
Java permissions relate to set of classes
Java targets and actions optional
Don’t want to hardcode mappings to avoid rewriting transformer
Solution

- LDAP
  - subject mapping
  - PK Transformer
    - Ponder Policy
      - target related mapping
        - Resource Description
          - Kava Meta objects come with resource
          - Java Policy
          - Kava Binding Spec
Resource description

Defines vocabulary for Ponder policy
Mappings for application and system resources.
Mapping rules:
  subject-domain $\Rightarrow$ protection domain
  target-type $\Rightarrow$ Java permission
  target-domain & actions $\Rightarrow$ permission parameters
  target-type $\Rightarrow$ binding specification/choice of metaobjects
Results

Benefits

Relationships exposed and localised in one place (almost).
Policies expressed using high-level language-neutral application abstractions.

Issues

Metaobject generation
What happens with overlapping Ponder policies?
How to implement constraints? negative authorisations?
Is it really easier to use? DSL question.