

# SPOCP Administrators Manual

by Roland Hedberg (Umeå University), Torbjörn Wiberg (Umeå University)

## Table of contents

1 Content.....	2
2 1. Introduction.....	2
3 2. Compiling spocd.....	2
4 3. Configuring the SPOCP serve.....	2
4.1 3.0 Introduction.....	3
4.2 3.1 Configuration file.....	3
4.3 4. Command line options.....	8
4.4 5. Rule file format.....	9

## 1. Content

1. *Introduction*
2. *Compiling spocd*
3. *Configuring the SPOCP server*
4. *Command line options*
5. *Rule file format*

### 2. 1. Introduction

This document is written for someone who wants to use a SPOCP server either as an authorization server or as a policy engine in some application context.

The document assumes some basic knowledge of S-expressions and how they can be used to manage authorization. If you know very little about this, you would do well to skim through *Introduction to S-expressions* before you dwell deeper into this document.

Throughout this document, I will use the canonical S-expression format when describing exactly what is produced. In discussions on transformations, I have chosen, in some cases, to use the advanced format, since that is easier to read.

### 3. 2. Compiling spocd

Normally, all it would take is to run:

```
./configure
```

```
make
```

```
make install
```

This would compile the package, with all the plug ins that are supported by libraries already installed on your system, and then place the binaries and libraries in subdirectories to `"/usr/local/spocp"`.

If this standard way is not sufficient in your case, I propose that you do the following:

```
./configure --help
```

and then use the options that apply in your case.

### 4. 3. Configuring the SPOCP serve

### **4.1. 3.0 Introduction**

The Spocd server can listen on Internet ports, as well as Unix domain sockets, and as protocol it can presently only use TCP. In the future, UDP might also be supported.

### **4.2. 3.1 Configuration file**

#### **4.2.1. 3.1.1 Server part**

The server has to know a couple of things when it starts, and these things are supposed to be learned from the configuration file. If the configuration file is 'empty', it has default values for some of the configurable things.

The format of the configuration file is as follows:

---

```
file = [servercnf] [dbackcnf] *pluginconf
servercnf = "[server]" CRLF *srvassgn
srvassgn = srvkeyword *SP "=" *SP value CRLF
srvkeyword = "calist" / "certificate" / "dhfile"
/ "entropyfile" / "log" / "passwd" / "port"
/ "privatekey" / "rulefile" / "threads" / "timeout"
/ "unixdomainsocket" / "sslverifydepth" / "pidfile" /
/ "maxconn" / "clientcert"
dbackcnf = "[dback]" CRLF dbackassgn
dbackassgn = name load *assignment
name = "name" *SP "=" *SP string CRLF
load = "load" *SP "=" *SP string CRLF
poolsize = "poolsize" *SP "=" *SP number CRLF
cachetime = "cachetime" *SP "=" *SP number CRLF
pluginconf = "[" string "]" CRLF load [poolsize] [cachetime] *assignment
```

---

assignment = string "=" 1\*UTF8 CRLF

CRLF = %x0D [ %x0A ]

SP = %x20 / %x09

digit = %x30-39

number = 1\*digit

string = 1\*printable

printable = digit / %x41-5A / %x61-7A

UTF8 = %x01-09 / %x0B-0C / %x0E-7F / UTF8-2 / UTF8-3 / UTF8-4 / UTF8-5 / UTF8-6

UTF8-1 = %x80-BF

UTF8-2 = %xC0-DF UTF8-1

UTF8-3 = %xE0-EF 2UTF8-1

UTF8-4 = %xF0-F7 3UTF8-1

UTF8-5 = %xF8-FB 4UTF8-1

UTF8-6 = %xFC-FD 5UTF8-1

---

Description of the configuration options:

**calist**

File of PEM-encoded Server CA Certificates that can be used to verify client certificates

**certificate**

The file that contains the signed certificate that belongs to the SPOCP server

**clientcert**

This maps against the openssl flags for server mode verifications of certificates

*none*

The server will not send a client certificate request to the client, so the client will not send a certificate. (SSL\_VERIFY\_NONE)

*demand*

The server sends a client certificate request to the client. The certificate returned (if any) is checked. If the verification process fails, the TLS/SSL handshake is immediately terminated with an alert message containing the reason for the verification failure.  
(SSL\_VERIFY\_PEER)

*hard* - this is the default.

Only request a client certificate on the initial TLS/SSL handshake. Do not ask for a client certificate again in case of a renegotiation (SSL\_VERIFY\_PEER|SSL\_VERIFY\_CLIENT\_ONCE).

**dhfile**

A Diffie-Hellman parameter file

**entropyfile**

A file that can be used to feed the randomness function. On a Linux system that has been running for a while, this is almost never needed.

**log**

Where SPOCP should write the log file entries. The value here can be one of two cases. Either file:<filename> or syslog:<loglevel>. For allowed log levels, see syslog(3).

**load**

Where the plugin-library is, which should be loaded

**maxconn**

The maximum number of connections that spocd will allow

**name**

Under which name this plug in should be known.

**passwd**

The password necessary to unlock the private key of the SPOCP server. If this is not provided, and the private key is protected by a password, then the server will ask for the password on startup. Something that might be feasible/necessary/correct in some circumstances. In others, it is completely unworkable.

**pidfile**

Where the server should write the process Id of the server. If you do not want a pid file, define this to be `"/dev/null"`.

**port**

The Internet port that SPOCP should listen on. If this keyword is defined, the keyword `"unixdomainsocket"` must not be defined, and vice versa.

**privatekey**

The file, which contains the private key, that should be used by this SPOCP server; this file can be the same as the one specified for the certificate. However, we strongly discourage this practice. Instead, we recommend that you separate the Certificate and the Private Key.

**rulefile**

Where the file containing the SPOCP rules is placed.

**sslverifydepth**

From the OpenSSL documentation: This directive sets how deeply `mod_ssl` should verify, before deciding that the clients don't have a valid certificate.

The depth is actually the maximum number of intermediate certificate issuers, i.e. the max number of CA certificates that are allowed to be followed while verifying the client certificate. A depth of 0 means that self-signed client certificates are accepted only; the default depth of 1 means the client certificate can be self-signed, or must be signed by a CA that is directly known to the server (i.e. the CA's certificate is under). Default is 1.

**threads**

The number of threads the server will use to service requests. This number is static, -that is, it will not change over time depending on load. Default is 5.

**timeout**

The inactivity timeout. If a client has been silent for this long (seconds) on a connection the server will unilaterally close down the connection. If set to 0, the server will never close the connection on a client. Default is 30.

**unixdomainsocket**

The unix domain socket that `spocp` should listen on.

**4.2.2. 3.1.2 Plugin part**

The native server handles what we call 'boundary conditions' through the use of plug ins. Plug ins are dynamically loaded software modules that are loaded at startup.

Since we don't know and can't guess what kind of plug ins there are going to be in the future, we have only defined how to tell the server about plug ins.

As with Apache plug ins, the plug in library contains information about which assignments you can make and what the keywords are.

There are a couple of keys that we have defined, which can be used by all plug ins:

**load**

The plug in library

**cachetime**

Definitions of the cache time for test results to use for different back ends.

**poolsize**

Every plug in can have a connection pool connected to it. It can use this pool to store open connections to information resources it uses for the future, instead of closing and opening connections for each query. Here, you can define how big the pool should be for each plug in, separately. If you do not define a poolsize for a plug in, that plug in will not be able to store connections for later use.

**4.2.3. 3.1.3 Example**

[server]

certificate = /usr/local/spocp/certs/spocpserver.pem

privatekey = /usr/local/spocp/certs/spocpserver.pem

passwd = NewKey

calist = /usr/local/spocp/certs/cacert.pem

logfile = /var/log/spocp

port = 3456

rulefile = /usr/local/spocp/pam/rules

[strmatch]

```
load =/usr/local/spocp/lib/libstrmatchplugin.so
```

```
[dback]
```

```
name =gdbm
```

```
load =/usr/local/spocp/lib/libdbackgdbm.so
```

```
gdbmfile=/usr/local/spocp/db/spocp.gdbm
```

#### 4.2.4. 3.1.4 Persistent rule store

If a dback is defined, and if the persistent store is not initialized, the rules are read from the rule file. If on the other hand, it is in existence, the rules will be read from the persistent store, and whatever is in the rule file will be ignored. During operation, if rules are added or deleted, the persistent store will be updated accordingly.

### 4.3. 4. Command line options

NAME

**spocd** - The spocp server

Synopsis

```
spocd [-d debug-level][-f config-file][-D][-t]
```

Description

Spocd is the standalone SPOCP daemon. It listens for SPOCP connections on Internet or Unix domain sockets, responding to operations received over such connections. Spocd is typically invoked at boot time, usually out of /etc/rc.local. Upon startup, spocd normally forks and disassociates itself from the invoking tty. The spocd process will print its process ID [see getpid(2)] to a .pid file, and unless something else is defined in the configuration file, the pid file will be named "spocd.pid".

Options

*-d debug-level*

Turn on debugging, as defined by num. *debug-level*, is taken as a bit string, with each bit corresponding to a different kind of debugging information.

*-f config-file*

Specifies the spocd configuration. The default is "config" in the directory where spocd is

started.

-D

Do not daemonize. If this option is set, the spocd process will **not** disassociate itself from the invoking tty.

-t

Run syntax tests for configuration files only. The program immediately exits after these syntax parsing tests, with either a return code of 0 (Syntax OK) or return code not equal to 0 (Syntax Error.)

#### **4.4. 5. Rule file format**

A formal definition:

line = ( rule / comment / "" / bconddef / include )

rule = sexp [ 1\*SP "=>" 1\*SP bcexp ] [ 1\*SP "==" 1\*SP blob ]

sexp = "(" \*SP string \*( 1\*SP element ) \*SP ")"

element = sexp / atom

blob = atom

atom = string / quoted-string / hex-string / base64-string

string = 1\*vchar

; printable except '"', '#', '%', '(', ')', '\*', '/',

; '[', '\', ']', '{', '|', '}'

vchar = x%21 / x%23 / x%24 / x%26 / x%27 / x%2B-%x2E / x%30-5A / x%5E-7B / x%7E

quoted-string = "" 1\*printchar ""

printchar = x%21 / x%23-7E

comment = "#" \*UTF8 CRLF ; has to be first on the line

bconddef = string \*SP "==" \*SP bcond

bcond = pname ":" \*unitsel ":" atom

unitsel = "{" 1\*path unit "}"

```
path = "/" string
unit = ( "/" * /
[" ( num [ "-" [ num ] ] / "-" num / "last" ) "]" )
SP = %x20 / %x09 ; space or tab
DIGIT = %x30-39
num = *DIGIT
val = 1*( UTF8 / pair )
include = ";include" filename ; reference to another rule file
hex-string = "%" 1*hexpair
hexpair = hexchar hexchar
hexchar = DIGIT / "A" / "B" / "C" / "D" / "E" / "F"
/ "a" / "b" / "c" / "d" / "e" / "f"
base64-string = "|" 1*bchar 1*2"=" ; a bit loose
bchar = DIGIT / %x41-5A / %x61-7A / %x2B / %x2F
pname = 1*pchar
pchar = DIGIT / 0x41-5A / 0x61-7A / %x2E
bcexp = bcond / bcomp
bcomp = and / or / not / ref
ref = "(" *SP "ref" 1*SP pname *SP ")"
and = "(" *SP "and" 2*( 1*SP bcomp ) *SP ")"
or = "(" *SP "or" 2*( 1*SP bcomp ) *SP ")"
not = "(" *SP "not" 1*SP bcomp *SP ")"
CRLF = %x0D [%x0A]
Example of a rule file:
strcmp := "strmatch: {/to[1]} {/o[1]}: ${0}: ${1}"
```

## *SPOCP Administrators Manual*

```
;include bconddef
```

```
#
```

```
(
```

```
  nya
```

```
  AF11_write1
```

```
  (
```

```
    role
```

```
    s
```

```
    (
```

```
      *
```

```
    set
```

```
    1
```

```
    2
```

```
  )
```

```
)
```

```
)
```

```
(nya AF11_write2
```

```
(role s 2))
```

```
#
```

```
(nya AF12_read (role ah (* set 2 3 4 5)))
```

```
== "This is a blob, which is supposed to be turned back with a positive answer"
```

```
(nya AF12_write (role ah (* set 4 5)))
```

```
(nya AF12_writeO (role ah (* set 4 5)))
```

```
#This is a boundary condition that belongs to the line above
=> (ref strcmp)
#
(nya AF13_read (role ah (* set 2 3 4 5)))
(nya AF13_write (role ah (* set 4 5)))
#
(nya AF13_writeO (role ah (* set 4 5))) => (ref strcmp) == |Rm9vQmFyCg==
(nya AF41_write (role ah (* set 2 3 4 5)))
```

```
# Server and operation access restrictions
//marcia/server/(server( ip(* prefix 213.79.154)))
//marcia/server/(server( ip 127.0.0.1))
//marcia/operation/(operation STARTTLS)
//marcia/operation/(operation QUERY
(server (ip)(host)(TransportSec (vers (* set "TLSv1/SSLv3" SSLv3))))))
//marcia/operation/(operation LOGOUT
(server (ip)(host)(TransportSec (vers (* set "TLSv1/SSLv3" SSLv3))))))
```

**Note:** Inclusion of other files are done where the their reference appears in the text.

**Note:** Rules can be divided into rule sets; the way to accomplish this is to prepend the rule definition itself with a path name. This is normally not used for application rules, but definitely for the access rules of the Spocp server itself. The default base for access rules to the Spocp server is "/"<hostname>/operation" for operation access rules, and "/"<hostname>/server" for access to the server itself. In the example above, Spocp clients with IP addresses 213.79.154/24 or 127.0.0.1 are allowed access to the Spocp server. These servers, -as long as they are not using SSLv3, do not have access to any command other than STARTTLS. Using TLS/SSL, they have access to the "QUERY" and "LOGOUT" operations.