Dunker
Inactive

Offline

Posts: 4

Thank You
-Given: 0
-Receive: 12

IEC 62056-21 Elec Meter Hacking via IR

« **on:** May 18, 2015, 11:37:22 23:37 »

It's been quite a while since I've looked at IEC 62056-21. I used to have a Siemens S2A at home and now I have an Ampy 5235. I bought an S2A from ebay (couldn't really be playing with the official meter could I). For the infra red interface I made the common 741 op-amp based interface from the net and it worked ok but I almost immediately figured it wasn't good enough (and I wanted far better quality of signal and also USB) so I designed my own.

For development I used Linux and Perl and recently someone asked me to port to Python which I have just started when time permits. I have reversed the Siemens P0/P2 password exchange algo and can freely communicate with the S2A. I did a bit of fuzzing with the meter and did get some interesting preliminary results (I can talk about this more later on). Seeing as a lot of the meters out there seem to be from the same base manufacturer I was wondering if my Siemens algo will work on other meters.

It could now be time to revive this project on 62056-21 meter interaction what with the new daisy chain smart meters that are apparently on their way.

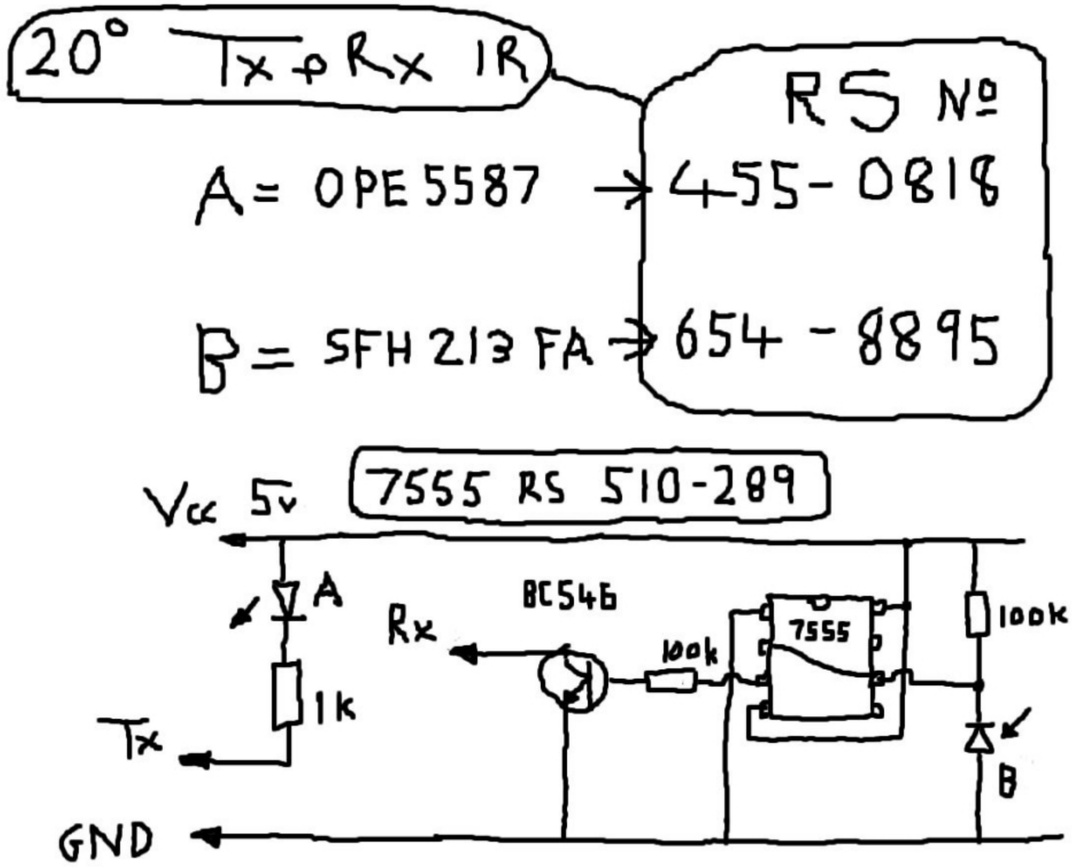
Anyway, if anyone fancies having a go at building an interface and joining in then the more the merrier. As I said, I use Linux for developement but if anyone wants to use Windows then I do have an XP box I could use - maybe with Strawberry Perl or Python for Windows. Only XP though. I have various scopes including a Picoscope and a logic analyzer to aid in the project from my end and can test for unknown problems in submitted code snippets for people that don't have the facilities.

The components for the interface are not expensive and test meters can be easily and cheaply bought from ebay etc. For the USB interface you will need a few components and a USB to Serial converter module. GET ONE WITH AN FTDI CHIP ON IT! I have tried a number of these modules and the FTDI ones are by far the best. The drivers for these seem to be the most complete and functional - maybe with the exception of the INTERCHAR_TIMEOUT but we will have to live with that.

These are the USB to Serial modules I have used:

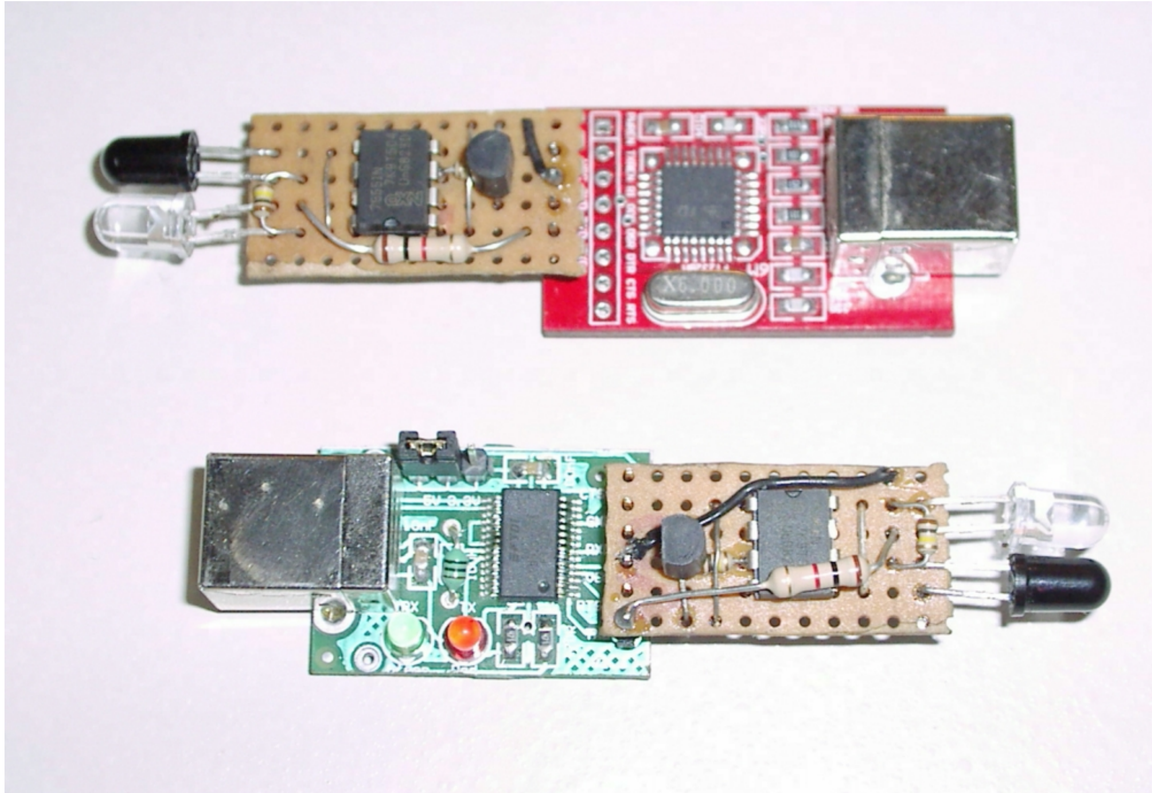
These are both the same except one has LED's that flash when data is flowing. (Both chips on these are the same but just have different packages).

This is the diagram for the infra red interface:



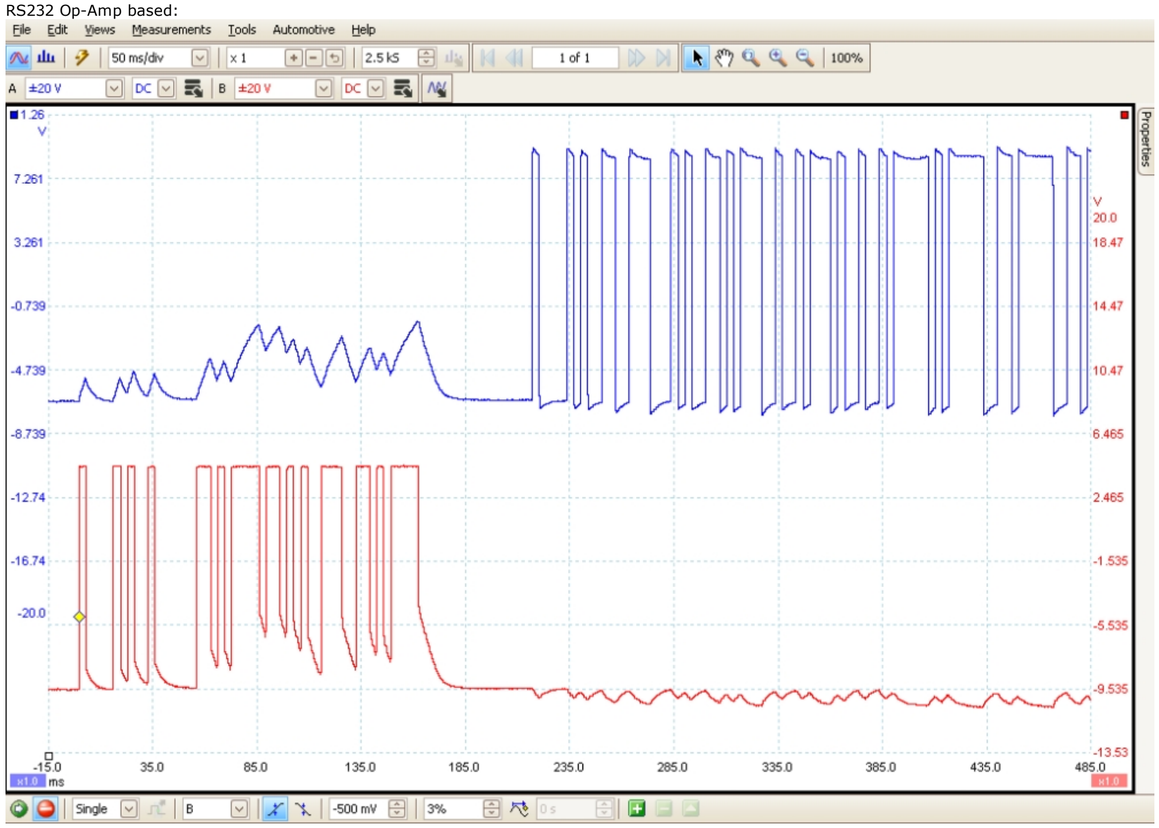
(Quality diagram I know).

And these are the USB IR modules built up:

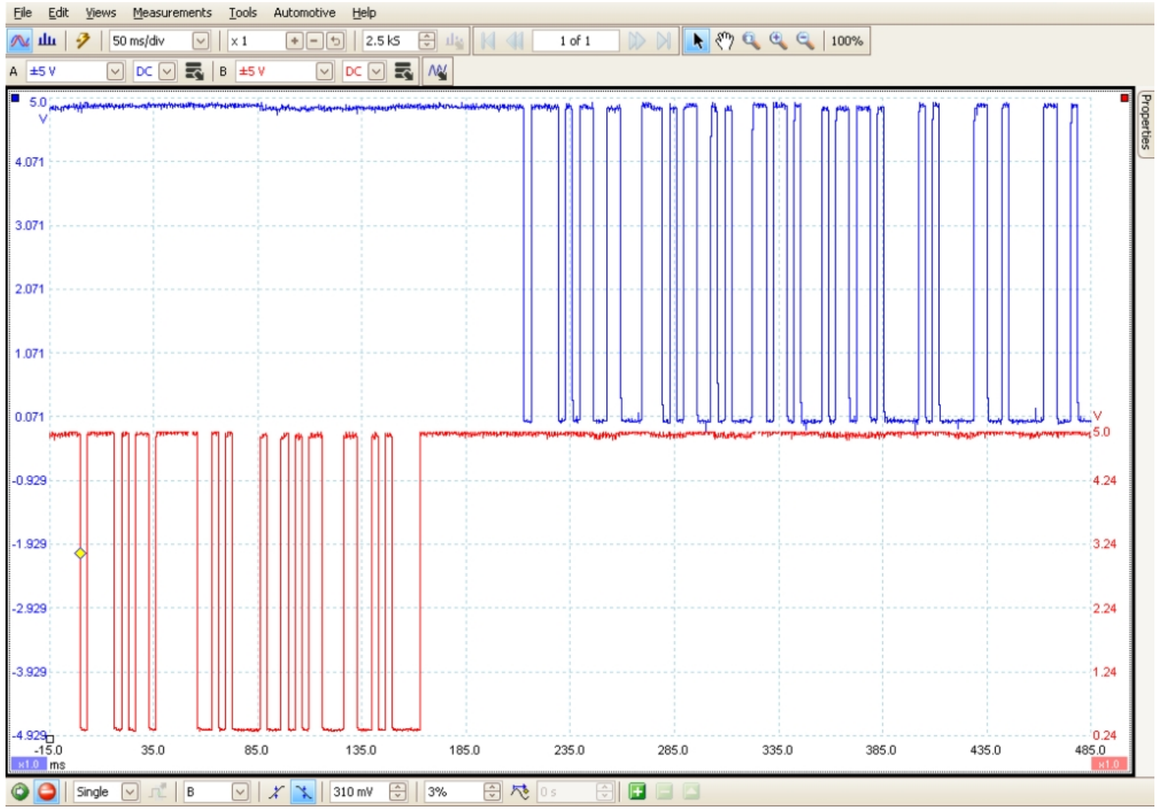


In the following pictures you can see the differences in the signal quality between the RS232 Op-Amp based interface and the USB interface. Don't get me wrong, the Op-Amp 741 interface does work but without a scope it can be a pain to set up. My USB interface uses a 7555 standard timer IC as a schmitt trigger to clean up the received signal (and then a transistor to invert the 7555 output). If anyone does want the 741 based diagram it is out there on the net somewhere but I can't find my diagram.

RS232 Op-Amp based:



USB/7555 Based:



Before building the interface it might be a good idea to read IEC62056-21 and see what you will be up against. It looks complicated but when you get your interface up and running and can see some data going in and out the clouds do begin to clear a little.
To start with we will need to get the interface working and talking to the meter:

```
First of all, we need to send the opening request message to the meter: / ? ! CR LF  which is  /?!\r\n
(If you are reading 62056-21.pdf don't worry about the device address from 6.3.1 - just leave this null).

If all is well the meter sends back its ID message. 50ms after the above send my meter returns the following:
2f 46 4d 6c 34 41 30 30 30 30 56 38 30 0d 0a  which is  /  F  M  l  4  A  0  0  0  0  V  8  0 CR LF

To close the connection with the meter now send the break message: SOH B 0 ETX BCC  which is  \x01B0\x03q
```

Here is the code for testing the connection, test_signon.py:

Code:

```
# test_signon.py

#-----------------------------------------------------------------------------------------------------------------------------------

import serial
import time
from binascii import hexlify
from watchdog import wdTimer

ser = serial.Serial()

ser.port = "/dev/ttyUSB0"                       # set interface instance
ser.baudrate = 300                              # set baudrate to 300
ser.parity = serial.PARITY_EVEN                 # set parity to even parity
ser.bytesize = serial.SEVENBITS                 # number of bits per byte
ser.stopbits = serial.STOPBITS_ONE              # number of stop bits
#ser.timeout = None                             # block read
#ser.timeout = 0                                # non-block read
ser.timeout = 0.05                              # timeout block read -set to 50ms to start with (value for 300bd)
```

And the code for watchdog.py:

Code:

```
# file: watchdog.py

import signal

class wdTimer(Exception):
# usage wdTimer(timeout_in_seconds) or Watchdog() >> defaults to 2 seconds
        def __init__(self, timeout=2):
                self.time = timeout

        def __enter__(self):
                signal.signal(signal.SIGALRM, self.handler)
                signal.alarm(self.time)

        def __exit__(self, type, value, traceback):
                signal.alarm(0)

        def handler(self, signum, frame):
                raise self
```

This is what test_signon.py looks like when it works:

```
~/test$ python test_signon.py
sending: / ? ! CR LF
data read: 2f464d6c3441303030305638300d0a
length: 15
sending: SOH B 0 ETX q
~/test$
```

This code is very basic - nothing fancy at all - but it works. Later functional code is marginally better - I've never used python so don't be taking the p1$$. Once this part is working there is a lot more stuff to think about. There is the baudrate change after signon and then the P0-P2 password exchange to gain access to the other modes of the meter.

Here is a screenshot of one of the later scripts connectiong to the meter with baudrate change and password exchange:

```
[redacted]@satan2:~/extra/elec$ perl test18.pl

>   / ? ! CR LF                                    written   5    hex = 2f3f210d0a > Tx

<   / F M l 4 A 0 0 0 0 V 8 0 CR LF                read     15    hex = 2f464d6c3441303030305638300d0a < Rx

>   ACK 0 4 1 CR LF                                written   6    hex = 063034310d0a > Tx

<   SOH P 0 STX ( 6 1 8 0 ) ETX o                  read     12    hex = 01503002283631383029036f < Rx

>   SOH P 2 STX ( 2 5 C 8 ) ETX RS                 written  12    hex = 01503022283235433829031e > Tx

<   ACK                                            read      1    hex = 06 < Rx

>   SOH R 1 STX 0 5 0 0 ( 0 8 ) ETX n              written  14    hex = 015231023035303028303829036e > Tx

<   STX ( 4 6 1 9 1 9 4 3 ) ETX BEL                read     13    hex = 022834363139311393433290307 < Rx

>   SOH R 1 STX 0 5 2 0 ( 0 8 ) ETX l              written  14    hex = 015231023035323028303829036c > Tx

<   STX ( 6 3 8 2 1 6 7 2 ) ETX SI                 read     13    hex = 022836338323136373229030f < Rx

>   SOH R 1 STX 0 5 3 0 ( 0 8 ) ETX m              written  14    hex = 015231023035333028303829036d > Tx

<   STX ( 8 2 3 6 0 3 7 0 ) ETX HT                 read     13    hex = 022838323336303337302903 09 < Rx

>   SOH W 1 STX 1 3 8 3 ( 0 ) ETX _                written  13    hex = 015731023133383328302903 5f > Tx

<   ACK                                            read      1    hex = 06 < Rx

>   SOH B 0 ETX q                                  written   5    hex = 0142300371 > Tx

[redacted]@satan2:~/extra/elec$
```

The baudrate change is immediately after the ACK 0 4 1 CR LF - see IEC62056-21 6.3.3
The password exchange are the following P 0 and P 2 lines.

All the files you will need for now are here: http://www.team594.site50.net/iec62056/

« Last Edit: May 19, 2015, 02:22:14 14:22 by Dunker »

Logged

Dunker
Inactive

Offline

Posts: 4

Thank You
-Given: 0
-Receive: 12

Re: IEC 62056-21 Elec Meter Hacking via IR

« **Reply #1 on:** May 25, 2015, 12:24:56 00:24 »

A little more info:

After signing on to the meter and after the meter sends its ID message we need to send the 'Acknowledgement/option select message' (iec62056-21 6.3.3). Explanation of the options is covered further in the iec62056-21 pdf file. To do anything worth while with these meters we need to get them in to programming mode. This involves a password exchange. According to the standard there can be different ways in which this can be implemented. Certain meters use a simple ascii password but the S2A I'm testing uses the P0 - P2 password exchange. It might have a simple password but I haven't looked for it - maybe I should have done, it would have saved a lot of work.

I have attached a file named 3.zip which contains everything pertinent to this post. If in future anyone needs help with anything the name of the zip file is important as modules contained in the zip only work with the scripts in that particular zip - they may be modified in the future and I can't be arsed with version numbers at this point.

Here is a readout of the new test_signon.py:

```
/dev/ttyUSB0 opened successfully
>  / ? ! CR LF
<  / F M l 4 A 0 0 0 0 V 8 0 CR LF   >>>   Data: 2f464d6c3441303030305638300d0a Length: 15
FMl : Siemens Measurements Ltd. (Formerly FML Ltd.)
  l : Last character of flag is in lower case, 20ms min reaction time
  4 : Baudrate changeover identification, rate = 4800bd
 ID : Meter identification string = A0000V80
>  SOH B 0 ETX q
/dev/ttyUSB0 closed successfully
```

Here, there is a little decoding of the ID message returned by the meter. I think V80 is the firmware version number.

Here is a readout of the meter entering programming mode with test_modec.py:

```
/dev/ttyUSB0 opened successfully
>  / ? ! CR LF
<  / F M l 4 A 0 0 0 0 V 8 0 CR LF   >>>   Data: 2f464d6c3441303030305638300d0a Length: 15
FMl : Siemens Measurements Ltd. (Formerly FML Ltd.)
  l : Last character of flag is in lower case, 20ms min reaction time
  4 : Baudrate changeover identification, rate = 4800bd
 ID : Meter identification string = A0000V80
>  ACK 0 4 1 CR LF
0 = normal mode  / 4 = 4800bd / 1 = programming mode
Switching to 4800bd
<  SOH P 0 STX ( 4 1 5 0 ) ETX `   >>>   Data: 015030022834313530290360 Length: 12
>  SOH P 2 STX ( 4 1 5 C ) ETX DC1
<  ACK   >>>   Data: 06 Length: 1
>  SOH B 0 ETX q
/dev/ttyUSB0 closed successfully
```

As you can see in this example, the meter asks for a password with: SOH P 0 STX (4 1 5 0) ETX (ETX is just 0x03, the block control character). To gain proper access to the meter the '4150' value needs to be manipulated to produce (in this case) the value '415C'. If the value returned to the meter is wrong the meter spits the dummy, breaks and disconnects. If the meter sends 'ACK' then you're in! Time to start having fun.

Here is the P0 - P2 password generation algorythm:

```
Access code sent from meter - 4 ascii digits.
Key generated and sent back to meter, 4 ascii digits.

Code sent in the form [X1].[X2].[X3].[X4] where each digit has 4 bits. (16 bits total):

[X1b3.X1b2.X1b1.X1b0].[X2b3.X2b2.X2b1.X2b0].[X3b3.X3b2.X3b1.X3b0].[X4b3.X4b2.X4b1.X4b0]

4 new values are created by rotating the bits right twice to give M4..M1:

[         M4        ].[         M3        ].[         M2        ].[         M1        ]
[X4b1.X4b0.X1b3.X1b2].[X1b1.X1b0.X2b3.X2b2].[X2b1.X2b0.X3b3.X3b2].[X3b1.X3b0.X4b3.X4b2]

3 new values x,y and z are created by xoring M3..M1 with M4:

[    x    ].[    y    ].[      z      ]
[M3 xor M4].[M2 xor M4].[M1 xor M4 + 1]

These values combined into one word R1 which is processed:

  R1 := 0xyz

  if R1 = 0000 then R1 :=0001

  R2 := R1 * 20h

  for i := 0 .. 10h
    if R2 & 0001 = 0001 then R2 := R2 xor 12h
    R2 := RSH R2
  next i

  L1 := R2 & 0F   (mask lower nibble ie. R2 & 1111)
  L2 := L1 & 0C   (mask upper 2 bits ie. L1 & 1100)
  L3 := L1 & 03   (mask lower 2 bits ie. L1 & 0011)

Remember R1 consists of 3 digits, x,y and z (0xyz):

S2 := x xor L1
S1 := y xor L1
S0 := z xor L1

Combine these 4 bit digits to give the word K1:

K1 := [   0   ].[         S2        ].[         S1        ].[         S0        ]
K1 := [0.0.0.0].[S2b3.S2b2.S2b1.S2b0].[S1b3.S1b2.S1b1.S1b0].[S0b3.S0b2.S0b1.S0b0]

then K1 := K1 * 4 (LSH K1 x2) to give:

K1 := [       p     ].[         q         ].[         r         ].[      s      ]
```

```
K1 := [0.0.S2b3.S2b2].[S2b1.S2b0.S1b3.S1b2].[S1b1.S1b0.S0b3.S0b2].[S0b1.S0b0.0.0]

K1 := K1 + L3 * 4000h (L3 bits = 0.0.a.b become a.b.0.0 and are added to p)

K1 := K1 + L2 / 4h    (L2 bits = c.d.0.0 become 0.0.c.d and are added to s)

Convert the 4 digits of K1 to ascii and send to meter.

Thar shi blahws.
```

A lot of the meters out there are made by the same manufacturer and are rebadged. I wonder how many different meters this algo will work for (or with slight modification).

The block control character is described in the pdf and my implementation is in the iec62056core.py module. Also included in the module is the P2 password generation routine. The 3.zip file also contains two 'sign off' scripts because if something goes wrong with one of your test scripts the meter can sit there for a couple of minutes before it times out and is ready to accept another input. Everything is fully functional (works for me here anyway).

« Last Edit: June 01, 2015, 11:22:16 23:22 by Dunker »

Logged

Dunker
Inactive

Offline

Posts: 4

Thank You
-Given: 0
-Receive: 12

Re: IEC 62056-21 Elec Meter Hacking via IR

« **Reply #2 on:** June 02, 2015, 12:03:31 00:03 »

Before we can start doing anything with the meter it might be an idea to find out what memory locations the meter has that we can modify. The best way to do that is to write a value to each memory location and see what the response from the meter is. (I say value and not byte because on the S2A the meter replies with nibbles and not bytes when reading single values). If the meter replies with an ACK then the memory location is writeable. If it is not, the meter will (usually) reply with an error. I say usually because there will be instances when the meter might stay quiet on purpose. Also, writing the wrong value to certain memory locations will brick the meter. For example, on the Siemens S2A, writing a wrong value to memory locations @ 0x600 or there abouts will cause the meter to brick with ERR8 shown on its display and it can only be brought back to life by physically removing and reprogramming the meters internal eeprom with a known good dump. Because of the fear of bricking the meter it is prudent to read one value at a time and then immediately write that value back to the meter at the same memory location. This way you are not modifying anything.

To read a value from the meter use:
SOH R 1 STX <addr> (01) ETX BCC
where addr is the memory address and BCC is the block control character.

Similarly, the write command is:
SOH W 1 STX <addr> (01) ETX BCC

You can omit leading zeroes from the addr string and the number to read value - (01) so:
SOH R 1 STX 0234 (01) ETX BCC is the same as **SOH R 1 STX 234 (1) ETX BCC**

The exact construction of these commands can be deduced when looking at fuzzmem.py and reading IEC62056-21 6.3.7. If anyone needs help with this let me know. I have added fuzzmem.py to 3.zip in the previous post.

Here is the listing:
Code:

```
# fuzzmem.py
#
# works with files from 3.zip
# usage: fuzzmem.py -s <start_addr> -e <end_addr> -o <filename>
# start_addr = first memory location to try
# end_addr = last memory location to try
# filename = name of file to write results to

import argparse
import iec62056core
import iec62056serial
from binascii import hexlify
from datetime import datetime

def auto_int(x):
    return int(x,0)

def pad4(inValue):
        outString = hex(inValue)[2:].upper()
```

Run fuzzmem.py or fuzzmem.py -h for usage.

Here is a sample run:

```
~/elec meter/iec62056$ python fuzzmem.py -s 0x130 -e 0x13f -o outfile
# 23:27:30 01-06-2015
# Start Address = 0130, End Address = 013F
/dev/ttyUSB0 opened successfully
>  / ? ! CR LF
<  / F M l 4 A 0 0 0 0 V 8 0 CR LF   >>>   Data: 2f464d6c3441303030305638300d0a Length: 15
FMl : Siemens Measurements Ltd. (Formerly FML Ltd.)
  l : Last character of flag is in lower case, 20ms min reaction time
  4 : Baudrate changeover identification, rate = 4800bd
 ID : Meter identification string = A0000V80
>  ACK 0 4 1 CR LF
0 = normal mode  / 4 = 4800bd / 1 = programming mode
Switching to 4800bd
<  SOH P 0 STX ( 2 1 1 0 ) ETX b   >>>   Data: 01503002283231313029036 2 Length: 12
>  SOH P 2 STX ( 5 6 6 3 ) ETX d
<  ACK   >>>   Data: 06 Length: 1
Start memory fuzzing...
address: 0130 | value read: E | writeback response: ERR7
address: 0131 | value read: D | writeback response: ERR7
address: 0132 | value read: 9 | writeback response: ERR7
address: 0133 | value read: E | writeback response: ERR7
address: 0134 | value read: D | writeback response: ERR7
address: 0135 | value read: 6 | writeback response: ERR7
address: 0136 | value read: D | writeback response: ERR7
address: 0137 | value read: 8 | writeback response: ERR7
address: 0138 | value read: 0 | writeback response: ERR7
```

```
address: 0139 | value read: 0 | writeback response: ERR7
address: 013A | value read: F | writeback response: ERR7
address: 013B | value read: F | writeback response: ERR7
address: 013C | value read: 8 | writeback response: ACK   >>> address writeable !!!
address: 013D | value read: 5 | writeback response: ACK   >>> address writeable !!!
address: 013E | value read: 0 | writeback response: ACK   >>> address writeable !!!
address: 013F | value read: 0 | writeback response: ACK   >>> address writeable !!!
End memory fuzzing...
>  SOH B 0 ETX q
/dev/ttyUSB0 closed successfully
~/elec meter/iec62056$
```

Here, a file called "outfile" is created (or appended) with the results of the fuzzing.

Also attached is S2A_fuzz.zip containing a fuzzmem run on my S2A showing its memory locations and if they are writeable. I know what a few of the locations are for and also the internal eeprom is from 0x1000 to 0x13FF.

Right then, I'm now off to the Isle of Man TT Races for a week so I'll only be fuzzing my head with guiness for the next week or so

« Last Edit: June 02, 2015, 01:59:26 01:59 by Dunker »

Logged

Dunker
Inactive

Offline

Posts: 4

Thank You
-Given: 0
-Receive: 12

Re: IEC 62056-21 Elec Meter Hacking via IR
« **Reply #3 on:** July 28, 2015, 12:32:46 00:32 »

I'm just on with writing a small command line set of meter instructions. After log-on the meter stays active for a couple of minutes which gives us time to pick and choose what we want to send to the meter on the fly. Also, a parser to read a set of instructions from a file that will log-on and then send the instructions to the meter one by one and then reads the results (to screen and a file). I'll either amend this post with the scripts or add to the thread if there's any input.

A quick note <23/08/2015> been in hospital for 3 weeks, just got out, burst appendix. I'll be back on with this ASAP.

« Last Edit: August 23, 2015, 03:32:33 15:32 by Dunker »

Logged

pickit2
Moderator
Hero Member

Offline

Posts: 3814

Thank You
-Given: 556
-Receive: 2045

There is no evidence that I muted SoNsIvRi

Re: IEC 62056-21 Elec Meter Hacking via IR
« **Reply #4 on:** August 10, 2015, 10:41:26 10:41 »

Quote from: kcwcc on August 03, 2015, 08:23:05 20:23

> nice work

looking At your time in forum 7 years and you post Crap Like This...
also asking for a link to a book ... you find it and just post I found it ...
Where is a link to the book you Asked for Then Found...

Logged

Note: If you have no posts other than, I want or reporting a dead link Then you can't complain If I remove your post So Stop Leeching

Pages: [**1**]

PRINT

« previous next »

Jump to: => General Electronics go

DISCLAIMER
WE DONT HOST ANY ILLEGAL FILES ON THE SERVER
USE CONTACT US TO REPORT ILLEGAL FILES
ADMINISTRATORS CANNOT BE HELD RESPONSIBLE FOR USERS POSTS AND LINKS

... Copyright © 2003-2999 Sonsivri.to ...
Powered by SMF 1.1.18 | SMF © 2006-2009, Simple Machines LLC | HarzeM Dilber MC