Industrial Control Systems Dynamic Code Injection

Nidhal BEN ALOUI

Cybersecurity Labs - DCNS Toulon
Nidhal.ben-aloui@dcnsgroup.com

Abstract. Since the day of the virus Stuxnet, the world has discovered the importance of securing Industrial Control Systems, more commonly known as SCADA, and their potential impacts on Critical Infrastructure Protection (CIP). The Stuxnet Malware [14] uses a specific exploit (CVE-2012-3015) which consists of Step 7 Insecure Library Loading. In this paper, we propose to demonstrate how easy it is to make a dynamic code injection in a S7-300 PLC without shutting down or restarting the equipment. We developed a program in C language, using Snap7 library, to push a new Organization Bloc (OB) inside the CPU. We developed a small HMI to illustrate the dynamic modification on the execution flow. In conclusion, we suggest some countermeasures or protection strategies against the dynamic code injection.

Keywords: ICS, Dynamic Code Injection, Cybersecurity, Siemens PLC

1 Introduction

Despite growing awareness of cyber-based attacks on Industrial Control Systems (ICS[1]), many IT security models continues to adhere to the outdated belief that physically isolating systems and “security by obscurity” is enough, says David Emm [1].

Most of the manufacturers of ICS (Siemens, Schneider, Rockwell, etc.) have the same Cybersecurity approach. Many works have been published which introduce cyber attacks or sets of familiar cyber attacks against industrial control systems [3], a lot of them talk about a set of attacks against ICS which use the Modbus communication protocol [18,19] or traditional IT Security [15].

So, we propose in this paper, a proof of concept, to explore and understand the internal operating of PLCs[2] and to see how easy it is to divert these functions [5]. We

[1] **Industrial Control System** (ICS): is a general term that encompasses several types of control systems used in industrial production, including supervisory control and data acquisition (SCADA) systems, Distributed Control Systems (DCS), and other smaller control system configurations such as Programmable Logic Controllers (PLC) often found in the industrial sectors and critical infrastructures.

[2] **Programmable Logic Controller** (PLC): is a digital computer used for automation of typically industrial electromechanical processes, such as control of machinery on factory assembly lines, amusement rides, or light fixtures.

propose to use a S7-300 PLC from Siemens to this paper. But, we point out that all these weaknesses may be valid on the other PLC manufacturers. While these attacks are known to function against Siemens PLC, the vulnerabilities the attacks exploit are general enough to also likely be found in other PLC manufacturers used by ICS.

Firstly we will introduce the main internal operating of the S7-300 PLC [6]. Secondly, we will detail the block oriented approach, the CPU flow execution and the basic programming concepts. The third part on this paper will describe a dynamic code injection attack. And finally, we will finish by a practical demonstration before concluding.

2 Operating principle

2.1 Siemens Protocols

Siemens PLCs, through their communication processors (CP) can communicate in Serial protocol called PROFIBUS or Ethernet protocol called PROFINET (IEC61158, IEC61784-1 and -2).

Profibus protocol uses the R485 serial interface and delivers a deterministic service. It is increasingly replaced by Profinet protocol which is a standard implementation of the TCP/IP protocol. It is stream oriented, though Siemens FC/FB needs to packetize the data stream into blocks.

The data body of Profinet packet is described by S7 Protocol, which is a proprietary Siemens communications protocol. Its Ethernet implementation relies on ISO TCP (RFC1006) which, by design, is block oriented. Each block is named **PUD** (Protocol Data Unit), its maximum length depends on the CPU and is negotiated during the connection (a full reverse engineering work is underway to decode it with a cyber-security vision and create security mechanisms).

S7 Protocol is **Function oriented** or **Command oriented**, i.e. each transmission contains a command query or a reply to it. If the size of a command does not fit in the PDU, then it must be split across more subsequent PDU (in our case, the Snap7 tool is in charge of this task, so we do nothing).

Each command (cf. **Fig. 1**) consists of:

- A header.
- A set of parameters.
- A parameters data.
- A data block.

The first two elements (header and set parameters data) are always present, the other two are optional. One example of valid S7-command is: **Upload** this **BD block** into **CPU**.

The structure of DB block is formatted in a message according to the S7 protocol specifications (known only by Siemens Company).

S7 Protocol, ISO TCP and TCP/IP follow the well-known encapsulation rule: every telegram is the "payload" part of the underlying protocol.

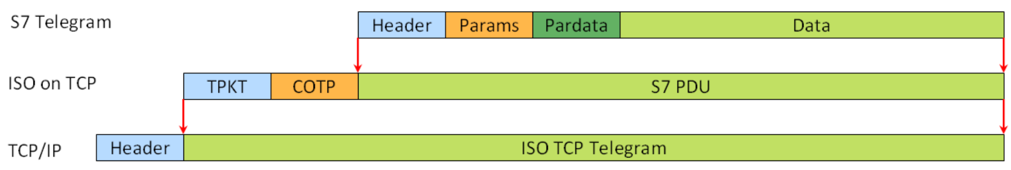


Fig. 1. Protocols Encapsulation.

S7 commands are divided into the following categories:

- Data Read/Write
- Cyclic Data Read/Write
- Directory info
- System Info
- **Blocks move**
- PLC Control
- Date and Time
- Security
- Programming

To ensure a continuous operation of production line, the PLC is designed to receive in real time a new program (locally or through the network) without the need to reboot. Only a CRC check is performed by the CPU. We will exploit this weakness[3] to inject a malicious code into the PLC.

2.2 Block oriented functions

The CPU PLC is composed of seven blocks:

- **OB (Organisation Block):** is the interface between the operating system and the user program. We can consider OB1[4] as the “main function” in C language.
- **FC and SFC (Function or System Function):** provides the specific function of the program (ie. performing a computation). SFCs are functions that can be configured. They are integrated into the operating system of the CPU. SFC features are fixedly defined.
- **FB and SFB (Function Block and System Function Block**): are FC with specific memory area as an instance data block. We can use FB to schedule re-

[3] Since the S7-1500 model and the following firmware version, an authentication session is needed, and in some case you need a password to upload (write into memory area) a new bloc into PLC.

[4] “O.B. one” not like Obi-Wan Kenobi in Star Wars movie (joke).

curring functions (ie. Regulations task). SFB are identical to SFC with their own specific memory area (used to save context or parameters)

- **DB (Data Block):** are user program data areas in which the user data are handled in a structured way.
- **SZL (System Zone List):** memory area containing the diagnostic data and system state. It is in this area that we can find, for example, the level of CPU protection CPU [7].

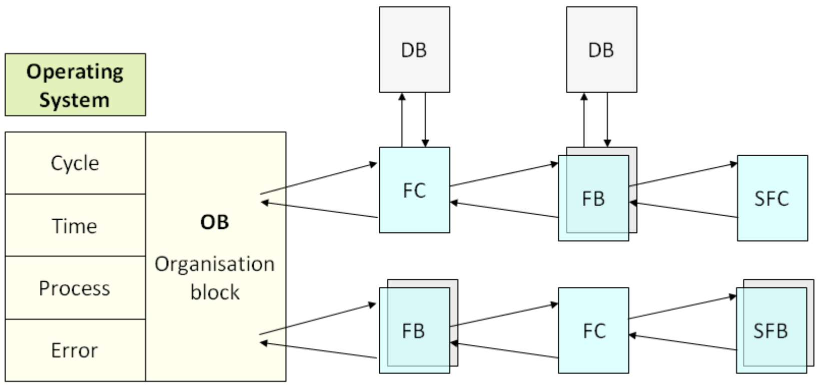


Fig. 2. Call pattern of block

The **Fig. 3** shows the global block structure (OB, FB, DB, etc.).

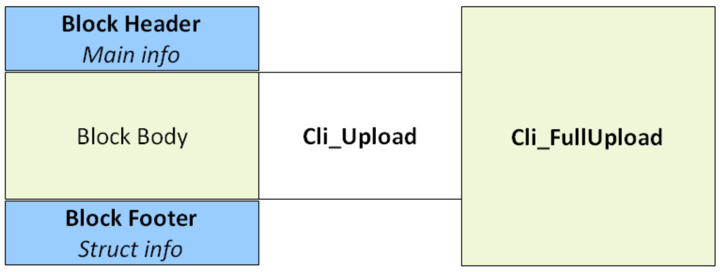


Fig. 3. Global block structure.

- A Header containing main block info such as MC7 size, date and time, etc.
- A Body containing the data.
- A Footer containing info about structure such as number and type of elements of a DB. Additional block info such as author, etc.

To avoid drawing attention, block load order is very important. If a DB calls a FC not yet loaded, this can result in a PLC crash. We should always start by loading the rank N-1 blocks before rank N blocks (cf. **Fig. 2**).

2.3 CPU flow execution

The CPU checks the status of inputs and outputs at each cycle (around a few milliseconds). Specific memory areas are used to store binary data modules: Inputs-Memory Picture (IMP) and Outputs-Memory Picture (OMP). The program (MC7 code) accesses these registers during treatment.

Inputs-Memory Picture (IMP) is in the memory area of the PLC CPU. It contains the logic state of all inputs. IMP is read at the beginning of each cycle.

Outputs-Memory Picture (OMP) contains the output values obtained after processing program. At the end of the cycle, these values are sent to physical outputs of the PLC.

The figure **Fig. 4** summarizes the reading process, the loading and processing of inputs and outputs.

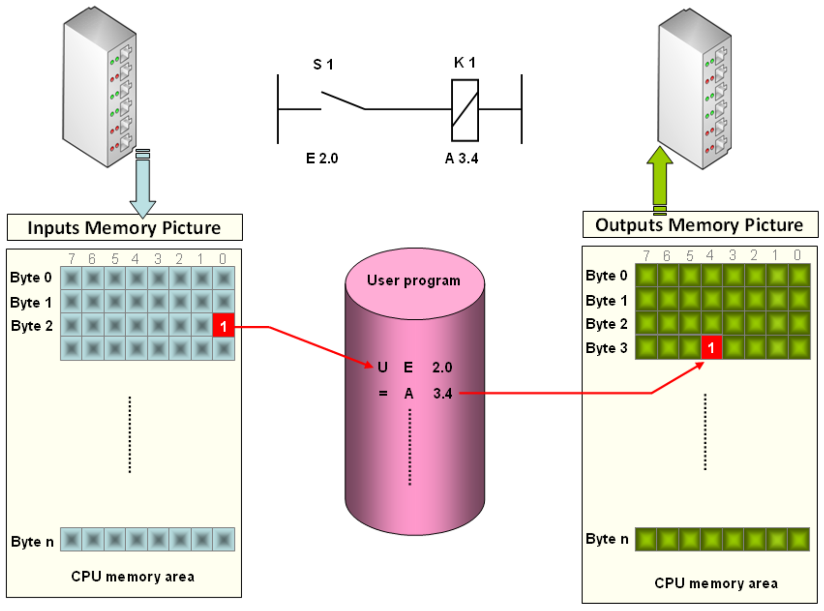
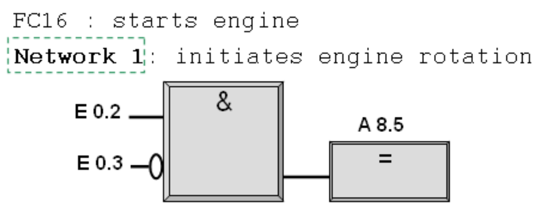


Fig. 4. Memory-pictures Process.

2.4 Programming

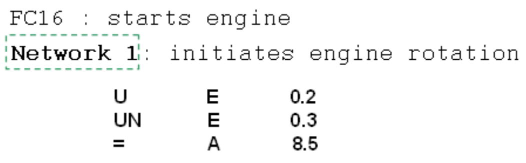
We will use the TIA Portal Software [9] to code a Step 7 program. Siemens offers three different kinds of programming languages to use [8]:

- **LOG** (for logigram): it is a language close to GRAPHSET [11]



```
FC16 : starts engine
Network 1: initiates engine rotation
```

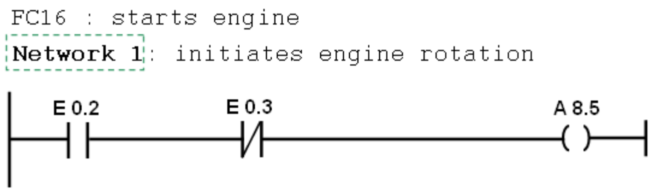
- **LIST** (for lists instructions): it is a language close to ASSEMBLER processor language [12]



```
FC16 : starts engine
Network 1: initiates engine rotation

        U       E       0.2
        UN      E       0.3
        =       A       8.5
```

- **CONT** (for contact): it is a language close to LADDER [10]



```
FC16 : starts engine
Network 1: initiates engine rotation
```

In our demonstration, we will use the **CONT** language to code the corrupted OB1, but you can also use **LIST** or **LOG**.
After coding, the source code is converted automatically to LIST language. Thereby, this first transformation standardizes the code and makes it compatible to all Siemens product ranges. The end build process generates a **MC7 file** which is a binary code machine that the PLC CPU can understand and execute (cf. **Fig. 5**).

Each generated block file is a MC7 file uploaded to the memory using TIA Portal Software. Historically, the small memory size of PLC requires the use of small file sizes (LIST hexadecimal code).

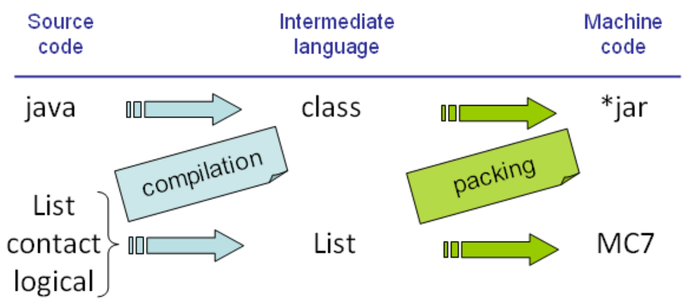


Fig. 5. Analogy between **java** and **step 7** languages build process.

The variables are regrouped into three categories:

- The inputs (I): are used to determine the state of an electrical input of the control (whether a voltage is applied to this input). The result of this query is written in the IMP and uses the time of an execution cycle.
- The outputs (Q): are used by the controller to deliver current. This is where we will connect our voltage regulation card for example. The result of this query is written in the OMP and uses the time of an execution cycle.
- Bit memory (M), corresponding to PLC memory areas (to store a temporary condition, or value).

All these variables must have a type depending on their use. The most common, in our case, is the bit (Boolean). Indeed, we use the digital inputs and outputs (all or nothing so) for our demonstration. Many states will therefore be represented by 0 (if nothing happens) or 1 (there is something going on). However, many other types are available, such as byte (byte), the word (double byte), etc.

These variables must be declared in a static way in source code, that is to say, they are affected a name but also a position (offset) in the memory (for example M2.4 indicates a memento that will be located on the second byte and will start from the 4th bit).

2.5 Loading blocs process

Snap7 is an open source, 32/64 bit, multi-platform Ethernet communication suite for interfacing natively with Siemens S7 PLCs. The new CPUs 1200/1500, the old S7200, the small LOGO 0BA7/0BA8 and SINAMICS Drives are also partially supported. For more details about main Snap 7 features you can go on website of Snap 7 [13].

The Snap 7 library provides two main functions: **`Cli_FullUpload()`** to upload a **full** block from the PLC CPU and **`Cli_Upload()`** to upload **only data** from a data block, depending on your need. After a local modification, we will use the

`Cli_FullUpload()` function to inject a corrupted or modified OB1 throughout the PLC CPU. We recall that the reference position is that of the PLC, we download and upload the date from the PLC (that is why the functions are reversed).

If the real user sends an authentication packet to the PLC, first the device makes a comparison to verify that date and time packet are correct. If this previous verification is ready, the device checks if the password or hash (CRC code), from the user's packet, matches the one inside of the project file stored in the PLC's memory.

If both conditions are true, the device flips a bit and allows a read/write/execute permission to the PLC's memory.

In case of false conditions, the device rejects the request then aborts the TCP connection.

Snap 7 library ensures the good execution of data exchange.

3 Description of the attack : dynamic code injection

In a real production environment we found a complex ICS architecture composed by: PLCs, supervision desktops, sensors, actuators, Control & Command HMI and some other devices connected together through an Ethernet network (PROFINET). It is easy to find a point of connexion on network switch. Sometimes, infecting the desktop of the operator maintenance can gives to the hacker an unauthorised access to the network and therefore to the PLC.

We want to upload a corrupted OB1 to an S7-300 PLC which the IP-address is "**192.168.0.1**".

We need to:

1. Create a Snap7Client.
2. Connect it to the PLC
3. Upload the corrupted OB1
4. Destroy the Client (the disconnection is automatic on destroy).

To do this, we simply include the Snap7 wrapper in our source code and use the Client class as follow:

```
//Upload Full Block OB1 (surely exists in PLC)
//---------------------------------------------------
void FullUploadOB1()
{
  int Size = sizeof(Buffer); //Size is IN/OUT parameters (161
bytes)
  // In input it tells the client the size available
  // In output it tells us how many bytes were uploaded
  int res=Cli_FullUpload(Client, Block_OB, 1, &Buffer, &Size);
  if (Check(res,"Block Upload (OB 1)"))
```

```
    {
      Printf("Dump (%d bytes) :\n",Size);
      Hexdump(&Buffer, Size);
    }
}
```

Parameters

	Type	Value	Description
Client	Native Integer		The handle as return value of Cli_Create(), passed by value
BlockType	integer 32	Block_OB	Type of Block that we need. In our case 0x38 for OB
BlockNum	integer 32	1	The Block number one
pUsrData	Pointer		Address of the user buffer
Size	Pointer to integer 32	64 K	Buffer size available on CPU
		Out	Bytes uploaded from the PLC

After obtaining the original OB1, we open it under TIA Portal then we modify it as we like (by coding the payload part: for example showing a false statements). Industrial control systems commonly use polling techniques to continuously monitor the state of remote process (as was in Natanz nuclear Iranian plant). Polling takes the form of a query transmitted from the PLC to the HMI supervision followed by a response packet transmitted from the HMI supervision to the PLC. The state information is used to provide a human machine interface (HMI) to monitor the process and take requisite control actions based upon process state. An example of paylod consists of modifying the original source code to send false information to the operator.

Finally we build the MC7 file by compiling the code source. Note that a legitimate station of development checks the correct CRC between the header of block and the body of the same block (no integrity problem with the CPU S7-300).

As for the Upload operation, conversely we made the Download of the corrupted OB1 from hacker station to PLC by using the following code source:

```
//Defined the new hexadecimal block in Buffer 2 (arbitrary code
generated by a legitimate TIA Portal development station)
unsigned char Buffer2[] =
  "\x70\x70\x01\x01\x02\x08\x00\x01\x00\x00\x00\x70\x00\x00\x00\x00"
  "\x03\x03\x6b\x68\x2d\x27\x03\xa1\x63\x83\x21\xa7\x00\x1c\x00\x06"
  "\x00\x14\x00\x06\xc0\x00\xd8\x88\x65\x00\x01\x00\x00\x14\x00\x00"
  "\x00\x02\x05\x02\x05\x02\x05\x02\x05\x02\x05\x02\x05\x05\x05\x05"
  "\x05\x05\x05\x0e\x05\x20\x01\x00\x04\x00\x00\x00\x00\x00\x00\x00"
  "\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"
  "\x00\x00\x00\x00\x01\x00\xde\x06\x00\x00\x00\x00\x00\x00\x00\x00";
```

```
//Download Full Block OB1 (including header and footer)
//---------------------------------------------------
void FullDownloadOB1()
{
  int Size = sizeof(Buffer2); //Size issued from FullUploadOB1
function
  int res=Cli_Downpload(Client, 1, &Buffer2, &Size);
  if (Check(res,"Block Download (OB 1)"))
  {
    Printf("Push (%d bytes) :\n",Size);
  }
}
```

Parameters

	Type	Value	Description
Client	Native Integer		The handle as return value of Cli_Create(), passed by value
BlockNum	integer 32	1	The Block number one
pUsrData	Pointer		Address of the user buffer
Size	Pointer to integer 32	122 bytes	Buffer size of OB1 block

3.1 Disabling of security mechanisms

The PLC S7-300 has two security functions described in next paragraphs.

3.1.1 A Password authentication against read/write/execute permissions to the PLC's memory

The PLC CPU uses a hard-coded password to give the users access. D. Beresford [15] has shown that it is possible to read and write data to the PLC's memory even when the password protection is enabled[5]. We have two other possibilities to bypass the password: extracting the hash of password then replaying it again or brute forcing the password offline. To do this, we use a funny SCADA tools composed by five **Python** scripts [.

`s7_password_hash_extractor()` to extract the password hash from the PEData.plf file. You will find the *.plf file into the PLC configuration or in Simatic environment.

`s7_brute_offline()` to crack the password by brute forcing the connection to the PLC. We recall that the number of connections attempts is unlimited. We need only 5 minutes to find the right password.

5 This is particularly true for previous models of Siemens PLC. We do not test it yet on S7-1500 PLC.

At the end, we connect to the PLC and disable the password protection to be able to use the Snap 7 Library without authentication session.

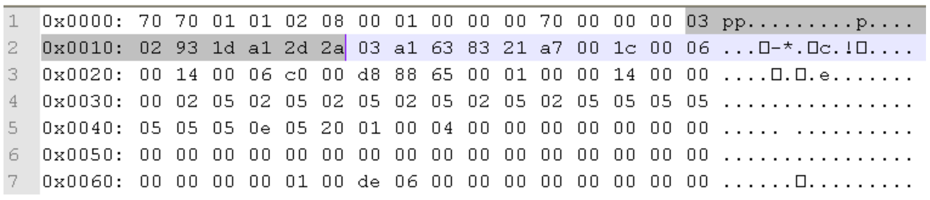
There is another confidential technique which allow to broke the password due to the poorness of cipher algorithm used, but we can not talking about it in this paper[6].

3.1.2 A KNOW-HOW-PROTECTED block

It is possible to set a flag in block properties (set the Know-How-protected variable to true). Thus, the code source of block is "normally" protected (i.e unreadable). But in fact, we can easily bypassing this protection just by modifying from the **16th bytes** to **22nd bytes** in binary MC7 code.

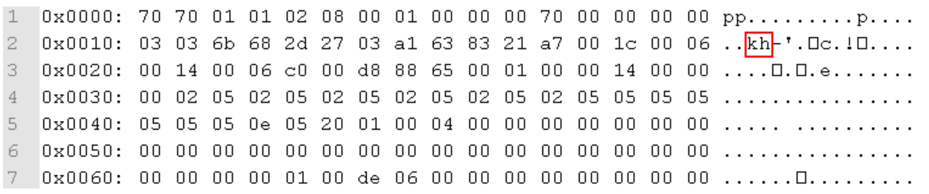
We replace theses bytes (cf. **Fig. 6**) by their equivalent bytes from an original OB1 MC7 file (it is the same hexadecimal value for all unprotected block). Finally, we download again this modified OB1 to the PLC.

Note that the checksum (CRC) of file is calculated just on the body of block. But as the Know-How-protect flag is placed in the header mc7 file, so we don't need computing the new good CRC.



```
0x0000: 70 70 01 01 02 08 00 01 00 00 00 70 00 00 00 03 pp.........p....
0x0010: 02 93 1d a1 2d 2a 03 a1 63 83 21 a7 00 1c 00 06 ...□-*.□c.!□....
0x0020: 00 14 00 06 c0 00 d8 88 65 00 01 00 00 14 00 00 ....□.□.e.......
0x0030: 00 02 05 02 05 02 05 02 05 02 05 02 05 05 05 05 ................
0x0040: 05 05 05 0e 05 20 01 00 04 00 00 00 00 00 00 00 ..... ..........
0x0050: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0x0060: 00 00 00 00 01 00 de 06 00 00 00 00 00 00 00 00 ......□.........
```

Fig. 6. Original OB1 MC7 File



```
0x0000: 70 70 01 01 02 08 00 01 00 00 00 70 00 00 00 00 pp.........p....
0x0010: 03 03 6b 68 2d 27 03 a1 63 83 21 a7 00 1c 00 06 ..kh-'.□c.!□....
0x0020: 00 14 00 06 c0 00 d8 88 65 00 01 00 00 14 00 00 ....□.□.e.......
0x0030: 00 02 05 02 05 02 05 02 05 02 05 02 05 05 05 05 ................
0x0040: 05 05 05 0e 05 20 01 00 04 00 00 00 00 00 00 00 ..... ..........
0x0050: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0x0060: 00 00 00 00 01 00 de 06 00 00 00 00 00 00 00 00 ......□.........
```

Fig. 7. Protected OB1 MC7 File with *Know-How-Protected.*

As showing in the red box of the second capture (**Fig. 7**), we can read two characters **kh** for Know-How encoded file. Theses bytes are used by the CPU (and Simatic framework) to open or not the code source of the block.

[6] Actually we just have done this exploit only on S7-300, so we can not say the same for other later PLC models.

4 Demonstration

In our Cybersecurity Labs we use a real S7-300 PLC with associated Power- module and Inputs/Outputs card to proof our cyber-attack. To simplify the demonstration[7], we will use a virtualized instance of S7-300 PLC. This simulated PLC has the same functionalities as a real PLC.

To do this, we use a **PLCSIM** program (proposed by Siemens Company) and a very friendly tool **NetToPLCsim** S7online [14] to communicate from the LAN to the simulated PLC.

The **Fig. 8** show the network interconnection of demonstration components.

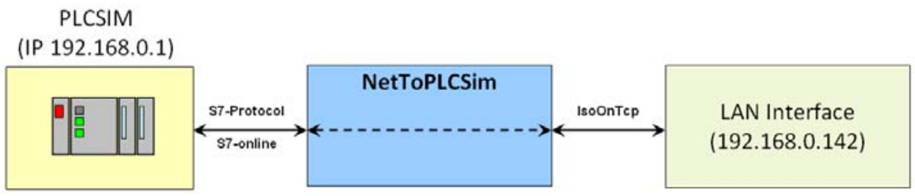


Fig. 8. Simulation with one single PLCSIM-PLC

We will use the following configuration:

S7-300 PN-CPU with IP-address:	**192.168.0.1**
LAN interface IP-address:	**192.168.0.142**
Hack PC IP-address:	**192.168.0.20**

To join the PLCSIM-PLC from the out of development station we just have to connect to 192.168.0.142 IP-address. The tool NetToPLCsim establishes the link between the physical interface network and the simulated instance of the PLC.

The last action is:

```
ssi@(none): ./client 192.168.0.142 0 2
```

This command connects to a PLC with IP-address 192.168.0.142, Rack=0 and slot=2. The program uploads a new corrupted OB1 into the CPU of PLC without stopping it or shutting it down. The MC7 code is injected dynamically through the non-volatile memory. At the next execution cycle, the PLC program will read the new arbitrary code then will execute it. So the attack is successful.

[7] This also reduces the equipment needed the day of conference.

5 Conclusion

In this paper, we saw that it's very easy to load a corrupted code in the PLC memory with the capability sometimes to turn off security flag (depending on model of PLC). The main reasons for insecure Industrial Control System is: CIP has been built from the very beginning without security in mind.

The potential impacts of malicious code injections include interruption process control, interruption of device communication, unauthorized modification of process set points or destroy the actuators (depending of the payload). Typically actuators, such as switches or valves, connected to physical processes are connected to a digital or analog output connected to intelligent electronic device. Each output connects to the cyber system by modeling it as a digital point in a register of the memory. Changing the state of a bit or a function block in PLC memory has an immediate impact on the physical actuator. An attacker who understands a device's implementation specifics including a memory map can introduce a backdoor into the process to alter actuator states.

In order to protect against the dynamic injection arbitrary code through the PLC we can use a probe as SURICATA [20] which is able to decode S7Com/S7Com Plus packets and detecting this kinds of actions. We wrote some specifics SURICATA rules catching S7 Command & Control PDU[8]. We have also the possibility to configure on the S7-300 PLC the "**alarm organization block**" to detect immediately each modification code or downloading blocks. Thus event detection can draw maintenance operator attention to check the PLC code.

Some of the main issues with current systems are that:

- SIL Certification (IEC 61508) doesn't evaluate security,
- modern PLCs are micro-processor-based, programmable systems that are configured with a basic Windows PC,
- major PLC integrate control and safety system using Ethernet communication with open insecure protocols (Profinet, Modbus TCP, OPC.),
- and many PLC communication interface modules run embedded Operating System [16] and Ethernet stack that have known vulnerabilities and default configurations.

For all theses reasons, and in order to successfully defend against any attack in the SCADA we need to implement a specific security requirements oriented industrial control system. To face cyber-criminal activity, security systems should be continually reviewed and reappraised.

[8] You just need to catch the hexadecimal value of uploading block from the PROFINET packet (code function **0x32** offset=**7** and code function **0x1d** offset=**21**).

6 References

1. Five myths of industrial control system security (http://www.scmagazineuk.com/five-myths-of-industrial-control-system-security/article/431387/)
2. Ralph Langner, "To kill a centrifuge. A technical analysis of what Stuxnet's creators tried" to achieve, Hamburg, Munich, November 2013.
3. Thomas H. Morris & Wei Gao, "*Industrial Control System Cyber Attacks*", 1st International Symposium for ICS & SCADA Cyber Security Research 2013 (ICS-CSR 2013), Leicester, UK, 16-17 September 2013.
4. https://en.wikipedia.org/wiki/Industrial_control_system
5. Siemens Simatic S7-300 PLC Remote Memory Viewer (https://exploit-db.com/exploits/19831/)
6. SIMATIC. *System Software for S7-300 400 System and Standard Function*, March 2006
7. SIMATIC, *Structure and memory CPU used*, January 2013.
8. Karl-Heinz John and Michael Tiegelkamp. "*IEC 61131-3: programming industrial automation systems: concepts and programming langugages, requirements for programming systems, decision-making aids"*. Springer Science & Business Media, 2010.
9. SIEMENS. "*Initiate programming with STEP7 API"*, 2005.
10. SIMATIC S7. *CONT language for SIMATIC S7-300/400 programming blocs*, Siemens.
11. SIMATIC S7. *LOG language for SIMATIC S7-300/400 manual reference*, Siemens.
12. SIMATIC S7. *LIST language for SIMATIC S7-300/400 manuel reference*, Siemens.
13. Davide Nardella, *SNAP7. Http://snap7.sourceforge.net/*
14. Antoine Mignon and David Boucart, "*Siemens S7 protocol communication analysis*", August 2013.
15. Dillon Beresford, "*Black hat. In Siemens Simatic S7 PLC Exploitation*", NSS Labs, 2011.
16. Dillon Beresford, *"Exploiting Siemens Simatic S7 PLCs"*, July 2011.
17. https://github.com/atimorin/scada-tools/blob/master/s7_password_hashes_extractor.py
18. Modbus application protocol specification V1.1b3 (www.modbus.org), April 26, 2012.
19. Modicon Modbus Protocol Reference Guide - PI–MBUS–300 Rev. J.
20. David Diallo and Mathieu Feuillet, "*Industrial Systems Detection : Suricata and Modbus case*", C&ESAR Conference, France, Novembre 2014.
21. http://nettoplcsim.sourceforge.net/