SCOP Protocol Definition

Version 1.5, 11th September 2006
David Ingram (dmi1000@cam.ac.uk)

Contents

1 API Level 0 - Protocol Definition

This document describes the low-level SCOP protocol. It will be of interest to those who need to port the library, for example to create a new programming language binding. Of course the source code for the existing bindings is also very useful.

1.1 Header format

All transmissions sent via the SCOP protocol are preceded by a standard header, *except* for replies from the server to `verify`, `emit` and `query` directives (in fact it would be better if future revisions added the header to these too). The header has the following format (note the trailing space):

"`sCoP XXYYZZ AABBCCDD `"

`XX` is the major protocol version number represented as two hex digits (currently `01`), `YY` is the minor version number (currently `02`) and `ZZ` is the release number (currently `00`). If the major or minor versions have changed you should assume that the protocol is incompatible. If only the release number is different it is guaranteed that the protocol hasn't changed.

The next eight hex digits store the length of the transmission body. This representation is called `8hexint` format, and is also used in the body of certain transmissions to encode other integers. Since numbers in `8hexint` format are stored as an ASCII string containing 8 hex digits they may be 32 bits in length. Transmission bodies are *not* NULL-terminated (they don't need to be, since their length is stored in the header).

Future protocol revisions will add at least password authentication.

1.2 Message passing

```
client->scopserver              client<-scopserver
======                                  ==========

message <endpoint>! <msg-string>         ---

verify <endpoint>! <msg-string>          ---
---                                     <ack-8hexint>

register <name>                         ---
---                                     <msg-string>...

listen <interest>                       ---
---                                     <msg-string>...

set-source-hint <endpoint>              ---

emit <msg-string>                       ---
---                                     <ack-8hexint>
```

1.3 Admin

```
client->scopserver              client<-scopserver
======                                  ==========

clear <endpoint>                        ---

log 0|1                                 ---

terminate                               ---

reconfigure                             ---

list                                    ---
---                                     <name>!<interest>!<src-hint>!\
                                        <name>!<interest>!<src-hint>!...

query <endpoint>                        ---
---                                     <count-8hexint>
```

Note: it's a bad idea to send `list` or `query` (or call RPC's or read cookies) if other applications may be sending you messages on the same connection at the same time, due to the crosstalk possibility (responses are all handled in-band and can't be reordered).

It's therefore best to design your application to use a different socket connection to `scopserver` for these kinds of "information gathering" commands from those which receive events. This guarantees safety. Even so, you currently can't protect against *malicious* exploitation of this.

1.4 Cookies

```
client->scopserver              client<-scopserver
======                                  ==========

set-cookie <text>                       ---

get-cookie <name>                       ---
---                                     <text>
```

1.5 RPC

```
client->scopserver            scopserver->server              server
======                        ==========                      ======

call <endpoint>! <args-string>  ---                            ---
---                           scop-rpc-call <args-string>      ---


client            client<-scopserver               scopserver<-server
======                    ==========                          ======

---                       ---                   reply <results-string>
---                       <results-string>                      ---
```

2 XML data format

Simplified XML-RPC format:

```
<int>...</int>
<double>...</double>
<binary n>...</binary>
<string n>...</string>
<list n>...</list>
```

Rationale: `bool` unnecessary, `struct == array == list`. `String` and `list` lengths not strictly XML but avoid the need for `\<` and `\\` escapes and make parsing easier.

Pretty-printing: 3 spaces indentation per level. Newline after every tag. Whitespace ignored when read except inside strings. Binary coded as hex digits (double space overhead). Newline every 16 bytes (32 digits) in binary dumps.