

Security

by DiSToAGe, SuperBonBon

1. Introduction

Today security is fundamental in IT. This document will explain security concepts in JAFS, and inter relations between the server and the OS.

2. Rights

The following paragraph explains JAFS servers rights concepts. You can reach a good level of security on your system when those are well used.

- JAFS server rights are internal and are not taken from the rights of the OS filesystem (JAFS has like its own virtual filesystem).
- JAFS rights have a better granularity than in standards filesystems (read, write, execute) with :
 - Read the file
 - Delete the file
 - Rename files
 - Disable ratio on file
 - Create a file
 - Overwrite a file
 - Delete directory
 - Create directory
 - Rename directory
 - List directory content
 - Hidden files

3. Separation

Separation of OS filesystem rights and JAFS servers filesystem rights can be interesting. Rights of the filesystem on the OS are never touched. Most of other ftp server written in C/C++ are relaying on OS filesystem rights. On such servers, files are owned by the user who initiates the transfer (except for anonymous). Rights for uploaded files are with the default rights of the user. In JAFS the files are always owned by the user of the JVM (the

user who has started the server and not the user who made the connection). Rights on uploaded files are with the default rights of the user who owns the server process.

Additionally JAFS ftp users can be different of system user (sometimes a fonctionnality in ftp server written in C/C++ but not a standard). You can currently use [JAAS based authentication modules](#) to authenticate users. Such modules can be configured to authenticate username/password credentials taken from the OS (windows nt only), flat password files, RDBMS, LDAP... with the possibility to create your owns.

Even if you relay on OS user and password, server and OS are separated. If you are logged with user x on a JAFS server you are not really logged with user x on the OS but with the server's process user. In other standard servers, the user who is logged is often the user of the running process in the system. On your system you can isolate the OS user of JAFS server with unix chrooting, limit system cpu, memory, create ACL on TCP ports, restrict the number of maximum processes...

With the help of the JVM who is a separated system, you can construct a virtual system who is more or less different than the OS system. The JVM can prevent to launch native process via buffer overflow attacks. Even if it was possible, the user of the native server process is not the logged in user.

Separation of OS user and server user can be useful on systems where you have more than one server. Such systems or more and more standards on IT. Security of a whole system is always equals to the worst of its smallest part. Actually in systems attacks, differents servers are often used in conjunction to gain illegal access (upload a file with ftp, that can be later used with http via server scripting for example). If you upload a file with JAFS, the file isn't owned by the user who made the connection, so you can protect yourself from being used by other running server's users, or other logged in server's users. The JAFS server process user can be started using an OS service user to prevent from being used for logging.

4. Portability

All rights ("JAFS filesystem", users) are in one xml file. You can imagine to put this file in a read only environment (cdrom, hardware rom ...). Even in changing OS (java is portable), put the xml file, backup/restore your files and all the structure and server rights is the same. How many sysadmin hours passed on chown/chmod or windows ACL changes for restoring rights... ;-)

5. Monitoring

You can monitor files (such as configs) with IDS (Intrusion Detection System), in order to detect changes on such files. Checks via md5 sums can be also used.

6. Unix TCP ports

On unix systems only root user can open a port smaller than 1024. This is a good protection but an old problem. On servers in C/C++ there is often a not so clean workaround : A part of the server is running with root which open the server socket, and other parts are running with normal users. With Java doing such process switch is too complicated, so unless you launch the JVM process with root you won't be able to use ports below 1024. Another workaround is port redirection on the firewall/router. It can be interesting to have a listening port outside the LAN on the firewall/router different than the one on the server. But for ftp servers this port redirection can be really a nightmare because of ports used for data communications (in ftp commands socket and data socket is not the same)

If you use an UPNP capable router you are a lucky man ;-), you can do such thing automatically. With JAFS and UPNP you can dynamically open a different port on the router/firewall and redirect data to the server listening port. You can use the [firewallPort](#) parameter to setup everything. First make sure that you have a network interface [firewall NAT ports mapper](#) defined in your service config file and that your server is configured to use this network interface. Finally check that the [mapper class](#) is set to `net.sbbi.jafs.nat.UPNPNatPortsMapper`. The following example maps the port 21 on the router to the port 2121 on the server :

```
<server ...
    code="sample_server"
    port="2121"
    firewallPort="2323"
    networkInterface="default"
    controlTOS="LOWDELAY"
    dataTOS="THROUGHPUT"
    wanAccess="true">
```

7. Firewall/router TCP ports

UPNP use can also help you to open [data server sockets](#) automatically, unfortunately you can't map them (currently) to a different port on the firewall/router. Data sockets listening ports in PASV mode behind a router/firewall is a common configuration problem with newbie ftp servers sysadmins. The data ports are not open on the router and nothing can work when a client is working in passive mode. You can avoid such problems with JAFS when UPNP is enabled.

If you have no UPNP devices on your network, but another firewall based let's say on SNMP, you can create your own NAT ports mapper. You simply need to create a class who implements the `net.sbbi.jafs.nat.NatPortsMapper` interface and plug it in your desired

network interface [firewall NAT ports mapper](#) config entry.

Warning:

When you have a large range of passive data socket ports allowed in your config (1024-65000) and UPNP enabled, this can lead to a problem since by default, all ports will be mapped on the router during server startup and usually ports mapping tables on the router are big enough to handle only a few mappings (specially true on ADSL routers that can usually handle max 15 NAT mappings). Normally the driver can detect the NAT table max size and raise an exception if you try to map more ports than the table can contains, unfortunately the detection could fail with a buggy UPNP device and the exception never raised. Another problem is that it takes time to map a port with UPNP (0.5 - 1 sec), so your server could take minutes to start and minutes to shutdown. Make sure that your range is not greater than then total number of simultaneous logins. If your total number of simultaneous logins is greater than the NAT table on your router read the following text :

To avoid such size problem, use the [NATOnTheFlyPortsOpen](#) setting to specify that data socket ports in passive mode must not be opened on the UPNP router/firewall during server startup, but each time a user is sending a PASV command. The only drawback is that it will take some time (0.5 - 1 sec) to open the port on the UPNP firewall/router after a client has issued a PASV command. The port will be automatically closed as soon as the client has created the TCP socket or if a timeout occurs for data socket creation (as defined by the [timeout](#) setting).

If you want to close JAFS by hand, make sure to use the TERM signal. The TERM signal who is sent with CTRL-C too, is intercepted by the JVM process and JAFS, which closes properly all opened ports on the router/firewall. This can prevent to have open "zombies" ports on your router. The KILL signal isn't detected by the JVM.

```
kill -s TERM <process_no>
```

If you restart your router/firewall for some reasons and ports are lost, a timer managed by JAFS will reopen all required ports on the router automatically, so that you don't need to restart the server to remap the ports. The [mappings-monitoring-time](#) setting defines the time interval to check that all ports are mapped correctly.

There are also some linux kernel projects whose fonctionnality is to manage ACL on sockets : [seLinux](#) or [grsecurity](#). You can via policies, control the rights you have on ports in conjunction with the server process. Such systems can help you to create a very secure environment. Such kernel patch are often included with linux distributions. This will help you to control rights on the serveur session control and data ports.

8. Passwords

Fonctionality like [OTP \(One Time Password aka S/KEY\)](#) can help protecting against passwords sniffing.

9. Cryptography

Filter on [X.509 certificates data](#) or [SSL algorithms](#) can help you to reach an higher level of security. This can add double security : certificates and passwords. You can also add your [own certificates check routines](#), java programming language knowledge is needed.

10. Unix Symlinks

Symlink (or Symbolic Links) in Unix are special "files" on filesystem that can give redirection of files or directory on other locations of the filesystem tree. There is no rights on the system associated on Symlinks (this why you read them with rights lrwxrwxrwx). So everybody can modify them even if the owner of the Symlink is root and if the user changing the Symlink has no right on the link target file.

A Symlink Attack is an attack on which an external user change the behaviour of the environment of a user (a server user or root). Such attacks are local attacks and are used to perturb server users. Server users overwrite fundamental files, or give access to other part of the system, thinking that they make normal actions (logging, giving access ...).

For security reasons, it should be avoided to use symlink on fundamental files of the server (config, log, ftp root directory ...). Symlink in the root ftp with destination outside the root has to be avoided too for security considerations. When you can't avoid symlink use, make sure that you keep all symlinks targets inside the ftp root directory, and watch that no other tools/users changes the links.

11. RADIUS

If you use [RADIUS](#) for Authentication to a Radius Server, use it only on LAN (between ftp server and radius server). The RADIUS protocol was designed for intra-domain use only. It is not encrypted (except passwords) and use UDP protocol. So RADIUS protocol is prone to protocol spoofing, requests modifications, passwords theft, replay attacks, connection hijacking ...

12. FTP bounce attacks

You can read this security advisory to understand FTP bounce attacks. The server can be protected against this kind of attacks when the [allowFXP](#) setting is set to false. So please consider to check this setting and enable it only when necessary.