Table of Contents

eToken experience

This is a summary of my installation and configuration attempts with an Aladdin eToken.

[update 1] Aladdin has apparently being sold to SafeNet Inc, which already bought Rainbow and its iKey 3000 token. It is not clear where Aladdin drivers can be obtained at the time of writing (25th May 2009).

[update 2] Seems like the drive is now back on the Russian site: http://www.aladdin.ru/support/download/detail.php?ID=177 (19/01/2010)

This page intends to complement http://www.nikhef.nl/pub/projects/grid/gridwiki/index.php/EToken.

Advantages and drawbacks

The good: The eToken is rather cheap, does not require any modification of the server-side or license subscription to be paid. Unless the device suffers from a particular vulnerability, the credentials you store on the device cannot be copied by an attacker (but a local privileged attacker could re-use the token while it is connected).

The bad: The eToken has proprietary drivers, which are rather painful to find, and are not compatible with some open source formats. Once the token has been formatted under a particular operating system, it will be rather difficult to re-use the credentials under another operating system. And the token does not solve all security issues.

The ugly: While Firefox is very easy to configure to use the token, most Linux distributions have decided to compile their binary SSH client **without** hardware token support. Yes, this means you need to recompile your SSH client. And if you do so, it still will not give you a Kerberos ticket upon successful authentication.

Kerberos and AFS support

A Kerberos ticket is needed to obtain an AFS token. However it is not possible to obtain a Kerberos ticket with an eToken without modifiying the KDC.

In principle, modern Kerberos implementations support PKINIT, which would enable a mapping between the DN presented by a PKCS11-compatible device (like the eToken) and a Kerberos principal. The amount of work required to enable and maintain this mapping is unclear.

Installation and configuration

Aladdin is providing proprietary drivers, which are only available upon request. The Nikhef Wiki page has a lot of information on the driver and its support under Windows, Linux and Mac OS X.

An alternative to the proprietary driver is the open source package OpenSC/OpenCT. This has drawbacks, as it is basically not in line with the Aladdin software framework, but it is pretty well supported. This section describes how to install and configure the OpenSC/OpenCT tools, how to store an existing X509 certificate on the token and use it with Firefox, as well as storing and SSH key pair on the same token and use it with an SSH agent.

Linux

My eToken worked easily on Ubuntu 8.04 for some time, and the following documentation has been written and verified under Debian 5.0 *Lenny*. There is a very useful website at http://www.etokenonlinux.org.

OpenSC/OpenCT installation

1. Download an unsigned binary from this russian website: http://www.aladdin.ru/upload/iblock/609/eToken_PKI_Client_4_55_Linux.rar, then install it as root 😬
   [update] Aladdin recently released the version 5 of their driver, and this file is gone for now. None of the drivers is visible on the Aladdin website. Will try to find it again.
2.
   ```
   $ sudo apt-get install opensc openct
   ```

Initialising the token

1. Prepare your X509 certificate (*usercert.pem*), and private key (*userkey.pem*)
2. Initialise the token and store your X509 private key on it:
   ```
   pkcs15-init -S userkey.pem -a 1 -u sign,decrypt --split-key
   ```
3. Copy your public key on the device too:
   ```
   pkcs15-init -X usercert.pem -v -a 1
   ```
4. Then a SSH public key can be produced from the token, to be stored in your *~/.ssh/authorized_keys* file:
   ```
   pkcs15-tool --read-ssh-key 45 | grep ssh-rsa >> ~/.ssh/authorized_keys
   ```
5. Your eToken is configured, enjoy!

Using the token with firefox

1. A new security PKCS11 module needs to be added in Firefox to use the token
2. (The following step-by-step screenshots may help: http://computing.fnal.gov/software/netidmgr/installingKCAintoFirefox.html)
3. The PKCS11 module is located at */usr/local/lib/libetpkcs11.so*

Using the token with SSH

Unfortunately, most Linux distributions provide OpenSSH clients that have not been compiled with OpenSC support. The OpenSSH source code also needs to be modified with a patch provided by OpenSC, in order to manage the PIN code of the token. The whole process is rather well described on the OpenSC website and here.

Using the token with GnuPG

I have not tried this possibility, but it is possible to also use the eToken with GnuPG to sign/encrypt content: http://www.rainerkeller.de/etoken.html Could this help packages signing?

Mac OS X leopard

OpenSC/OpenCT installation

1. The OpenSC community provides a nice Mac OS X installer called SCA. Reading this short page is highly recommended.
2. Download and install the relevant packages.
3. The binary tools are located */Library/OpenSC/bin*.

Initialising the token

1. Prepare your X509 certificate (*usercert.pem*), and private key (*userkey.pem*)
2. Initialise the token and store your X509 private key on it:

```
        /Library/OpenSC/bin/pkcs15-init -S userkey.pem -a 1 -u sign,decrypt --split-key
```
3. Copy your public key on the device too:

```
        /Library/OpenSC/bin/pkcs15-init -X usercert.pem -v -a 1
```
4. Then a SSH public key can be produced from the token, to be stored in your *~/.ssh/authorized_keys* file:

```
        /Library/OpenSC/bin/pkcs15-tool --read-ssh-key 45 | grep ssh-rsa >> ~/.ssh/authorize
```
5. Your eToken is configured, enjoy!

Using the token with Firefox

1. A new security PKCS11 module needs to be added in Firefox to use the token
2. (The following step-by-step screenshots may help: http://computing.fnal.gov/software/netidmgr/installingKCAintoFirefox.html)
3. The PKCS11 module is located at */Library/OpenSC/lib/opensc-pkcs11.so*

[update]: the PKCS11 module seems to cause some HTTP connections to be interrupted with Firefox. I tried different versions of the module with the same results. Unloading the module and restarting the Web browser seems to fix the problem. I also can no longer reader encrypted emails. It seems to be an issue with etokend, which I thought I had move away. To be investigated further.

Using the token with SSH (⚠ this session is still be worked on)

I initially tried to recompile the vanilla OpenSSH sources with hardware token support against an Intel Mac running Leopard. Unfortunately, the compilation failed with a cryptic OpenSSL error from the OpenSC header. Given I would probably have lost support for the Leopard's Keychain support, I chose another strategy.

However, SCA provides a pre-compiled SSH client in */Library/OpenSC/bin/*, which works pretty well with the eToken:

```
$ /Library/OpenSC/bin/scssh -I 0 lxadm.cern.ch
******************************************************************************
* http://cern.ch/ComputingRules : Govern the use of CERN computing facilities *
******************************************************************************
Enter PIN for Private Key:
Last login: Mon May  4 23:01:39 2009 from XXX.XXX.net
<snip>
[lxadm06] /afs/cern.ch/user/r/rwartel >
```

To add the eToken to an existing ssh-agent:

```
$ ssh-add -s0
Enter passphrase for smartcard:
Card added: 0
$ /Library/OpenSC/bin/scssh -I 0 lxadm.cern.ch
******************************************************************************
* http://cern.ch/ComputingRules : Govern the use of CERN computing facilities *
******************************************************************************
Last login: Mon May  4 23:21:27 2009 from XXX.XXX.net
<snip>
[lxadm06] /afs/cern.ch/user/r/rwartel >
```

⚠ There is one issue there: "existing agent" may be a problem on Leopard.

- The OpenSC agent can be initiated manually and works great, but the SSH_AUTH_SOCK environment variable it creates is lost if a new Terminal is opened. This means the agent is not usable across different Terminal instances.

- The Apple *ssh-agent* binary is launched automatically on the fly by *launchd*, and is persistent across Terminal opening/closure. This is very effective, and configurable in:

```
$ cat /System/Library/LaunchAgents/org.openbsd.ssh-agent.plist
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/P
<plist version="1.0">
<dict>
   <key>Label</key>
   <string>org.openbsd.ssh-agent</string>
   <key>ProgramArguments</key>
   <array>
      <string>/usr/bin/ssh-agent</string>
      <string>-l</string>
   </array>
   <key>ServiceIPC</key>
   <true/>
   <key>Sockets</key>
   <dict>
      <key>Listeners</key>
      <dict>
         <key>SecureSocketWithKey</key>
         <string>SSH_AUTH_SOCK</string>
      </dict>
   </dict>
</dict>
</plist>
```

If you replace the Apple *ssh-agent* path with the OpenSC *ssh-agent* path, the OpenSC *ssh-agent* will not work as "- l" is not a recognised parameter. This parameter is not documented on Apple's side either, which is odd:

```
$ ssh-agent -l
launch_msg response: Permission denied
$ sudo ssh-agent -l
launch_msg response: Permission denied
$ ssh-agent -h
ssh-agent: illegal option -- h
usage: ssh-agent [options] [command [arg ...]]
Options:
  -c          Generate C-shell commands on stdout.
  -s          Generate Bourne shell commands on stdout.
  -k          Kill the current agent.
  -d          Debug mode.
  -a socket   Bind agent socket to given name.
  -t life     Default identity lifetime (seconds).
$ /Library/OpenSC/bin/scssh-agent -l
scssh-agent: illegal option -- l
```

I tried to use the following */System/Library/LaunchAgents/org.openbsd.ssh-agent.plist* configuration file:

```
$ cat /System/Library/LaunchAgents/org.openbsd.ssh-agent.plist
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/P
<plist version="1.0">
<dict>
   <key>Label</key>
   <string>org.openbsd.ssh-agent</string>
   <key>ProgramArguments</key>
   <array>
      <string>/Library/OpenSC/bin/scssh-agent</string>
      <string>-a $SSH_AUTH_SOCK</string>
   </array>
   <key>ServiceIPC</key>
   <true/>
   <key>Sockets</key>
```

```
    <dict>
       <key>Listeners</key>
       <dict>
          <key>SecureSocketWithKey</key>
          <string>SSH_AUTH_SOCK</string>
       </dict>
    </dict>
</dict>
</plist>
```

But *launchd* complains:

```
org.openbsd.ssh-agent[597]: bind: Address already in use
```

Suggestions there are welcome!

[update] Dan Pritts (Internet2) rightfully pointed out that "ssh-agent doesn't have a -l option" and the command above should be "ssh-add" instead.

Using the token with GnuPG

I have not tried this possibility, but it is possible to also use the eToken with GnuPG to sign/encrypt content: http://www.rainerkeller.de/etoken.html Could this help packages signing?

This topic: Main > ITGD-eToken
Topic revision: r9 - 2010-01-19 - RomainWartel