The eXtended subsystem: Remote Procedure Call Proxy

The idea behind this subsystem is to provide a way to execute arbitrary code as RPC. This means that the subsystem will execute the intended process feeding its standard input with the data provided by the client, as well as will return to the client the data provided by the process standard output. The client can test if this subsystem is implemented by using the command XS_RPCPROXY. If the server responds “UNKNOWN”, you will realize that the subsystem is not implemented.

XS_RPCPROXY

XS_RPCPROXY makes the server enters the subsystem. Once inside the subsystem, the server remains there until the client explicitly exits the module with the proper signal header. The return status for this command could be:

OK
UNKNOWN

The UNKNOWN response is due to the lack of implementation on the server.
The OK response means the server is inside the module.

This subsystem runs in a loop of the following form:

The server reads from the network a 32bit signed integer -big-endian- header that may contain -1 which tells the subsystem to exit and return to the TFProtocol main interface or the size of the HASH which is a security token that represents a binary or scripting software to be executed as a RPC, then the HASH must be sent.

The server will return a 32bit integer header -big-endian- with -1 to indicate a failure or 0 to indicate successfully HASH processing. If -1 is sent, the subsystem is back again to wait for the exit signal (-1) or the HASH size. After the client receives 0 as a header, the server will start reading a 32bit signed integer -big-endian- header with the size of the incoming payload and next, the payload itself.

As the payload is coming, it is written to the process -RPC- standard input, until the server receives 0 as the header. At this point, the subsystem will close the standard input of the RPC process and will start to read from its standard output until the RPC process closes it. As the subsystem read the data, it will keep forwarding it to the client through the network. Once the RPC process standard output is closed, the subsystem will send a header with the value 0 to the client. After that, the server will send another 32bit integer -big-endian- header with the RPC process exit status.

Note that this isn’t an asynchronous communication, instead, the client first sends all the data to the RPC input, and then the RPC, after processing it, will send to the client through the subsystem all the output, and finally, the exit status will be sent.

About buffers and header: After the subsystem is entered which can be noted by the OK return after using XS_RPCPROXY command, any header used will be a 32bit signed integer converted to big-endian. The buffer size cannot exceed the standard communication size indicated in the main interface of TFProtocol. The pair header + payload can be sent to and received from the server as many times as need it. The header with value 0 is the signal that marks the flow in one direction has ended and the other should begin.

Once the client receives the RPC process exit status the subsystem will wait again for -1 to exit, or for another HASH size.

The RPC process exit status is process dependent but the special value 128 must be avoided in the programming of the RPC code. This value is sent to the client by the subsystem and indicates that the exec function -the one that loads the program image in memory -the RPC-- has failed executing it. The reason could be Invalid HASH, the wrong path to the binary, and so on.

About programming RPCs processes. It could be a binary, a python, java, bash script, or any other AOT o JIT language. The first thing the RPC should do at the start is to read from its standard input until receives EOF. Then it can process the read data. Then the RPC process should flush all the output to its standard output. Then it can exit.

About descriptors: The RPC process will be executed with its standard output and standard error redirected to a pipe. This implies two things: any error could be mixed with the legitime output. To avoid that, the standard output can be redirected to /dev/null. The second thing is that if the RPC process does not end after finishing sending data, it must close its standard output and standard error -if wasn't redirected- and run a flush system call. These steps guarantee that the subsystem receives all the data and notices the RPC processing has ended.