pam_xacml

The eXtensible Access Control Markup Language (XACML) allows for generic access control policies in XML format. pam_xacml provides XACML support for many existing PAM enabled applications (no changes to the code required).

Contact Information:
Tobias Heide <heide at informatik.uni-tuebingen.de>
Andreas Klenk <klenk at informatik.uni-tuebingen.de>

pam_xacml includes work from **Joseph Bester**[1] within it's distribution.

Aims of pam_xacml **1**
Installation **1**
Installation on Mac OS X 1
Configuration **2**
General options 2
Option requestBuilder 2
Option requestTemplate 2
Legacy parameters 2
Option pdpRequester 3
Arguments to pdpRequester 3
Parameter obligationsScript 3
First Steps **3**
Sample Configuration Files 3
Simple PDP 3
Joseph Besters PDP 3
Appendix **4**
Sample request template 4
A sample obligationsScript 4

1. Aims of pam_xacml

pam_xacml shall ease the integration of XACML to existing applications. It can operate in two modes, the first one allowing the use of XACML Policy Decision Points without changes to the application (the so-called legacy-mode) and the second mode requiring changes to the application, but allowing more powerful authorization.

pam_xacml is based on the Pluggable Authentication Modules that are standardized by the Open Group[2]. In legacy mode, pam_xacml is fully compliant to current PAM implementations. When using the extended mode, custom message types are passed over the PAM conversation mechanism, which have to be understood by the application.

2. Installation

You will need development versions of the following external libraries to build pam_xacml: PAM, gSOAP, libxml2, pkg-config, patch

For GSOAP you will need: bison/yacc, flex, libssl/openssl

We rely on the standard GNU autotools to build pam_xacml. You should install gSOAP version 2.7.10 before starting the installation of pam_xacml. You can get it from [3]. Afterwards compile gSOAP typing the following commands:

```
./configure --prefix=/usr
make
sudo make install
```

It is important to install gSOAP into a location where it can be found by the configure script of pam_xacml. When done installing gSOAP you can progress with the installation of pam_xacml. Normally you can skip running the autotools when using the distribution, so only issue the following five commands if you run into trouble.

```
aclocal
libtoolize
autoheader
autoconf
automake -a -c --foreign
```

Afterwards just follow the standard GNU autotools installation steps:

```
./configure --prefix=/usr
make
sudo make install
```

Installation on Mac OS X

When compiling on Mac OS X platform, you will need to take a different build process. First, install fink some other GNU environment.[4] You will need to install several libraries:

```
fink install pkgconfig
fink install libxml2 libxml2-bin
fink install libxml2-shlibs
export PKG_CONFIG_PATH=/usr/lib/pkgconfig/
```

The process of initializing the autotools environment also differs a bit from above:

```
aclocal
glibtoolize
autoheader
autoconf
```

[1] Joseph Besters original work can be found at http://www.mcs.anl.gov/~bester/xacml/
[2] see X/Open Single Sign-on Service (XSSO) - Pluggable Authentication Modules, http://www.opengroup.org/onlinepubs/8329799/
[3] gSOAP download page http://sourceforge.net/project/showfiles.php?group_id=52781
[4] Fink can be obtained from http://www.finkproject.org/

```
automake -a -c --foreign
```

(Note the use of glibtoolize instead of libtoolize). Then issue the following commands to compile pam_xacml:

```
./configure --prefix=/usr \
    PAM_DIR=/usr/lib/pam
make
sudo make install
```

You need to provide the PAM_DIR variable to configure, because Mac OS uses a non-standard location for the PAM libraries. After compilation the installation behaves like a installation on a „normal“ UNIX environment.

3.Configuration

To add pam_xacml to an authentication chain of an application, you need to edit the corresponding file under /etc/pam.d. For example, if you would like to add pam_xacml to sudo, you will have to add the following line to /etc/pam.d/sudo:

```
account sufficient pam_xacml.so <param>
```

<param> is a list of configuration options that is explained in the following sections.

General options

Name	Opt	Description
debug	x	If present, pam_xacml will generate verbose syslog messages.
dumpXacml	x	If present, pam_xacml will dump the XACML request and reponse to syslog.

Option requestBuilder

This option provides information to pam_xacml, how the request that is sent to the PDP shall be generated. Several values are possible:

Value	Description	Further parameters
INTERNAL	Fills out a XML template with several values	requestTemplate
APPLICATION	The application is XACML capable and will provide a complete request	mandatoryObligations
SCRIPT	An external script will generate the request.	requestBuilderScript

Option requestTemplate

This option is used in conjunction with the INTERNAL request builder. As a value, you shall provide the complete path to an XML template which will be filled out by pam_xacml. You may use the following fields in your template:

Field	Description
Resource	Konstanter String "TestRessource"
Username	Username of the currently logged on user
Action	Konstanter String "TestAction"
Hostname	The hostname of the host executing PamXacml.
GUID	A random string that is also passed to the obligationsScript as first parameter. This can be used in helper scripts to identify the response that was given by a PDP to a request, because you can insert <pamxacml:GUID /> as an argument to a <pamxacml:INVOKE /> command. This value is also passed to the obligationsScript as first argument, if specified in the PAM configuration file.

The value of a field can be requested by writing <pamxacml:Fieldname /> inside the XML template. Another powerful capability of the template mechanism is to call external scripts. If you use a line like

```
<pamxacml:INVOKE>/path/to/Script argument1
argument2 <pamxacml:Fieldname /></
pamxacml:INVOKE>
```

The script /path/to/script will be invoked and will be given the parameters you specified. You can even provide pam_xacmls fields as arguments like Username or GUID. You can see a complete example in the appendix.

Legacy parameters

The following options control the behavior of pam_xacml in regards of compatibility to existing applications:

Name	Opt	Description
obligationsNotMandatory	X	If present, pam_xacml will not fail if the application does not understand the obligations.

Name	Opt	Description
obligationsIgnore	X	If present, pam_xacml will not send any obligations to the application (for legacy applications). This can only be set together with obligationsNotMandatory.

Option pdpRequester

This option controls what PDP will be asked for the decision. Several PDPs exist, some only useful for deb ugging purposes. To implement your own PDP requesters, have a look at the appropriate directory in the source code (src/core/...).

Value	Description	Further parameters
FILE	Will always return this XACML response, independent of the request.	staticResponse
SIMPLE	For a simple TCP PDP that accepts plain XACML Requests and sends pure XACML Response. (Without SOAP, SAML, Security!)	pdpAddress
SOAP	For a PDP that accepts SOAP encapsulated XACML Requests and XACML Response.	pdpEndpoint
HANNES	For the PDP written by Hannes Angst.	pdpEndpoint

Arguments to pdpRequester

Name	Description
staticResponse	Path to a file containing a valid XACML response.
pdpAddress	This parameter contains the ip address and port number of a TCP service. Look for example at the SimplePDP in the PDP subdirectory of the pam_xacml distribution. The value of this parameter should look like 192.168.0.119:1234
pdpEndpoint	The value of this parameter is an URL to a webservice which answers XACML requests. For example, use http://localhost:8080/webservice

Parameter obligationsScript

This parameter needs as value the complete path to a script which receives the XACML response from the PDP on STDIN and exits with 0 if it understood the obligations and was able to enforce them or any other return code if it could not ensure the obligations. Additionally, the script is passed the GUID as first command line argument which is also passed to the INTERNAL and the SCRIPT engine request generators. A sample script can be seen in the appendix.

4.First Steps

Sample Configuration Files

You can find some sample configurations files in the dist/ directory of the distribution. Configuration files for the included test program are contained within the dist/pam.d directory and should be copied to /etc/pam.d. The dist/templates directory contains a request templated that can be used with the internal request builder.

Simple PDP

We included a simple, Java-based Policy Decision Point that is based upon the Sun XACML framework. It can be found in the PDP/SimplePDP subdirectory of the distribution. To build it, you will need to get several libraries from the internet:

- JAXP_14_FCS.jar
- sunxacml.jar
- xercesImpl.jar

Place them into the subdirectories according to the README in the PDP/SimplePDP directory.

Afterwards, you will be able to run the build and the execute script. You can test your installation by running makeRequest.sh.

SimplePDP listens on Port 1234 by default and receives raw XACML requests over a TCP/IP connection and responses in a raw format.

Joseph Besters PDP

Joseph Besters PDP is build automatically when you build pamxacml. It is contained in the subdirectory src/ext-lib/xacml-1.0. The server is named „xacml-server“. When you start it, you will be able to use it with pdpRequester=BESTER. Please specifiy http://localhost:8080/wsrt/services/XACML as pdpEndpoint (or modify it appropiatly).

5.Appendix

Sample request template

```
<?xml version="1.0" encoding="UTF-8"?>
<Request xmlns="urn:oasis:names:tc:xacml:
2.0:context:schema:os" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:oasis:names:tc:xacm
l:2.0:context:schema:os http://docs.oasis-
open.org/xacml/access_control-xacml-2.0-
context-schema-os.xsd">
   <Subject>
    <Attribute
AttributeId="urn:oasis:names:tc:xacml:
1.0:subject:subject-id" DataType="http://
www.w3.org/2001/XMLSchema#string">
      <AttributeValue><pamxacml:Username /
></AttributeValue>
    </Attribute>
  </Subject>
  <Resource>
     <Attribute
AttributeId="urn:oasis:names:tc:xacml:
1.0:resource:resource-id" DataType="http://
www.w3.org/2001/
XMLSchema#string"><AttributeValue><pamxacml
:Resource /></AttributeValue></Attribute>
  </Resource>
  <Action>
     <Attribute
AttributeId="urn:oasis:names:tc:xacml:
1.0:action:action-id" DataType="http://
www.w3.org/2001/
XMLSchema#string"><AttributeValue>TestActio
n</AttributeValue></Attribute>
  </Action>
  <Environment>
     <Attribute
AttributeId="urn:pamxacml:hostname"
DataType="http://www.w3.org/2001/
XMLSchema#string"><AttributeValue><pamxacml
:Hostname /></AttributeValue></Attribute>
     <Attribute
AttributeId="urn:pamxacml:extended"
DataType="http://www.w3.org/2001/
XMLSchema#string"><AttributeValue><pamxacml
:INVOKE>/Users/theide/Documents/workspace/
pamxacml/tests/argument-returner.sh Arg1
<pamxacml:Username /> Arg2 Arg3</
pamxacml:INVOKE></AttributeValue></
Attribute>
     <Attribute
AttributeId="urn:pamxacml:guid"
DataType="http://www.w3.org/2001/
XMLSchema#string"><AttributeValue><pamxacml
:GUID /></AttributeValue></Attribute>
  </Environment>
</Request>
```

A sample obligationsScript

Please pay attention to the line breaks when testing this example.

```
#!/bin/bash

x=`cat /dev/stdin`
date=`date`
echo "========================" >>/tmp/
     obligations-receiver
echo "Unique ID: $1" >>/tmp/
     obligations-receiver
echo "Received on $date" >>/tmp/
     obligations-receiver
echo $x >> /tmp/obligations-receiver

#exit -2
exit 0
```