Published on *Gooze* (http://www.gooze.eu)

GNU|Linux smartcard logon using PAM-PKCS11

This guide describes how to logon a GNU/Linux host using PAM_PKCS11.

Audience

This tutorial is suited for users who would like to **secure access** to several workstations using smartcards and X.509 certificates. As PAM_PKCS11 is able to verify X.509 certificates and Certificate Revocation List (CRL) it is well suited for organizations.

This tutorial does not cover the case of an LDAP mapping, which will be a future tutorial.

Prerequisites

As a prerequisite, you should read our smart card quickstarter guide [1], in order to learn how to install and configure smartcards.

Hereafter, we consider that you installed a smart card reader and configured a smart card either with a self-signed certificate or a free X.509 certificate like offered by CAcert.org community. Make sure to backup your certificates and keys as explained previously, because you will not be able to extract private keys from your smart card.

PAM and PAM-PKCS11 features

GNU/Linux uses PAM (Pluggable Authentication Modules) to authenticate using a variety of methods.
PAM is installed on every workstation. PAM documentation can be read in details: The Linux-PAM System Administrators' Guide [2].

PAM_PKCS11 is an OpenSC [3] project designed for authentication using smartcards and X.509 certificates. You can visit OpenSC Pam-PKCS11 page for information: http://www.opensc-project.org/pam_pkcs11/ [4]

Pam-PKCS11 offers the following features:

- Verification of X.509 certificates against locally stored certificates.
- Verification of X.509 certificates against Certification Authorities.
- Certificate Revocation List (CRL).
- Verification of X.509 certificates against Certification Authorities.
- Automatic and custom Mapping rules from X.509 certificates to users.
- Tools to handle screen saver when the card is removed/inserted.
- Tools to inspect the content of certificates.

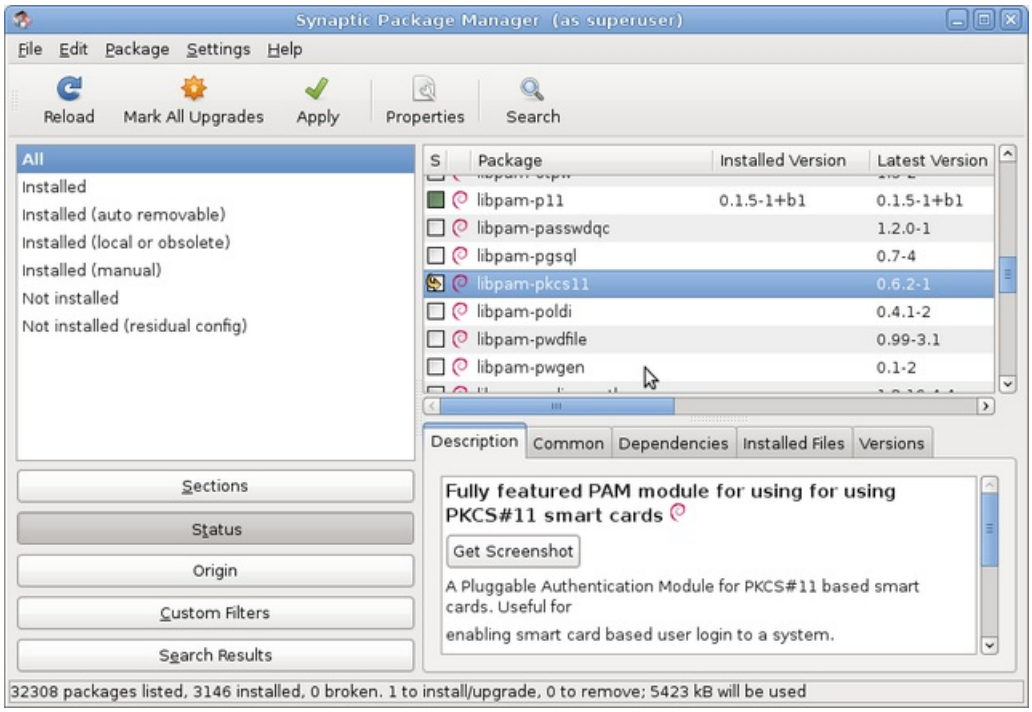
In a production environment, PAM-PKCS11 should be preferred over PAM-P11 as it offers more features, including certificate verification and revocation. For users who need to manage simple access, read our tutorial GNU/Linux smart card logon using PAM-P11 [5].

Installation using binary packages

Under Debian based / Ubuntu, install libpam-pkcs11 package:

```
$ apt-get install libpam-pkcs11
```

Alternatively, use a graphical installer like Synaptic:



Installation from sources

Use SVN to fetch the latest trunk:

```
$svn co http://www.opensc-project.org/svn/pam_pkcs11/trunk [6] pam_pkcs11
```

Compile and install:

```
$ tar xvzf pkcs11_login-X.Y.Z.tar.gz
$ cd pkcs11_login-X.Y.Z
$ ./configure
$ make
$ sudo make install
```

Configuring Pam_PKCS11

PAM configuration files are stored in the /etc/pam.d/ directory.

PAM common files

Let us have a look at the common-auth configuration file:

```
$ cat /etc/pam.d/common-auth
```

This displays:

```
$ # here are the per-package modules (the "Primary" block)
auth [success=1 default=ignore] pam_unix.so nullok_secure
# here's the fallback if no module succeeds
auth requisite pam_deny.so
# prime the stack with a positive return value if there isn't one already;
# this avoids us returning an error just because nothing sets a success code
# since the modules above will each just jump around
auth required pam_permit.so
# end of pam-auth-update config
```

As of pam 1.0.1-6, this file is managed by pam-auth-update by default.

To take advantage of this, it is recommended that you configure any local modules either before or after the default block, and use pam-auth-update to manage selection of other modules.

pam-config mechanism stores templates in /usr/share/pam-configs.

Let us explore this directory:

```
$ ls /usr/share/pam-configs
consolekit gnome-keyring unix
```

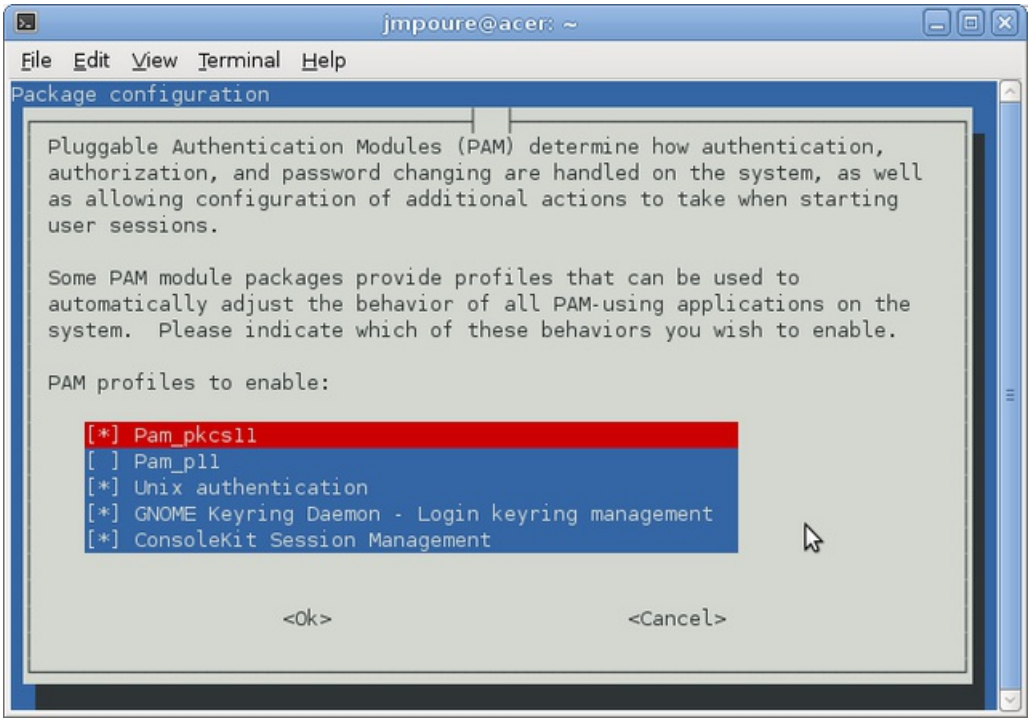
Now we simply create a template for pam_p11 login.
Create an empty file /usr/share/pam-configs/pkcs11 and add:

```
Name: Pam_pkcs11
Default: yes
Priority: 800
Auth-Type: Primary
Auth: sufficient pam_pkcs11.so config_file=/etc/pam_pkcs11/pam_pkcs11.conf
```

To regenerate PAM configuration files, we need to execute:

```
$ pam-auth-update
```

A Debian configuration dialog is displayed:



Make sure 'Unix authentication' is enabled, otherwise there is a risk to lose the ability to connect using passwords.

Enable 'libpam-pkcs11' and Disable 'libpam-p11' to avoid a separate access system using smart cards.

Click Okay.

Let us have a look at the common-session configuration file:

```
$ cat /etc/pam.d/common-auth
```

```
# here are the per-package modules (the "Primary" block)
auth sufficient pam_pkcs11.so config_file=/etc/pam_pkcs11/pam_pkcs11.conf
auth [success=1 default=ignore] pam_unix.so nullok_secure try_first_pass
# here's the fallback if no module succeeds
auth requisite pam_deny.so
# prime the stack with a positive return value if there isn't one already;
# this avoids us returning an error just because nothing sets a success code
# since the modules above will each just jump around
auth required pam_permit.so
# and here are more per-package modules (the "Additional" block)
# end of pam-auth-update config
```

Again, make sure that this line allow connecting using passwords:

```
auth [success=1 default=ignore] pam_unix.so nullok_secure try_first_pass
```

Notice the line:

```
auth sufficient pam_pkcs11.so /etc/pam_pkcs11/pam_pkcs11.conf
```

PAM PKCS11 configuration file

We also need to install the /etc/pam_pkcs11.conf file, which is not installed by default:

Create a pam-pkcs11 folder:

```
$ sudo mkdir /etc/pam_pkcs11
```

Create an empty pam_pkcs11.conf file using the pam_pkcs11 default configuration file. This file is not installed by defaut, so we need to install it.

As root:

```
$ cp /usr/share/doc/libpam-pkcs11/examples/pam_pkcs11.conf.example.gz /etc/pam_pkcs11/
$ cd /etc/pam_pkcs11/
$ gunzip pam_pkcs11.conf.example.gz
$ mv pam_pkcs11.conf.example pam_pkcs11.conf
```

For information, you can explore content of pam_pkcs11.conf:

```
$ cat /etc/pam_pkcs11/pam_pkcs11.conf
```

This displays:

```
#
# Configuration file for pam_pkcs11 module
#
# Version 0.4
# Author: Juan Antonio Martinez
#
pam_pkcs11 {
# Allow empty passwords
nullok = true;

# Enable debugging support.
debug = true;

# Do not prompt the user for the passwords but take them from the
# PAM_ items instead.
use_first_pass = false;

# Do not prompt the user for the passwords unless PAM_(OLD)AUTHTOK
# is unset.
try_first_pass = false;

# Like try_first_pass, but fail if the new PAM_AUTHTOK has not been
# previously set (intended for stacking password modules only).
use_authtok = false;

# Filename of the PKCS #11 module. The default value is "default"
use_pkcs11_module = opensc;

pkcs11_module opensc {
module = /usr/lib/opensc-pkcs11.so;
description = "OpenSC PKCS#11 module";

# Which slot to use?
# You can use "slot_num" or "slot_description", but not both, to specify
# the slot to use. Using "slot_description" is preferred because the
```

```
# PKCS#11 specification does not guarantee slot ordering. "slot_num" should
# only be used with those PKCS#11 implementations that guarantee
# constant slot numbering.
#
# slot_description = "xxxx"
# The slot is specified by the slot description, for example,
# slot_description = "Sun Crypto Softtoken". The default value is
# "none" which means to use the first slot with an available token.
#
# slot_num = a_number
# The slot is specified by the slot number, for example, slot_num = 1.
# The default value is zero which means to use the first slot with an
# available token.
#
slot_description = "none";

# Where are CA certificates stored?
# You can setup this value to:
# 1- A directory with openssl hash-links to all certificates
# 2- A CA file in PEM (.pem) or ASN1 (.cer) format,
# containing all allowed CA certs
# The default value is /etc/pam_pkcs11/cacerts.
ca_dir = /etc/pam_pkcs11/cacerts;

# Path to the directory where the local (offline) CRLs are stored.
# Same convention as above is applied: you can choose either
# hash-link directory or CRL file
# The default value is /etc/pam_pkcs11/crls.
crl_dir = /etc/pam_pkcs11/crls;

# Some pcks#11 libraries can handle multithreading. So
# set it to true to properly call C_Initialize()
support_threads = false;

# Sets the Certificate verification policy.
# "none" Performs no verification
# "ca" Does CA check
# "crl_online" Downloads the CRL form the location given by the
# CRL distribution point extension of the certificate
# "crl_offline" Uses the locally stored CRLs
# "crl_auto" Is a combination of online and offline; it first
# tries to download the CRL from a possibly given CRL
# distribution point and if this fails, uses the local
# CRLs
# "signature" Does also a signature check to ensure that private
# and public key matches
# You can use a combination of ca,crl, and signature flags, or just
# use "none".
cert_policy = ca,signature;

# What kind of token?
# The value of the token_type parameter will be used in the user prompt
# messages. The default value is "Smart card".
token_type = "Smart card";
}

# Aladdin eTokenPRO 32
```

```
pkcs11_module etoken {
module = /usr/local/lib/libetpkcs11.so
description = "Aladdin eTokenPRO-32";
slot_num = 0;
support_threads = true;
ca_dir = /etc/pam_pkcs11/cacerts;
crl_dir = /etc/pam_pkcs11/crls;
cert_policy = ca,signature;
}

# NSS (Network Security Service) config
pkcs11_module nss {
nss_dir = /etc/ssl/nssdb;
crl_policy = none;
}

# Default pkcs11 module
pkcs11_module default {
module = /usr/lib/pam_pkcs11/pkcs11_module.so;
description = "Default pkcs#11 module";
slot_num = 0;
support_threads = false;
ca_dir = /etc/pam_pkcs11/cacerts;
crl_dir = /etc/pam_pkcs11/crls;
cert_policy = none;
}

# Which mappers ( Cert to login ) to use?
# you can use several mappers:
#
# subject - Cert Subject to login file based mapper
# pwent - CN to getpwent() login or gecos fields mapper
# ldap - LDAP mapper
# opensc - Search certificate in ${HOME}/.eid/authorized_certificates
# openssh - Search certificate public key in ${HOME}/.ssh/authorized_keys
# mail - Compare email fields from certificate
# ms - Use Microsoft Universal Principal Name extension
# krb - Compare againts Kerberos Principal Name
# cn - Compare Common Name (CN)
# uid - Compare Unique Identifier
# digest - Certificate digest to login (mapfile based) mapper
# generic - User defined certificate contents mapped
# null - blind access/deny mapper
#
# You can select a comma-separated mapper list.
# If used null mapper should be the last in the list :-)
# Also you should select at least one mapper, otherwise
# certificate will not match :-)
use_mappers = digest, cn, pwent, uid, mail, subject, null;

# When no absolute path or module info is provided, use this
# value as module search path
# TODO:
# This is not still functional: use absolute pathnames or LD_LIBRARY_PATH
mapper_search_path = /usr/lib/pam_pkcs11;

#
```

```
# Generic certificate contents mapper
mapper generic {
debug = true;
#module = /usr/lib/pam_pkcs11/generic_mapper.so;
module = internal;
# ignore letter case on match/compare
ignorecase = false;
# Use one of "cn" , "subject" , "kpn" , "email" , "upn" or "uid"
cert_item = cn;
# Define mapfile if needed, else select "none"
mapfile = file:///etc/pam_pkcs11/generic_mapping; [7]
# Decide if use getpwent() to map login
use_getpwent = false;
}

# Certificate Subject to login based mapper
# provided file stores one or more "Subject -> login" lines
mapper subject {
debug = false;
# module = /usr/lib/pam_pkcs11/subject_mapper.so;
module = internal;
ignorecase = false;
mapfile = file:///etc/pam_pkcs11/subject_mapping; [8]
}

# Search public keys from $HOME/.ssh/authorized_keys to match users
mapper openssh {
debug = false;
module = /usr/lib/pam_pkcs11/openssh_mapper.so;
}

# Search certificates from $HOME/.eid/authorized_certificates to match users
mapper opensc {
debug = false;
module = /usr/lib/pam_pkcs11/opensc_mapper.so;
}

# Certificate Common Name ( CN ) to getpwent() mapper
mapper pwent {
debug = false;
ignorecase = false;
module = internal;
# module = /usr/lib/pam_pkcs11/pwent_mapper.so;
}

# Null ( no map ) mapper. when user as finder matchs to NULL or "nobody"
mapper null {
debug = false;
# module = /usr/lib/pam_pkcs11/null_mapper.so;
module = internal ;
# select behavior: always match, or always fail
default_match = false;
# on match, select returned user
default_user = nobody ;
}

# Directory ( ldap style ) mapper
```

```
mapper ldap {
debug = false;
module = /usr/lib/pam_pkcs11/ldap_mapper.so;
# hostname of ldap server (use LDAP-URI for more then one)
ldaphost = "";
# Port on ldap server to connect, this is also the default
# if no port is given in URI below
# if empty, then 389 for TLS and 636 for SSL is used
ldapport = ;
# space separted list of LDAP URIs (URIs are used by given order)
URI = "";
# Scope of search: 0-2
# Default is 1 = "one", meaning the set of records one
# level below the basedn.
# 0 = "base" means search only the basedn, and
# 2 = "sub" means the union of entries at the "base" level
# and ? all or "one" level below ??? FIXME
scope = 2;
# DN to bind with. Must have read-access for user entries
# under "base"
binddn = "cn=pam,o=example,c=com";
# Password for above DN
passwd = "";
# Searchbase for user entries
base = "ou=People,o=example,c=com";
# Attribute of user entry which contains the certificate
attribute = "userCertificate";
# Searchfilter for user entry. Must only let pass user entry
# for the login user.
filter = "(&(objectClass=posixAccount)(uid=%s))"
# SSL/TLS-Switch
# This is a global switch, you can't switch between
# SSL or TLS and non secured connections per URI!
# values: off (standard), tls or on (ssl) or ssl
ssl = tls
# SSL specific settings
# tls_randfile = ...
tls_cacertfile = /etc/ssl/cacert.pem
# tls_cacertdir = ...
tls_checkpeer = 0
#tls_ciphers = ...
#tls_cert = ...
#tls_key = ...
}

# Assume common name (CN) to be the login
mapper cn {
debug = false;
module = internal;
# module = /usr/lib/pam_pkcs11/cn_mapper.so;
ignorecase = true;
# mapfile = file:///etc/pam_pkcs11/cn_map: [9]
mapfile = "none";
}

# mail - Compare email field from certificate
```

```
mapper mail {
debug = false;
module = internal;
# module = /usr/lib/pam_pkcs11/mail_mapper.so;
# Declare mapfile or
# leave empty "" or "none" to use no map
mapfile = file:///etc/pam_pkcs11/mail_mapping; [10]
# Some certs store email in uppercase. take care on this
ignorecase = true;
# Also check that host matches mx domain
# when using mapfile this feature is ignored
ignoredomain = false;
}

# ms - Use Microsoft Universal Principal Name extension
# UPN is in format login@ADS_Domain. No map is needed, just
# check domain name.
mapper ms {
debug = false;
module = internal;
# module = /usr/lib/pam_pkcs11/ms_mapper.so;
ignorecase = false;
ignoredomain = false;
domain = "domain.com";
}

# krb - Compare againts Kerberos Principal Name
mapper krb {
debug = false;
module = internal;
# module = /usr/lib/pam_pkcs11/krb_mapper.so;
ignorecase = false;
mapfile = "none";
}

# uid - Maps Subject Unique Identifier field (if exist) to login
mapper uid {
debug = false;
module = internal;
# module = /usr/lib/pam_pkcs11/uid_mapper.so;
ignorecase = false;
mapfile = "none";
}

# digest - elaborate certificate digest and map it into a file
mapper digest {
debug = false;
module = internal;
# module = /usr/lib/pam_pkcs11/digest_mapper.so;
# algorithm used to evaluate certificate digest
# Select one of:
# "null","md2","md4","md5","sha","sha1","dss","dss1","ripemd160"
algorithm = "sha1";
mapfile = file:///etc/pam_pkcs11/digest_mapping; [11]
# mapfile = "none";
}
```

```
}
```

We will use this configuration in the next section.

X.509 certificate verification

Verifying the Certificate Authority (CA) is a preliminary operation before authentication, with one limitation:

- Only local CAs can be verified.
- Online CAs like CAcert.org or StartSSL cannot be verified.

Why is CA verification limited?

According to OpenSC pam-pkcs11 documentation [1], there is a limitation in OpenSSL preventing online CA validation.
But we doubt that OpenSSL is limited. We think there may be a bug in pam_pkcs11.

Therefore, we are considering two scenarios:

Scenario 1: local CA

A local CA is managed localy on your computer.

pam-pkcs11 needs a list of authorized certificate authorities (CAs) and a Certificate Revocation List (CRL).

Notice these lines in /etc/pam_pkcs11/pam_pkcs11.conf:

```
# Where are CA certificates stored?
# You can setup this value to:
# 1- A directory with openssl hash-links to all certificates
# 2- A CA file in PEM (.pem) or ASN1 (.cer) format,
# containing all allowed CA certs
# The default value is /etc/pam_pkcs11/cacerts.
ca_dir = /etc/pam_pkcs11/cacerts;
```

Create the needed folder:

```
$ mkdir /etc/pam_pkcs11/cacerts;
```

Copy CA certificates in /etc/pam_pkcs11/cacerts in PEM format.

Create hash links using OpenSC pkcs11_make_hash_link utility:

```
$ pkcs11_make_hash_link /etc/pam_pkcs11/cacerts
```

As for CRL, notice these lines in /etc/pam_pkcs11/pam_pkcs11.conf:

```
# Path to the directory where the local (offline) CRLs are stored.
# Same convention as above is applied: you can choose either
# hash-link directory or CRL file
# The default value is /etc/pam_pkcs11/crls.
crl_dir = /etc/pam_pkcs11/crls;
```

Create the needed folder:

```
$ mkdir /etc/pam_pkcs11/crls;
```

Copy your CRL file in /etc/pam_pkcs11/crls.

Finaly, set policy to:

```
cert_policy = ca,signature,crl_auto;
```

Scenario 2: online CA

An online CA is a certification Authority like CAcert.org.

In /etc/pam_pkcs11/pam_pkcs11.conf, set:

```
cert_policy = signature;
```

We are not very sure of what "signature" means, but it proved to work in offline mode.

Avoid authentication on simple values like "email", "subject" or any value that could be easily forked to create false smartcards. When using online CAs without validation, only use public keys to map users.

[1] http://www.opensc-project.org/doc/pam_pkcs11/pam_pkcs11.html#configfile [12]

X.509 user mapping

In a second phase, certificates are mapped to user accounts according to rules configured in /etc/pam_pkcs11/pam_pkcs11.conf

```
# Which mappers ( Cert to login ) to use?
# you can use several mappers:
#
# subject - Cert Subject to login file based mapper
# pwent - CN to getpwent() login or gecos fields mapper
# ldap - LDAP mapper
# opensc - Search certificate in ${HOME}/.eid/authorized_certificates
# openssh - Search certificate public key in ${HOME}/.ssh/authorized_keys
# mail - Compare email fields from certificate
# ms - Use Microsoft Universal Principal Name extension
# krb - Compare againts Kerberos Principal Name
# cn - Compare Common Name (CN)

# uid - Compare Unique Identifier
# digest - Certificate digest to login (mapfile based) mapper
# generic - User defined certificate contents mapped
# null - blind access/deny mapper
#
# You can select a comma-separated mapper list.
# If used null mapper should be the last in the list :-)
# Also you should select at least one mapper, otherwise
# certificate will not match :-)
use_mappers = digest, cn, pwent, uid, mail, subject, null;
```

use_mappers can have several values.

The most common mappers are: mail, subject, opensc and openssh.
ldap is not described here and will be described in another tutorial.

Email user mapper

Mail mapping is only secure when the CA was verified.
It is also a convenient way to test pam_pkcs11 settings.

Define the mapper:

use_mappers = mail;

Create /etc/pam_pkcs11/mail_mapping

mapping file for Certificate E-email
format: email -> login

foo@bar.com [13] -> foo

Adapt to your user and email.

Subject user mapper

Define the mapper:

use_mappers = subject;

Create /etc/pam_pkcs11/mail_mapping

Mapping file for Certificate Subject
format: Certificate Subject -> login
#
/C=ES/O=FNMT/OU=FNMT Clase 2 CA/OU=500051483/CN=NOMBRE MARTINEZ CASTA\xF1O JUAN ANTONIO - NIF 50431138G -> foo

Adapt to your subject and user.

OpenSC user mapper

Notice these lines in /etc/pam_pkcs11/pam_pkcs11.conf:

```
# Search certificates from $HOME/.eid/authorized_certificates to match users
mapper opensc {
debug = false;
module = /usr/lib/pam_pkcs11/opensc_mapper.so;
}
```

The module path is wrong on Debian systems, it should be:

```
module = /lib/pam_pkcs11/opensc_mapper.so;
```

Configure PAM-pkcs11 to use OpenSC mapper.

Edit /etc/pam_pkcs11/pam_pkcs11.conf

```
use_mappers = opensc, null;
```

In each user directory, create an .eid folder:

```
$ mkdir -p ~/.eid
$ chmod og= ~/.eid
```

Query X.509 certificates on your smartcard:

```
$ pkcs15-tool --list-certificates
Using reader with a card: Feitian SCR301 01 00
X.509 Certificate
Flags : 2
Authority: no
Path : 3f0050153100
ID : 7645d913d5b4e03f3fe54816ff02324c23a7ebf4
```

Extract the X.509 certificate with ID 7645d913d5b4e03f3fe54816ff02324c23a7ebf4 to /.eid/authorized_certificates:

```
$ pkcs15-tool --read-certificate 7645d913d5b4e03f3fe54816ff02324c23a7ebf4 -o ~/.eid/authorized_certificates
```

OpenSSH user mapper

Notice these lines in /etc/pam_pkcs11/pam_pkcs11.conf:

```
# Search public keys from $HOME/.ssh/authorized_keys to match users
mapper openssh {
debug = true;
module = /lib/pam_pkcs11/openssh_mapper.so;
}
```

Edit /etc/pam_pkcs11/pam_pkcs11.conf and add OpenSSH mapper:

```
use_mappers = openssh, null;
```

Query the RSA public keys on your card :

```
$ pkcs15-tool --list-public-keys
Using reader with a card: Feitian SCR301 01 00
Public RSA Key [Private Key]
Com. Flags : 2
Usage : [0x4], sign
Access Flags: [0x0]
ModLength : 2048
Key ref : 0
Native : no
Path : 3f0050153000
Auth ID :
ID : c6f280080fb0ed1ebff0480a01d00a98a1b3b89a
```

In the example, we have one public key with ID c6f280080fb0ed1ebff0480a01d00a98a1b3b89a.

Now, extract and copy the RSA public key to ~/.ssh/authorized_keys:

```
$pkcs15-tool --read-ssh-key c6f280080fb0ed1ebff0480a01d00a98a1b3b89a -o ~/.ssh/authorized_keys

Using reader with a card: Feitian SCR301 01 00
Please enter PIN [User PIN]:
```

Testing single sign-on logon (sso)

Now, you should be able to logon using your smartcard:

```
$ su foo

Please insert your Smart card or enter your username.
Found the Smart card.
Welcome François Pérou (User PIN)!
Smart card PIN: ******
DEBUG:openssh_mapper.c:387: OpenSSH mapper started. debug: 1, mapfile:
/etc/pam_pkcs11/authorized_keys
```

Gnome smartcard screen locking

For security, it may be useful to lock the screen when the card is removed.

Modify this file: /etc/pam_pkcs11/card_eventmgr.conf

```
pkcs11_eventmgr {

# Run in background? Implies debug=false if true
daemon = true;

# show debug messages?
debug = false;

# polling time in seconds
polling_time = 1;

# expire time in seconds
# default = 0 ( no expire )
expire_time = 0;

# pkcs11 module to use
pkcs11_module = /usr/lib/opensc-pkcs11.so;

#
# list of events and actions

# Card inserted
event card_insert {
# what to do if an action fail?
# ignore : continue to next action
# return : end action sequence
# quit : end program
on_error = ignore ;

# You can enter several, comma-separated action entries
# they will be executed in turn
action = "gnome-screensaver-command --poke";
```

```
}

# Card has been removed
event card_remove {
on_error = ignore;
action = "gnome-screensaver-command --lock";
}

# Too much time card removed
event expire_time {
on_error = ignore;
action = "/bin/false";
}
}
```

If you would like a more granular configuration based on users:

You may specify in /etc/pam_pkcs11/card_eventmgr.conf

```
pkcs11_eventmgr {
nodebug
nodaemon
polling_time=5
config_file=${HOME}/.pkcs11_eventmgr.conf
}
```

And move .pkcs11_eventmgr.conf in each user directory.

References

To study PAM-PKCS11 mapping, you may refer to PAM-PKCS11 User Manual [14].

Known issues

PAM PKCS11 version 0.6.7

When pam_pkcs11 authorizes it asks for all private keys from card and chooses the first one found

Issue: http://www.gooze.eu/forums/support/epass2003-pam-pkcs11 [15]
Solution: install PAM PKCS11 from source using SVN as described in our manual.

Source URL: http://www.gooze.eu/howto/gnu-linux-smartcard-logon-using-pam-pkcs11

Links:
[1] http://www.gooze.eu/howto/smart-card-quickstarter-guide
[2] http://www.kernel.org/pub/linux/libs/pam/Linux-PAM-html/Linux-PAM_SAG.html
[3] http://www.opensc-project.org
[4] http://www.opensc-project.org/pam_pkcs11/
[5] http://www.gooze.eu/howto/gnu-linux-smart-card-logon-using-pam-p11
[6] http://www.opensc-project.org/svn/pam_pkcs11/trunk

[7] http://www.gooze.eu/etc/pam_pkcs11/generic_mapping;
[8] http://www.gooze.eu/etc/pam_pkcs11/subject_mapping;
[9] http://www.gooze.eu/etc/pam_pkcs11/cn_map;
[10] http://www.gooze.eu/etc/pam_pkcs11/mail_mapping;
[11] http://www.gooze.eu/etc/pam_pkcs11/digest_mapping;
[12] http://www.opensc-project.org/doc/pam_pkcs11/pam_pkcs11.html#configfile
[13] mailto:foo@bar.com
[14] http://www.opensc-project.org/doc/pam_pkcs11/pam_pkcs11.html
[15] http://www.gooze.eu/forums/support/epass2003-pam-pkcs11