Modbus Protocol Fuzzer written by Ali Abbasi

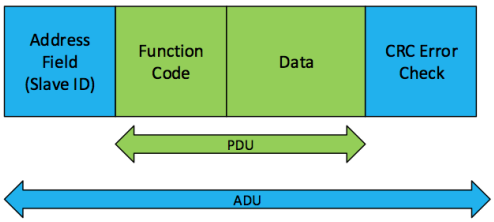
I wrote my master thesis about “Critical Infrastructure Vulnerability Assessment and Protection From Protocol Layer to Hardware Layer” in Tsinghua Unuiversity.

At the beginning I was looking for some Industrial protocol Fuzzers in order to find security holes in such protocol enabled devices. Unfortunately I couldn’t find any. If you search in Internet, you notice a Defcon talk about industrial protocol Fuzzer presented by Ganesh Devarajan, He mention in this talk that he is going to release the source code of his Fuzzer after his talk. Unfortunately he didn’t. I had chance to ask him about it and he answered me:

“I had the code out for only a few days. I was asked to take it down by many of the Industry leaders and companies due to the potential risk that they could face”

Well I started to feel bad. Why there is no public open source Fuzzer for industrial protocols? I put it in my work list for my master thesis. If you search you maybe able to find some uncomplete, undocumented Protocol Fuzzers which is based on other frameworks such as Peach or Scapy. I decided to design a very basic Modbus Fuzzer. To design your own Fuzzer you first must understand your target protocol design. Modbus is an industrial communication protocol designed by Modicon (right now Schneider Electric) in 1979 and it’s widely supported by industrial devices and it’s still one of the most common industrial protocol in the world. Modbus usually used to connect a supervisory control computer to a RTU or PLC device. The Modbus first designed for communication based on serial interfaces.

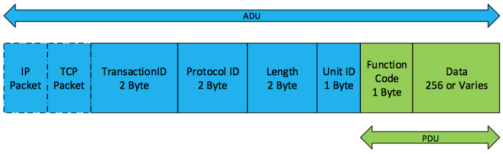
There is different variant of Modbus in the industry. It includes Modbus RTU/ASCII, which is used for serial communications and Modbus TCP/IP which using TCP/IP for Modbus communication. An initial version of Modbus general packet structure contains, address, Function Code, Data and Error checking.



[http://1.bp.blogspot.com/-UNR_HQgn_nY/Uuj2_UKBAGI/AAAAAAAAApU/AUrcmy3HFR8/s1600/Screen+Shot+2014-01-27+at+9.01.37+PM.png]

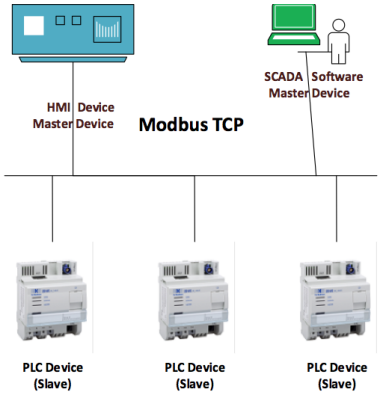
Modbus TCP packet have two element, ADU which is packet header and PDU which is the payload of the packet. The Transaction ID, Protocol ID and Length contain two byte while Length and Function code are only one byte. The Data in standard version is 256 byte but it can be up to 65535 bytes (in some machines).

The packet structure is simple but just for important parts we have to mention that protocol ID in Modbus is always 0×0000. The length of the packet is actual Modbus TCP packet minus 6 byte and UnitID in a Modbus packet is usually 0×00 or 0xff.



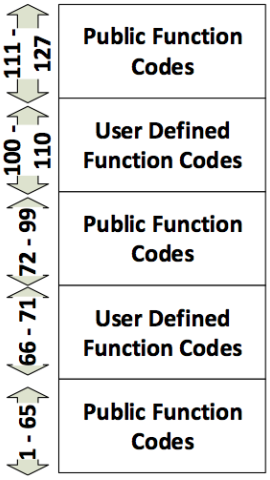
[http://1.bp.blogspot.com/-iGQJm05Z1WY/Uuj3UobPlWI/AAAAAAAAApc/cEXWVHQn_vE/s1600/Screen+Shot+2014-01-27+at+9.04.16+PM.png]

Because of very old design in Modbus, This protocol don’t support any encryption and don’t have any protection against reply attacks. A Modbus communication can contain Modbus slave device and Modbus master device. Modbus master device usually refer to the Modbus client which can be SCADA machine or HMI device. And Modbus slave refer to a RTU or PLC devices.



[http://4.bp.blogspot.com/-Al623Jn4qQQ/Uuj3hbwXKrI/AAAAAAAAApk/Ei8R7DR09RQ/s1600/Screen+Shot+2014-01-27+at+9.06.12+PM.png]

Modbus supporting different function codes. There is two type of function codes in Modbus. The first one is Modbus general function codes which is supported by most of the Modbus enabled devices in the industry and another category is user defined Modbus function codes. Modbus function codes starting from function code 01.



[http://3.bp.blogspot.com/-zrD07HSFbCE/Uuj30xO2ezI/AAAAAAAAAps/h7FHYhqHh2E/s1600/Screen+Shot+2014-01-27+at+9.08.13+PM.png]

Also we have to indicate that data encoding type in Modbus is Big Endian, which means that when a numerical quantity larger than a single byte is transmitted, the most significant byte is sent first. So for example the data inside e Modbus packet which is like 0×1234 will be send 0×12 first and then 0×34. It seems obvious that 32-bit and 64-bit values should also be transferred using big-endian order. However, some manufacturers have chosen to treat 32-bit and 64-bit values as being composed of 16-bit words, and transfer the words in little-endian order.

Fuzzing can be categorized based on how we generate the data. The first category called mutation based fuzzing. In mutation based fuzzing the Fuzzer will first capture the standard packet or payload and then the data units will change randomly or based on heuristic algorithms. For example we can change the standard Modbus TCP packet length to 300 byte by adding multiple data to different segment of the protocol. In second type of fuzzing which called generation based fuzzing we will follow the standards in the protocol. Which means the data will remain as standard Modbus TCP packet length but it will change the different part of the protocol in order to make an unexpected result.
We designed a Modbus Fuzzer. We call it dumb Fuzzer. Since it randomly generate data for Modbus protocol and it is working as generation based Fuzzer. The Fuzzer successfully worked against some Chinese made PLC's. The program written in python. The Fuzzer will generate Transaction ID, Protocol ID, Length, Unit ID, Function Code and Function Data. By making such data we can find possible vulnerabilities inside a Modbus server. By using -I switch the user are able to send its own payload to Modbus server. The -D switch is required to generate custom data and send it to the Modbus server.By using -S command you can use the fuzzer to fuzz multiple network devices.
Of course it is not very efficient fuzzer. Actually it don't have any smart generation algorithm and there is no feedback control check to mutate the algorithm. Using peach with its heuristics algorithms would probably better.
I will talk about designing fuzzer using peach in future. The code also availabe in GIthub. You can get it from here:

https://github.com/bl4ckic3/Modbus-Fuzzer

```
#!/usr/bin/python
'''
Created on Apr 16, 2013 v0.1
```

```
Modified and added scanning function, Dec 14, 2013 v0.2
@author: Ali, Sami
'''
import socket
import sys
from types import *
import struct
HOST = '127.0.0.1'  # The remote host
dest_port = 502      # The same port as used by the server
TANGO_DOWN = ''
sock = None
dumbflagset = 0;
def create_connection(dest_ip, port):
  try:
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
  except socket.error, msg:
    sys.stderr.write("[ERROR] %s\n" % msg[1])
    sys.exit(1)
  HOST = dest_ip
  print 'Connecting to %s' % HOST
  try:
    sock.settimeout(0.001)
    sock.connect((HOST, dest_port))
    #sock.settimeout(None)
  except socket.error, msg:
    #sys.stderr.write("[ERROR] %s\n" % msg[1])
    sys.exit(2)
  print 'Connected successfully'
  return sock
def dumb_fuzzing(dest_ip):
 sock = create_connection(dest_ip, dest_port)
 length1 = 0
 length2 = 6
 unitID = 1
 for transID1 in range(0,255):
  for transID2 in range(0,255):
   for protoID1 in range(0,255):
    for protoID2 in range(0,255):
#     for length1 in range(0,255):
#      for length2 in range(0,255):
#       for unitID in range(0,255):
        for functionCode in range(0,255):
         for functionData1 in range(0,65535):
          for functionData2 in range(0,65535):
           TotalModbusPacket = struct.pack(">B", transID1) + \
                     struct.pack(">B", transID2) + \
                     struct.pack(">B", protoID1) + \
                     struct.pack(">B", protoID2) + \
                     struct.pack(">B", length1) + \
                     struct.pack(">B", length2) + \
                     struct.pack(">B", unitID) + \
                     struct.pack(">B", functionCode) + \
                     struct.pack(">H", functionData1) + \
                     struct.pack(">H", functionData2)
#           print '     transID protoID length uID funcCode funcData'
           print 'Sent Msg : %02x %02x, %02x %02x, %02x %02x,  %02x,  %02x,  %04x, %04x' % (transID1, tra
           try:
            sock.send(TotalModbusPacket)
           except socket.timeout:
            print ''
           try:
            data = sock.recv(1024)
            print 'Received %s:' % repr(data)
           except socket.timeout:
            print ''
 sock.close()
def smart_fuzzing(dest_ip, msg):
  sock = create_connection(dest_ip, dest_port)
  strInput = msg
  dataSend = ""
  shortInput = ""
  sock.send(msg)
#  cnt = 1
#  for chInput in strInput:
#       shortInput += chInput
#    if cnt%2 == 0:
#      intInput = int(shortInput,16)
```

```
#      dataSend += struct.pack(">B", intInput)
#      print 'short: %s, intInput: %s, dataSend: %s'%(repr(shortInput), intInput, repr(dataSend))
#      shortInput = ""
#    cnt += 1
#  print '%s' % repr(dataSend)
#  sock.send(dataSend)
#  print 'sent: %s' % repr(dataSend)
  print '%s' % repr(msg)
  try:
    dataRecv = sock.recv(1024)
    print >>sys.stderr, 'received: %s' % repr(dataRecv)
  except socket.timeout:
    print 'recv timed out'
#  if dataRecv==TANGO_DOWN:
#    print 'TANGO DOWN !!!'
  sock.close()
def atod(a): # ascii_to_decimal
  return struct.unpack("!L",socket.inet_aton(a))[0]
def dtoa(d): # decimal_to_ascii
  return socket.inet_ntoa(struct.pack("!L", d))
def scan_device(ip_range):
  net,_,mask = ip_range.partition('/')
  mask = int(mask)
  net = atod(net)
  for dest_ip in (dtoa(net+n) for n in range(0, 1<<32-mask)):
    try:
      sock = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    except socket.error, msg:
      sock.close()
    try:
      sock.settimeout(0.2);
      sock.connect((dest_ip, dest_port))
    except socket.error, msg:
      print "connection error at %s" % dest_ip
      continue
    except socket.timeout:
      print 'ip %s timeout error' % dest_ip
      continue
    unitID = 0
    dataRecv = ''
    while True:
      dataSend = struct.pack(">H", 0) \
              + struct.pack(">H", 0) \
              + struct.pack(">H", 6) \
              + struct.pack(">B", unitID) \
              + struct.pack(">B", 3) \
              + struct.pack(">H", 0) \
              + struct.pack(">H", 1)
      try:
        sock.send(dataSend)
        print "Sent: %s to %s" % (repr(dataSend), dest_ip)
      except socket.error:
        print 'FAILED TO SEND'
        #sock.close()
        #continue
      try:
        dataRecv = sock.recv(1024)
        print "Recv : %s" % repr(dataRecv)
      except socket.timeout:
        sys.stdout.write('.')
      if len(dataRecv) < 1:
        sys.stdout.write('.')
        #print "."
        unitID += 1
      else:
        print '\nunit ID %d found at IP %s' % (unitID, dest_ip)
        if dumbflagset == 1 :
          print 'now starting dumb fuzzing'
          dumb_fuzzing(dest_ip)
        break
  sock.close()
# main starts here
if len(sys.argv) < 3:
  print "modbus fuzzer v0.1"
  print ""
  print "Usage: python modFuzzer.py [-D] [destination_IP]"
  print "              [-I] [destination_IP] [packet]"
```

```
  print "              [-S] [IP_range]"
  print "              [-SD][IP_range]"
  print " "
  print "Commands:"
  print "Either long or short options are allowed."
  print " --dumb  -D Fuzzing in dumb way"
  print " --input  -I Fuzzing with given modbus packet"
  print " --scan  -S Scan the modbus device(s) in given IP range"
  print " --sc_dumb -SD Scan the device(s) and doing dumb fuzzing"
#  print " "
#  print "Option:"
#  print " --port  -p Port number"
  print " "
  print "Example:"
  print "python modFuzzer.py -D 192.168.0.123"
#  print "python modFuzzer.py -D 192.168.0.123 -p 8888"
  print "python modFuzzer.py -I 192.168.0.23 000000000060103000A0001"
  print "python modFuzzer.py -S 192.168.0.0/24"
  print ""
  exit(1)
argv1 = sys.argv[1]
argv2 = sys.argv[2]
argv3 = ''
if len(sys.argv) > 3:
  argv3 = sys.argv[3]
if (argv1=='-D') or (argv1=='--dumb'):     # dumb fuzzing
  dumb_fuzzing(argv2)
  sys.exit(1)
elif (argv1=='-I') or (argv1=='--input'):     # smart user input
  smart_fuzzing(argv2, argv3)
elif (argv1=='-S') or (argv1=='--scan') or (argv1=='-SD'):    # scan device
  if argv1 =='-SD' :
    dumbflagset = 1
  scan_device(argv2)
sys.exit(0)
```

Posted 29th January 2014 by Milad Kahsari Alhadi

0 Add a comment