PLC-Blaster: A Worm Living Solely in the PLC

Ralf Spenneberg, Maik Brüggemann, Hendrik Schwartke

[1]OpenSource Security Ralf Spenneberg

info@os-s.de

Abstract. *Industrial processes are controlled by programmable logic controllers (PLC). Many PLCs sold today are equipped with Ethernet ports and can communicate using IP. Based on the Siemens SIMATIC S7-1200 we will demonstrate a worm. This worm does not require any additional PCs to proliferate. The worm lives and runs only on the PLC. The worm scans the network for new targets (PLCs), attacks these targets and replicates itself onto the found targets. The original main program running on the target is not modified. Once infected the worm on the target again starts the scan. We will analyze the impact of the worm on the target and possible mitigation techniques.*

1. Introduction

IT systems are critical components in modern industrial processes. These processes would not be possible without modern communication networks. Unfortunately with the adoption of modern IT systems and communication networks in industrial systems the users are exposed to attacks long known in the IT world. IT hacking attacks may harm industrial systems in several ways. They can cause outages and high financial losses but at the same time may have negative on the health and life of their environment. The efficiency of these attacks has been demonstrated by Stuxnet [1]. Programmable logic controllers of Siemens were modified to hinder the Iran in the enrichment of fissile material. The worm spread amoung the PCs of the enrichment plant by exploiting vulnerabilities in the Microsoft Windows operating system. The software of the PLCs was modified in such a way that the centrifuges used for the enrichment process were destroyed. The worm required a personal computer to spread and attacked the PLCs from the PC. This paper will demonstrate a worm which spreads only among the PLCs themselves. No PC is required. The worm may be introduced into the plant using a already manipulated PLC. The worm then spreads to further PLCs by replicating itself and modifing the target PLCs to execute the worm in addition to the already installed user program.. This paper is based on the Siemens SIMATIC S7-1200v3. The worm was written in *Structured Text* (ST), one of the languages used to develop PLC software.

2. Related Work

2015 at BlackHat USA 2015 Klick et. al. demonstrated malware running on a PLC [2]. They implemented a proxy using the communication features of the PLC. We are using the same communication features to implement the protocol used to transfer the worm program. By using this protocol the worm may spread directly from one PLC to another PLC. The worm does not require any support by further systems. Instead of using the already well known SIMATIC S7-300 we based our work on the new S7-1200v3. The protocol used by these PLCs differs from the older models. This paper will document the new protocol as well.

3. PLC Architecture

PLCs use a simple architecture. They are based on a central processing module (CPU) and further modules supporting digital inputs and outputs. The CPU executes the operating system of the PLC and runs the user program. Additionally the CPU is responsible for the communication with additional devices and manages the process image.

The process image stores the state of all inputs and outputs. The user program operates on the process image rather than on the physical inputs and outputs. The user program is run in cycles. The process image is refreshed by the CPU at the beginning and end of each cycle. The maximum limit for a cycle is the cycle time. If this limit is violated the PLC stops the user program and throws an exception.

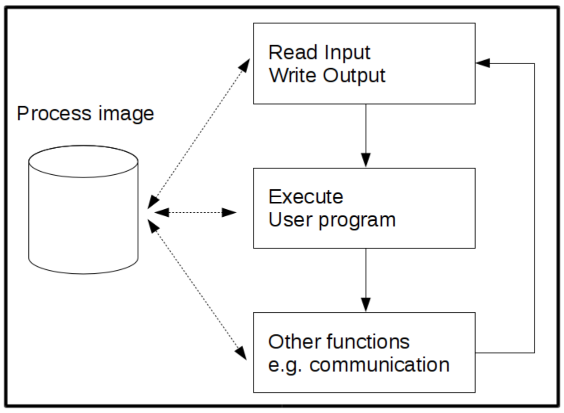


Figure 1. Zyklus eines PLCs

The user program is structered by *Program Organization Units* (POU). These units contain the instructions to control the PLC and thus the industrial process. Siemens SIMATIC S7-1200 support the following POUs:

- *Organisation block* (OB): Main entry into the user program
- *Data block* (DB): Global memory
- *Function* (FC): Function
- *Function block* (FB): Function with persistent local memory

Additionally to the userdefined POUs several functions provided by Siemens are available. This paper uses the system POUs *TCON* and *TDISCON*. Using these POUs the PLC may initiate and destroy TCP connections to arbitrary systems. Buffers may be send and received on these connections with *TRCV* und *TSEND*.

4. Computer Worms

Computer worms have been around since 1988 [3] and are a well known species of malware. Several different flavours exist but they all have the same basic structure [4]. Every worm attack can be structured in the following phases: Detection of possible targets, Spreading to the target, Execution on the target and a malware function. On PLCs a worm must support these functions as well. This paper will demonstrate the implementation of each required component.

5. Implementing a Worm on the S7 1200

5.1. Architecture

The worm was written like any other worm software with some constraints. During the development the specific PLC constraints must be met. Especially the maximum cycle limit needs to be kept. The worm has to interrupt its execution every few milliseconds. The execution may be continued during the next cycle. To meet these requirements the worm was designed using a state machine. The current state is stored in a global variable. At the start of each cycle the appropiate code in the worm is called. Thus the maximum cycle time is never violated.

The sequence of the steps are shown in figure 2. The worm starts by initiating a connection to a probable target. Once the connection is established the worm first checks whether the target is already infected. If no infection is detected the worm will stop the execution of the user program on the target to enable the transfer of its own code. The worm then copies itself to the target and subsequently starts the target PLC again. The worm then tests the next probable target.

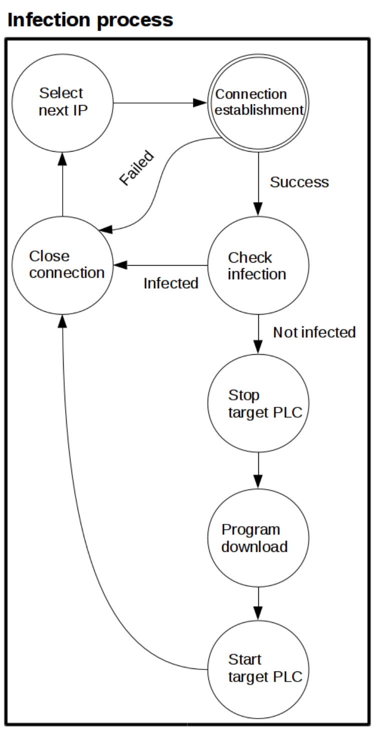


Figure 2. Execution sequence of the worm

5.2. Target Detection

The worm starts by scanning for probable targets. Siemens SIMATIC PLCs may be identified by the port 102/tcp. This port can only be closed by an external firewall. No other common service uses this port.

The S7-1200 manages TCP connections using the POU *TCON*. The usage of this POU is shown in listing 3 on line 3. An arbitrary IP address and port are passed on line 9. Once the POU is called, the PLC will attempt to establish the connection. This happens asynchronously. In later cycles the current state of the connection can be verified. The return value *DONE* (Zeile 5) signals whether the connection was established. If true, the infection will continue. If the IP address and port is unreachable no error is triggered. The timeout needs to be implemented by the worm by incrementing a counter each cycle.

```
1  IF "data".con_state = 10 THEN
2  
3    "TCON_DB"(REQ:="data".action,
4              ID:=1,
5              DONE=>"data".con_done,
6              BUSY=>"data".con_busy,
7              ERROR=>"data".con_error,
8              STATUS=>"data".con_status,
9              CONNECT:="data".con_param);
10 
11   IF "data".con_done = True THEN
12     // connection open
13     "data".con_state := 20;
14   ELSE
15     // connection not open
16     "data".con_timeout := "data".con_timeout + 1;
17     // connection timeout?
18     IF "data".con_timeout > 200 THEN
19       "data".con_state := 0;
20     END_IF;
21   END_IF;
22 
23 GOTO CYCLE_END;
24 END_IF;
```

Figure 3. Target Detection using SCL

If no connection was established after 200 cycles the worm will continue in listing 4. Although no connection was established the POU *TDISCON* has to be called to free the resources for the next connections. In line 13 the IP adress is incremented. Thus a full /24 subnet is scanned for open ports 102/tcp.

```
 1 IF "data".con_state = 0 THEN
 2
 3   "TDISCON_DB"(REQ:="data".action,
 4                ID:=1,
 5                DONE=>"data".con_done,
 6                BUSY=>"data".con_busy,
 7                ERROR=>"data".con_error,
 8                STATUS=>"data".con_status);
 9
10   IF "data".con_error = True OR
11      "data".con_done = True
12   THEN
13     "data".con_param.REM_STADDR[4] := \
14           ("data".con_param.REM_STADDR[4] + 1) MOD 255;
15     "data".con_timeout := 0;
16     "data".con_state := 10;
17   END_IF;
18
19   GOTO CYCLE_END;
20 END_IF;
```

Figure 4. Target detection in SCL

5.3. Infection

During the infection phase the worm replicates itself to the target PLC. Normally software is transferred to the PLC using the Siemens TIA-Portal. The worm mimics the TIA-Portal and implements the proprietary Siemens protocol. While we analyzed the protocol on our own, the Wireshark plugin of Thomas Wiens can interpret the protocol as well [5].

5.3.1. Software Transfer Protocol

The proprietary protocol will be called S7CommPlus for the rest of this paper. It is a binary protocol which utilizes both TPKT [6] and ISO8073 [7]. Typically both of these protocols use port 102/tcp.

The main features of S7CommPlus are:

- Configuration of the PLC
- Starting and stopping the PLC
- Reading and writing process variables
- Program transfer (Up-/Download)
- Debugging
- Providing debugging information
- Alerting

5.3.2. Messages

Every message used by S7CommPlus has a similar structure. Figure 5 presents the first message in a connection. The TIA portal sends this message to initiate a connection. The general structure will be explained based on this message. The first two fields represent the TPKT and ISO8073 protocol. Their contents is explained in the respective documentation. The following byte `0x72` represents the start of the S7CommPlus message. A version number distinguishes between different variants of the protocol. The length field does not take into account the frame boundary. If the frame boundary is missing, further data is following in additional messages. Following the length field the type field is transferred. The subtype further specifies the messages. The sequence number is incremented for each message. Additional data is transferred in separate attribute blocks.

```
03 00 00 e1 02 f0 80 72  01 00 d2 31 00 00 04 ca  .......r ...1....
00 00 00 02 00 00 01 20  36 00 00 01 1d 00 04 00  .......  6.......
00 00 00 00 a1 00 00 00  d3 82 1f 00 00 a3 81 69  ........ .......i
00 15 16 53 65 72 76 65  72 53 65 73 73 69 6f 6e  ...Serve rSession
5f 36 42 36 31 38 32 46  31 a3 82 21 00 15 2c 31  _6B6182F 1..!..,1
3a 3a 3a 36 2e 30 3a 3a  54 43 50 2f 49 50 20 2d  :::6.0:: TCP/IP -
3e 20 49 6e 74 65 6c 28  52 29 20 50 52 4f 2f 31  > Intel( R) PRO/1
30 30 30 20 4d 54 20 44  2e 2e 2e a3 82 28 00 15  000 MT D .....(..
00 a3 82 29 00 15 00 a3  82 2a 00 15 11 4d 41 49  ...).... .*...MAI
4b 2d 50 43 5f 33 32 39  31 38 39 35 31 35 a3 82  K-PC_329 189515..
2b 00 04 01 a3 82 2c 00  12 00 2d c6 c0 a3 82 2d  +.....,. ..-....-
00 15 00 a1 00 00 00 d3  81 7f 00 00 a3 81 69 00  ........ ......i.
15 15 53 75 62 73 63 72  69 70 74 69 6f 6e 43 6f  ..Subscr iptionCo
6e 74 61 69 6e 65 72 a2  a2 00 00 00 00 72 01 00  ntainer. .....r..
00
```

TPKT, ISO8073, Magic Byte, Version, Length, Type, Reserved, Subtype,
Sequence number,
Attribute blocks, Frame boundary,
Unknown

Figure 5. S7CommPlus message structure

5.3.3. Attribute blocks

The actual data is structured using attribute blocks. Figure 6 shows the first attribute block of the preceding example. Each attribute block starts with the byte `0xA3`. This block contains a string. The actual string starts with its length and the value of the string.

```
                                                 a3 81 69  ........ .......i
00 15 16 53 65 72 76 65  72 53 65 73 73 69 6f 6e  ...Serve rSession
5f 36 42 36 31 38 32 46  31                        _6B6182F 1
```

Attribute block start, Attribute ID, Format?, datatype, Length, Value,

Figure 6. Attribute block

5.3.4. Coding numbers

Within the attribute blocks numbers are coded in a special way. The number may have a variable length. The first bit of each byte encodes whether further bytes follow. Figure 7 explains the decoding of the attribute id and the length field of the last example. If values are stored within the attribute block this encoding is not used.

Byte following

$$81\ 69_{(16)} = 10000001\ 01101001_{(2)} \rightarrow 233_{(10)}$$

$$16_{(16)} = 00010110_{(2)} \rightarrow 22_{(10)}$$

Figure 7. Encoding of numbers

5.3.5. Anti Replay Mechanism

The S7CommPlus protocol detects replay attacks. To detect the replay attack the PLC sends a random byte in the 25th byte of its response message (figure 8). The value of the random byte ranges between `0x06` and `0x7f`. This is the anti replay challenge.

The TIA portal needs to base the 24th and 29th byte in its response based on the challenge. The anti replay byte is calculated by the following formula:

```
antireplaybyte = challenge + 0x80
```

```
03 00 00 89 02 f0 80 72  01 00 7a 32 00 00 04 ca  .......r ..z2....
00 00 00 02 36 11 02 87  22 87 3d a1 00 00 01 20  ....6... ".=....
82 1f 00 00 a3 81 69 00  15 00 a3 82 32 00 17 00  ......i. ....2...
00 01 3a 82 3b 00 04 82  00 82 3c 00 04 81 40 82  ..:.;... ..<...@.
3d 00 04 84 80 c0 40 82  3e 00 04 84 80 c0 40 82  =.....@. >.....@.
3f 00 15 1b 31 3b 36 45  53 37 20 32 31 32 2d 31  ?...1;6E S7 212-1
42 45 33 31 2d 30 58 42  30 20 3b 56 33 2e 30 82  BE31-0XB 0 ;V3.0.
40 00 15 05 32 3b 35 34  34 82 41 00 03 00 03 00  @...2;54 4.A.....
a2 00 00 00 00 72 01 00  00
```

Challenge, Attribute block

Figure 8. Anti replay mechanim

All further messages sent by the TIA portal to the S7-1200 need to use the anti replay byte in their 24th byte. The grey attribute block needs to be mirrored as well.

```
03 00 00 8c 02 f0 80 72  02 00 7d 31 00 00 05 42 .......r ..}1...B
00 00 00 03 00 00 03 a2  34 00 00 03 a2 01 01 82 ........ 4.......
32 01 00 17 00 00 01 3a  82 3b 00 04 82 00 82 3c 2......: .;.....<
00 04 81 40 82 3d 00 04  00 82 3e 00 04 84 80 c0 ...@.=.. ..>.....
40 82 3f 00 15 00 82 40  00 15 1a 31 3b 36 45 53 @.?....@ ...1;6ES
37 20 32 31 32 2d 31 42  45 33 31 2d 30 58 42 30 7 212-1B E31-0XB0
3b 56 33 2e 30 82 41 00  03 00 00 00 00 00 00 04 ;V3.0.A. ........
e8 89 69 00 12 00 00 00  00 89 6a 00 13 00 89 6b ..i..... ..j....k
00 04 00 00 00 00 00 00  72 02 00 00
```

🟥 Anti replay byte, ▨ Attribute block

Figure 9. Anti replay Mechanism

5.3.6. Transfer of the program

To transfer the user program a specific message is used (Figure 10). Each message transfers one POU. The POU type distinguishes between the different POU variants. The block number specifies the place im memory on the PLC.

```
03 00 04 00 02 f0 00 72  02 05 a9 31 00 00 04 ca .......r ...1....
00 00 00 1d 00 00 03 a2  34 00 00 00 03 00 04 00 ........ 4.......
00 00 00 00 a1 8a 32 00  01 94 57 20 00 a3 81 69 ......2. ..W ...i
00 15 04 4d 61 69 6e
...
```

🟥 POU type, 🟦 Block number, ▨ Attribute block

Figure 10. Transferring the user program

Several attribute blocks follow the message header. Additionally to the actual byte code meta information is stored on the S7. This meta information specifies the required memory, the creation date, block number, used language, source code and protection features. The TIA portal may use this information to verify the code.

5.3.7. Determining the required messages

During the transfer of the user program several messages are exchanged which are not mandatory for the process. These irrelevant messages would increase the memory required by the worm and are therefore skipped.

The figure 11 displays the required messages for a successful infection. The communication is initiated. To avoid repeated infections the worm first tests the target and tries to download a copy of itself. The upload of additional code is only possible after halting the PLC. Then the program is transferred. At the end the PLC is started again.

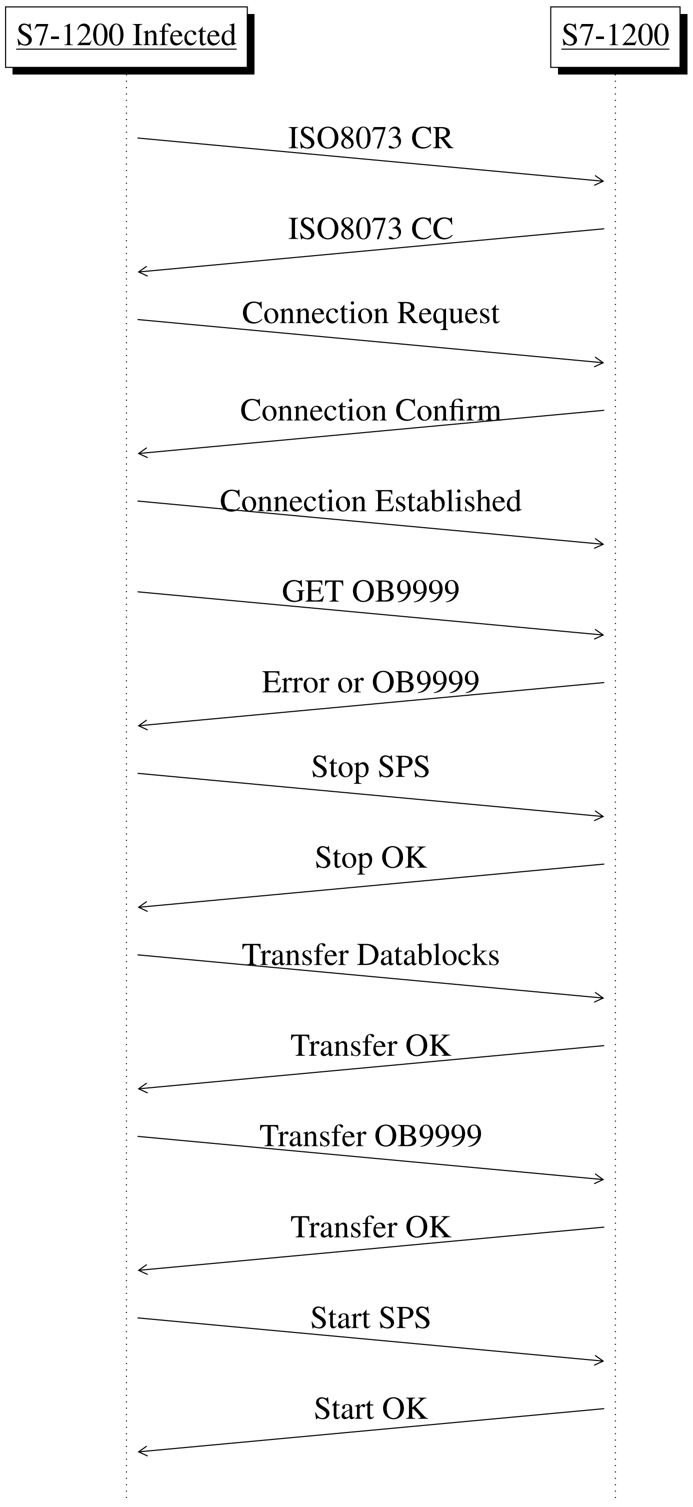


Figure 11. Messages exchanged during infection

5.3.8. Implementation

Based on the analysis of the protocol the transfer of a program may be recorded, modified and replayed to a PLC. All required messages are known. To store the messages within the worm a static DB POU is used. Additional DBs are used to store temporary variables and the send/receive buffer.

The static DB has to store all messages required for the infection. This DB block cannot be generated using the TIA portal but needs to be coded manually.

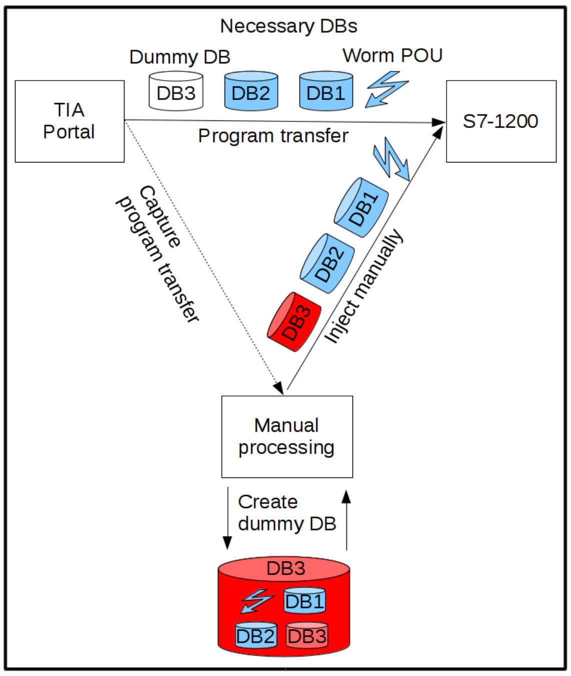


Figure 12. Manual generation of the DB

6. Starting the worm

The transferred code adds to the user program already running on the target PLC. An additional OB and the required DBs are added. The original code on the target is untouched. The OB is automatically detected by the PLC and executed.

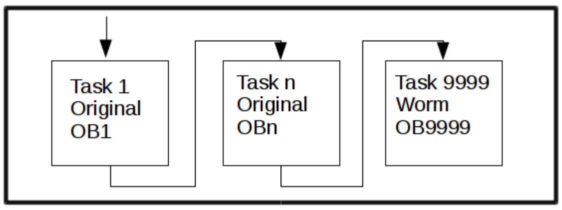


Figure 13. The worm is run as normal code

7. Malware Function

While the worm is already functional we implemented different malware function to demonstrate the possible impact.

7.1. Command & Control Server

Our worm will contact a command & control server. The communication is based on TCP. Different features within the worm may be triggered by the C&C server.

7.2. Socks4 Proxy

Our worm serves a Socks4 proxy. Once the worm connected to the C&C server, arbitrary connections to additional clients within the network of the PLC may be initiated using the embedded Socks4 proxy.

7.3. Denial of Service

The execution on the PLC can be stopped by violation the cycle time limit. The worm implements an endless loop triggering an error condition within the PLC with the impact of a DoS.

7.4. Manipulating Outputs

The worm may manipulate any outputs of the PLC. Using the POU *POKE* any value within the process image may be modified.

8. Detection of the worm, Persistance and Resources

8.1. Detection of the worm

8.1.1. TIA portal

The TIA portal may verify the user program on the PLC and can detect modified and added POUs (Figure 14). The red outlined area points at the POUs used by the worm. The analysis of the POUs is not possible though, because the TIA portal only analyzes the XML sourcecode. Additionally, by exploiting a bug in the TIA portal the worm may crash the application.

8.1.2. Stopping the PLC

The PLC is stopped during the infection for about 10 seconds. The original user program does not run during this time. This interruption may be noticed and is logged within the PLC.

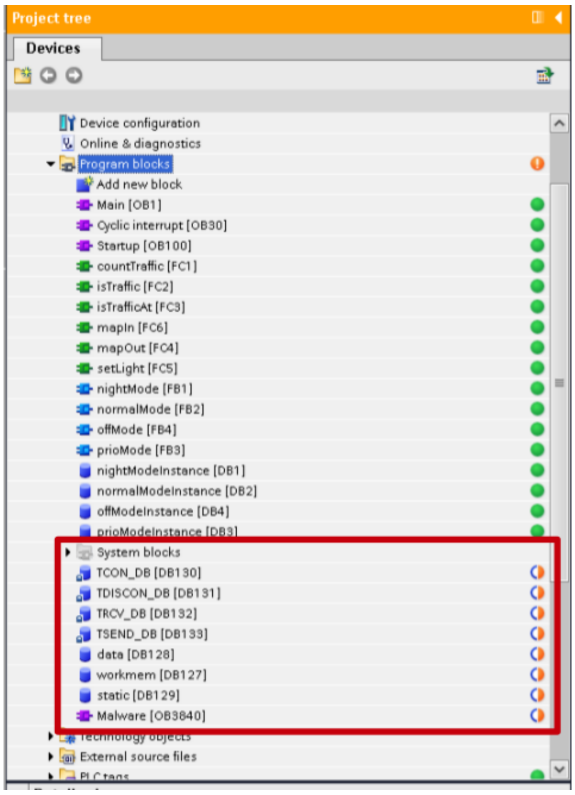
8.1.3. Network traffic

The worm generates unusual network traffic within an ICS environment. During the scan and infection phase many suspicous packets are sent.

8.2. Persistence

8.2.1. Restart/Reboot

The worm is stored on the PLC. It is part of the user program and will survive a Restart and even the removal of power.


```
Project tree
Devices

    Device configuration
    Online & diagnostics
  ▾ Program blocks
      Add new block
      Main [OB1]
      Cyclic interrupt [OB30]
      Startup [OB100]
      countTraffic [FC1]
      isTraffic [FC2]
      isTrafficAt [FC3]
      mapIn [FC6]
      mapOut [FC4]
      setLight [FC5]
      nightMode [FB1]
      normalMode [FB2]
      offMode [FB4]
      prioMode [FB3]
      nightModeInstance [DB1]
      normalModeInstance [DB2]
      offModeInstance [DB4]
      prioModeInstance [DB3]
    ▸ System blocks
      TCON_DB [DB130]
      TDISCON_DB [DB131]
      TRCV_DB [DB132]
      TSEND_DB [DB133]
      data [DB128]
      workmem [DB127]
      static [DB129]
      Malware [OB3840]
  ▸ Technology objects
  ▸ External source files
  ▸ PLC tags
```

Figure 14. TIA portal exposes the worm

8.2.2. Factory Reset

The TIA portal may trigger a factory reset of the PLC. All settings and the user program including the worm are cleared from the device.

8.2.3. Program transfer

Our worm is stored in OB9999. If this POU is overwritten the worm is removed from the PLC.

8.3. Resources

8.3.1. Cycle time

The maximum cycle limit is a hard quota. By default the limit is 150ms. The worm must not violate this limit. We measured the cycle time of a PLC without any user program. The cycle time is 0 ms. We then infected the S7 using our worm and measured again. The maximum measured cycle time is 7ms. This represents 4.7% of the limit.

8.3.2. Memory

The worm requires 38,5KB RAM for storing the worm. 9,0KB (23,3%) are needed by the malware functions. Additionally 216,6KB flash is needed. Table 1 displays the available memory of the different models.

Model	RAM	Flash
S7-1211	50KB (77%)	1MB (21%)
S7-1212	75KB (51%)	1MB (21%)
S7-1214	100KB (38%)	4MB (5%)
S7-1215	125KB (30%)	4MB (5%)
S7-1217	150KB (25%)	4MB (5%)

Table 1. Memory footprint

9. Protection Features

The PLC S7-1200v3 offers three different protection features. We will analyze each of these features and will evaluate whether the feature will protect the PLC against the worm infection. This analysis is based on TIA portal V11 SP2 Update 5 and the S7-1200 using firmware 3.0.2.

9.1. Knowhow Protection

The knowhow protection protects the user program from unauthorized access. Using a password the unauthorized access and modification of a POU is prohibited.

The knowhow protection is implemented using an attribute block. This block is written on the PLC during the program transfer. The block is displayed in figure 15. The flag toggles the knowhow protection. The stored password hash is generated based on the password P using the following formula:

```
H = sha-1(encode_utf-16le(P))
```

```
A3 93 5C 00 17 00 00 0D 77 9A 78 00 0B 00 01 9A    .......w.x.....
79 10 02 14 E8 F9 7F BA 91 04 D1 EA 50 47 94 8E    y..........PG..
6D FB 67 FA CD 9F 5B 73 00                         m.g...[s.
```

Start of the attribute block, datatype, container ID, Attribute ID (container knowhow), datatype, attribut-ID (flag), datatype, flag Attribut-ID (passwort hash), datatype, length, passwort hash,

Figure 15. Attribute block: knowhow protection

The TIA portal evaluates the attribute block. If the flag is set the TIA portal prohibits read and write access to the corresponding program block without the correct password. The password is checked using the hash.

The XML source code on PLC is encrypted using AES128-CBC. This prevents access to the code. Figure 16 shows the encrypted source code.

9.1.1. Vulnerabilities

Missing integrity protection

The block may be read and modified despite the knowhow protection. The protection is

```
<NetworkContainer>
 <Network Lang="SCL" ProgrammingContext="Plain"
       Mnemonic="International" RefID="1">
  <EncryptedData Type="http://www.w3.org/2001/04/xmlenc#Element"
         xmlns="http://www.w3.org/2001/04/xmlenc#">
   <EncryptionMethod
            Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc" />
    <CipherData>
     <CipherValue>
         AQIDAQIDAQIDAQIDAQIDAS/acA/s+L3CoQYaeAQMOq ...
    </CipherValue>
   </CipherData>
  </EncryptedData>
...
```

Figure 16. Encrypted source code

implemented within the TIA portal and not within the PLC. Using self-written tools to read and write blocks on the PLC any access is granted. Even the knowhow protection flag may be reset resulting in full access using the TIA portal.

AES-key may be derived
The key for the AES encryption may be derived from the password hash. The password hash may be read using self-written software. The key is calculated using the following formula:

```
K = truncate128Bit(H) XOR M
```

with M :

```
M = 0x28,0x6f,0x76,0x5c,0x6e,0x3b,0x1e,0x4c,
    0xd0,0x8e,0x42,0x31,0x43,0x7b,0x8e,0xbf
```

Fixed by the vendor in SSA-833048[1].

9.1.2. Evaluation of the protection in the context of the worm

This feature does not protect against the worm attack.

9.2. Copy protection

The copy protection prohibits the duplication of the user program to a second PLC. The serial number of the target PLC is stored within the user program preventing the TIA portal from transferring the user program to additional PLCs. The serial number is stored in a separate attribute block.

[1]http://www.siemens.com/cert/pool/cert/siemens_security_advisory_ssa-833048.pdf

9.2.1. Vulnerabilities

Missing integrity protection
The integrity of the attribute block is not protected. The stored serial number may be modified or removed. The PLC does not check the serial number itself. The protection is only implemented within the TIA portal.

Fixed by the vendor in SSA-833048[2].

9.2.2. Evaluation of the protection in the context of the worm

This feature does not protect against the worm attack.

9.3. Access protection

The access protection prevents access to the PLC using the S7CommPlus protocol without a password. Three different protection levels are available. Table 2 lists the different levels.

Function	**None**	**Write**	**Read/Write**
Start/Stop CPU	yes	no	no
Write Program	yes	no	no
Read Program	yes	yes	no
Manipulate memory/output	yes	yes	yes
Read identification	yes	yes	yes
Setting IP address	yes	yes	yes
Setting date	yes	no	no
Factory reset	yes	no	no

Table 2. Access based on the protection level

The authenticaton uses a challenge response mechanism [8].

9.3.1. Evaluation of the protection in the context of the worm

The access protection can protect the PLC against the worm attack. The write protection prevents anybody from modifying the code on the PLC. The used challenge response authentication is probably secure. If the used password is not known to the worm the worm may not infect the PLC. By default the access protection ist turned off.

10. Conclusion

In this paper we demonstrate the feasibility of a PLC worm. Such a worm represents a new threat to industrial networks. Traditionally such networks are well protected against attacks from the outside. By introducing a PLC already infected with the worm the PLC is the origin of the attack and not just the target. Infected PLCs may be distributed by

[2] `http://www.siemens.com/cert/pool/cert/siemens_security_advisory_ssa-833048.pdf`

a supplier of an industrial component or during the transport of such a component. The worm may then spread internally and does not require any standard PCs or servers. It will therefore not be detected by any antivirus product. Furthermore the plant operator has very few options to detect the malware on the PLCs.

References

[1] Eric Chien Nicolas Falliere, Liam O Murchu. W32.Stuxnet Dossier. Last accessed `https://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_stuxnet_dossier.pdf` on 06.02.2016, 2011.

[2] Johannes Klick u.a. Internet-facing PLCs - A New Back Orifice. `https://www.blackhat.com/docs/us-15/materials/us-15-Klick-Internet-Facing-PLCs-A-New-Back-Orifice-wp.pdf`, 2015.

[3] pressetext.deutschland. Malware-Jubiläum: 20 Jahre Internet-Würmer. `http://www.pressetext.com/news/20081101001`, 2008.

[4] Nicholas Weaver u.a. A Taxonomy of Computer Worms. `https://www1.icsi.berkeley.edu/~nweaver/papers/2003-taxonomy.pdf`, 2003.

[5] thomas_v2. S7comm Wireshark dissector plugin. `http://sourceforge.net/projects/s7commwireshark/files/`.

[6] Dwight E. Cass Marshall T. Rose. ISO Transport Service on top of the TCP. Last accessed `https://tools.ietf.org/html/rfc1006` on 21.02.2016, 1987.

[7] International Organization for Standardization. Connection oriented transport protocol. *ISO Std. 8073*, 1988.

[8] SCADAStrangeLove. S4x13 Releases: S7 password offline bruteforce tool. `http://scadastrangelove.blogspot.de/2013/01/s7brut.html`, 2013.