CSP Python Getting Started

Bryson Peeters

Download and Install

Download and install Python 2: https://www.python.org/
We were using using 2.7.8 but any Python 2 installation should work.

Next you want to install Git if you haven't already:http://git-scm.com/

And finally we will use Git to acquire the CSP source. Open a terminal and navigate to a directory where you wish to install the source and run the command:
`git clone https://github.com/GomSpace/libcsp.git`

CSP is available on Github at https://github.com/GomSpace/libcsp if you wish to browse through the repository.

Next we need to configure and build the source. CSP uses a Python build management tool named Waf. You can configure and build/install CSP and its Python bindings by running the following commands:

```
./waf configure --toolchain= --with-os=posix --prefix=install
--enable-promisc --install-csp --enable-bindings --enable-rdp

./waf build install
```

We were building CSP on Lubuntu 14.10 so we left the toolchain blank and set the OS to posix. Modify the options to waf as needed. More configuration options are available in the ***wscript*** file in the main directory.

--prefix=install: sets the directory to where CSP will be installed
--enable-bindings: enables the Python bindings
--enable-promisc: the Python bindings are dependent on this being enabled in the build.
--install-csp: installs the csp header
--enable-rdp: builds with RDP communications included. Default is UDP.

In the main directory create a file ***simple.py*** and copy the provided source code below into the the file. At this point everything should be ready to go and the sample application can be run via:

```
python simple.py
```

Note: The provided source code is running CSP in UDP mode even though CSP has been built with RDP included. To enable RDP you can change the options on the *connection* and *socket* objects. Connections and sockets with the correct options have been included in comments within the code. At this time though we have found that while changing these options in the C examples successfully enables RDP, in the Python bindings we hang waiting for a connection. This is yet to be resolved.

Note: the python bindings provided by CSP may look for an incorrectly named library (libpycsp.so) when the correct name is libcsp.so. This can easily be corrected by editing the import library name on line 189 of pycsp.py.

Source Code

[simple.py]

```
import imp
import sys
import threading
import traceback
from array import array
from ctypes import *

# Append the path to the CSP install libraries
sys.path.append('src/csp/install/lib')

# Import the CSP Bindings
pycsp = imp.load_source('pycsp', 'src/csp/bindings/python/pycsp.py')

MY_ADDRESS = 1
MY_PORT = 10

class ClientThread(threading.Thread):
    def __init__(self, threadID, name, counter):
        threading.Thread.__init__(self)
        self.threadID = threadID
        self.name = name
        self.counter = counter

    def run(self):
        try:
            conn = pycsp.csp_connect(pycsp.CSP_PRIO_NORM, MY_ADDRESS, MY_PORT,
10000, 0)
            #conn = pycsp.csp_connect(pycsp.CSP_PRIO_NORM, MY_ADDRESS, MY_PORT,
1000, pycsp.CSP_O_RDP)
```

```
            for i in range(20):

                p_packet = POINTER(pycsp.csp_packet_t)

                if pycsp.csp_buffer_remaining() > 0:
                    packet_addr = pycsp.csp_buffer_get(100)

                    packet = cast(packet_addr, p_packet)

                    msg = "Hello " + str(i)
                    #print "Client sending: " + msg
                    msg_str_bytes = array("B", msg)
                    msg_byte_array = bytearray(msg_str_bytes)

                    raw_bytes = (c_ubyte * 256)(*(msg_byte_array))

                    packet.contents.data = raw_bytes
                    packet.contents.length = sizeof(raw_bytes)
                    pycsp.csp_send(conn, packet, 1000)
            except pycsp.NullPointerException:
                print 'Client error.'
                print traceback.format_exc()

            print 'Closing connection on client'
            pycsp.csp_close(conn)


class ServerThread(threading.Thread):
    def __init__(self, threadID, name, counter):
        threading.Thread.__init__(self)
        self.threadID = threadID
        self.name = name
        self.counter = counter

    def run(self):

        # socket with no socket options
        socket = pycsp.csp_socket(0)
        #socket = pycsp.csp_socket(pycsp.CSP_SO_RDPREQ)

        # bind all ports to the socket
        pycsp.csp_bind(socket, pycsp.CSP_ANY)
```

```
        # start listening with a 10 connection queue
        pycsp.csp_listen(socket, 10)

        # wait for a connection, timeout 10s
        conn = pycsp.csp_accept(socket, 10000)
        while 1:
            try:
                packet = pycsp.csp_read(conn, 1000)
                port = pycsp.csp_conn_dport(conn)
                if port is MY_PORT:
                    data = bytearray(packet.contents.data)
                    print "Found packet: " + data
                    pycsp.csp_buffer_free(packet)
                else:
                    pycsp.csp_service_handler(conn, packet)
            except pycsp.NullPointerException:
                print "No packets left to read"
                break
        print 'Closing connection on Server'
        pycsp.csp_close(conn)


# initialize and kick everything off

# Init buffer with 10 packets of 256 bytes
pycsp.csp_buffer_init(10, 256 + pycsp.CSP_BUFFER_PACKET_OVERHEAD)

# Init CSP with address 1
pycsp.csp_init(MY_ADDRESS)

# Start router task with 500 word stack, OS task priority 1
pycsp.csp_route_start_task(500, 1)


print 'Starting Server...'
server = ServerThread(1, 'Server Thread', 1)
server.start()
print 'Server started.'

print 'Starting client...'
client = ClientThread(2, 'Client Thread', 2)
```

```
client.start()
print 'Client Started.'
```

Author: Bryson Peeters