Supervising the Supervisor

Reversing Proprietary SCADA Tech

Jean-Baptiste Bédrune
jbbedrune@quarkslab.com
Alexandre Gazet
agazet@quarkslab.com
Florent Monjalet
fmonjalet@quarkslab.com

Plan

Introduction

Us

- Jean-Baptiste Bedrune
- Alexandre Gazet
- Florent Monjalet
- Security researchers at Quarkslab

Quarkslab

- Security R&D and services
- Software editor

Study

- 3 - 4 months

Plan

Critical Systems

Critical systems

- Transportation, energy, financial systems...
- Every system depend on some critical infrastructure
- Consequences of a malfunction
- Interdependencies

Industrial systems

- Water distribution
- Nuclear plant
- Access control
- Production chains

"Identifying, Understanding, and Analyzing Critical Infrastructures Interdependencies", IEEE Control Systems Magazine

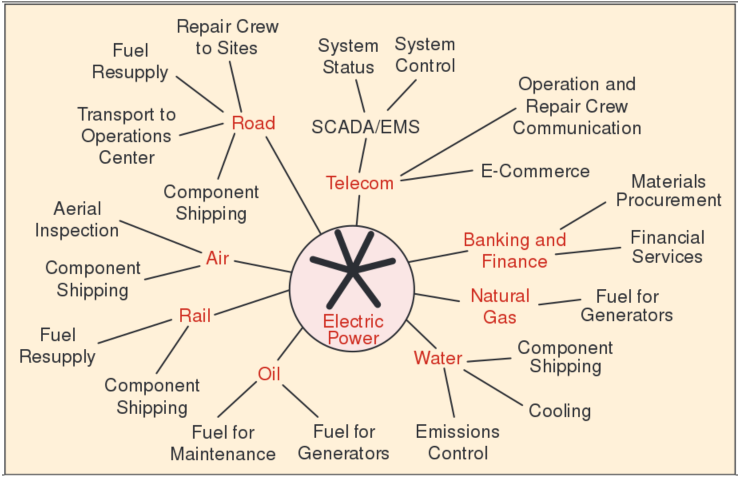


Figure: Examples of electric power infrastructure dependencies

So what is an Industrial System?

Industrial Control System (ICS)

Computer networks that control a physical process.

Supervisory Control and Data Acquisition (SCADA)

Part of an ICS that directly controls and monitors the physical process (sub-part of an ICS).

SCADA

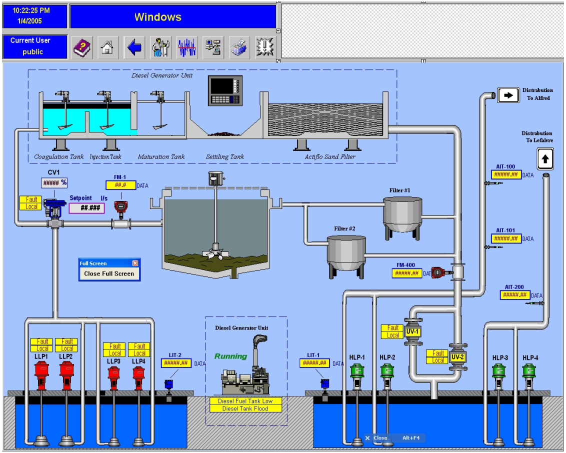


Figure: A SCADA HMI Example *(fastweb.it)*

Components

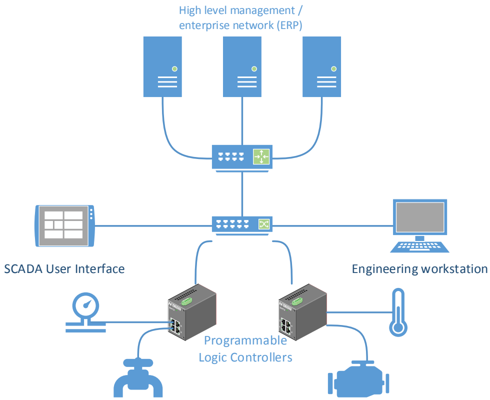


Figure: ICS Components

A Concrete Example

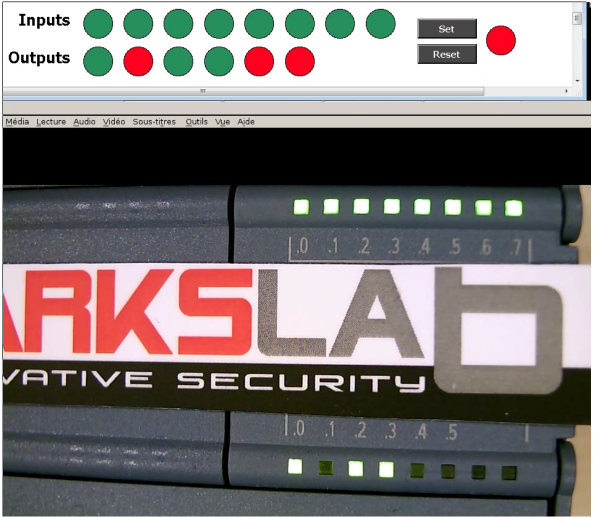
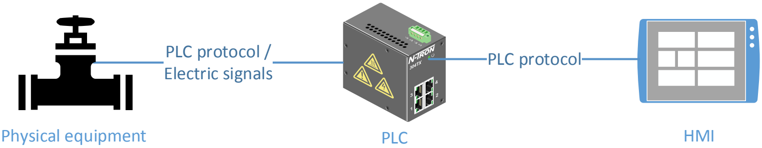


Figure: A PLC and the associated HMI

Plan

Why Specifically an Industrial Protocol?

- **Most public vulnerabilities are related to**
  - Either vulnerabilities not specific to industrial networks (embedded Web servers, for example)
  - Or protocols with a public specification
- **Industrial protocols are of main interest**
  - Critical: direct, low-level control of an industrial process
  - Essential: heart of the industrial system



Choosing Our Target

- Popular vendor, particularly in Europe
- Recent protocol, designed to be secure
    - Older protocol: partially documented, insecure
    - Recent version: state of the art security for an ICS
    - Offers password authentication
- Handles all the operations (both programming and supervision)
- Proprietary
    - Very few public work
    - Many things to be discovered

Previous Work

Previous versions

- Serious vulnerabilities (full RAM access)
- Showed that the (now older) protocol had no security feature

Same product family

- Work on password authentication
- Proofs of concept
- Some vulnerabilities
- Basic work on the protocol

What Did We Intend To Do?

1. Reverse a part of the protocol spec to build dissectors
2. Assess the protocol security
    - How does it implement authentication/integrity?
    - Any flaws in the design?
3. Assess the protocol implementations

Plan

Black-Box Analysis

- **Goals:**
  - Understand the general structure of the packets
  - Get the global signification of the traffic
  - Look for points of interest
- **Methodology:**
  - Controlled traffic generation
  - Differential analysis, between packets from:
    - Same session, different host
    - Same session, same host, different position
    - Different session, same host, same position
    - Etc.

Differential Analysis

"Believe it or not, if you stare at the hex dumps long enough, you start to see the patterns" - Rob Savoye

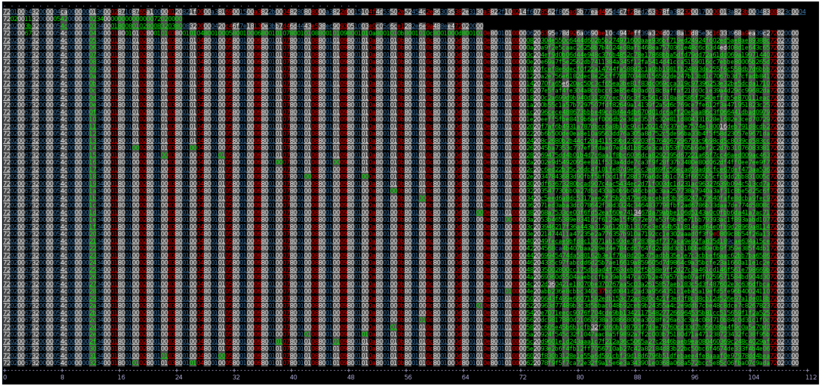


Figure: Differences between similar packets

hexlighter (https://github.com/fmonjalet/hexlighter)

Differential Analysis

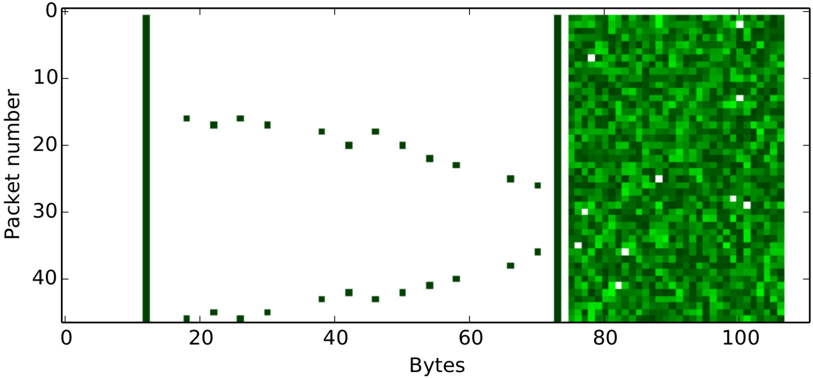


Figure: Differences between similar packets (brighter = greater absolute difference)

Differential Analysis

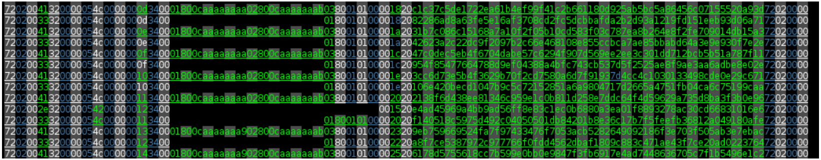


Figure: Realigned heterogeneous packets

Results

Results:

- Part of the specification has been deduced
- Dissection tools have been written
- Cryptography related fields have been identified: 32-byte high entropy field

And now?

- Cryptographic fields need white-box analysis
- Time to grab IDA

Plan

Choosing a Haystack

What shall we look at?

- Windows protocol clients (SCADA HMI): easy to debug/instrument
- Firmware: packed in a custom way and very hard to instrument
- Guess where we started...

What are we looking for?

- Code that processes network data
- Possible implementation of standard cryptographic primitives

Finding the Protocol Stack

How can we do that?

- The smart way: generate a trace of one process and taint data coming from network:
  - quite complicated on big software
  - multi process and shared memory issues
  - alternatively trace the whole system: can be really powerful, but requires specific software
- The half-smart way: follow the data from the network by breaking on memory/code: a hell in big enterprise-asynchronous-multiprocess-full-of-copies software
- The pragmatic way: look for specific cryptographic algorithm, in our case 32-byte hashing ones (such as SHA-256)

Letting signsrch do the job

- `signsrch` (http://aluigi.altervista.org/mytoolz.htm): automatic detection of classic cryptographic constants/code
- Executed on every DLL used by the main process
- One was more interesting than the others:

```
offset   num  description [bits.endian.size]
--------------------------------------------
xxxxxxxx 1036 SHA1 / SHA0 / RIPEMD-160 initialization [32.le.20&]
xxxxxxxx 2053 RIPEMD-128 InitState [32.le.16&]
xxxxxxxx 876  SHA256 Initial hash value H (0x6a09e667UL) [32.le.32&]
xxxxxxxx 1016 MD4 digest [32.le.24&]
xxxxxxxx 1299 classical random incrementer 0x343FD 0x269EC3 [32.le.8&]
[...]
xxxxxxxx 1290 __popcount_tab (compression?) [..256]
xxxxxxxx 874  SHA256 Hash constant words K (0x428a2f98) [32.le.256]
xxxxxxxx 894  AES Rijndael S / ARIA S1 [..256]
xxxxxxxx 897  Rijndael Te0 (0xc66363a5U) [32.be.1024]
xxxxxxxx 899  Rijndael Te1 (0xa5c66363U) [32.be.1024]
xxxxxxxx 901  Rijndael Te2 (0x63a5c663U) [32.be.1024]
xxxxxxxx 903  Rijndael Te3 (0x6363a5c6U) [32.be.1024]
xxxxxxxx 915  Rijndael rcon [32.be.40]
[...]
```

Plan

Unwinding the Cryptosystem

Starting point:

1. Break on suspicious code (SHA-256)
2. See that it is actually used with data from the packet
3. Static analysis reveals HMAC SHA-256.
4. Uses a MAC key, where does it come from?

Unwind:

1. Find out how the MAC key is generated
2. Black-Box analysis: locate the key exchange in the packets
3. White-Box analysis: find out how it is exchanged
4. Etc.

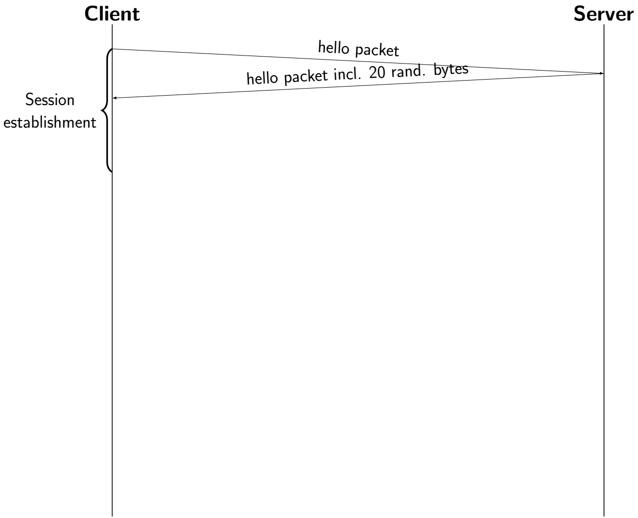
Cryptosystem Summary

- A session is established (with a given session number)
- The HMI generates a 128 bit AES key and a 180 bit MAC key
- The AES key is exchanged using an unknown algorithm
  - White-box cryptography, obfuscation
- The MAC key is sent encrypted using AES-128 GCM
- All the packets are now authenticated:

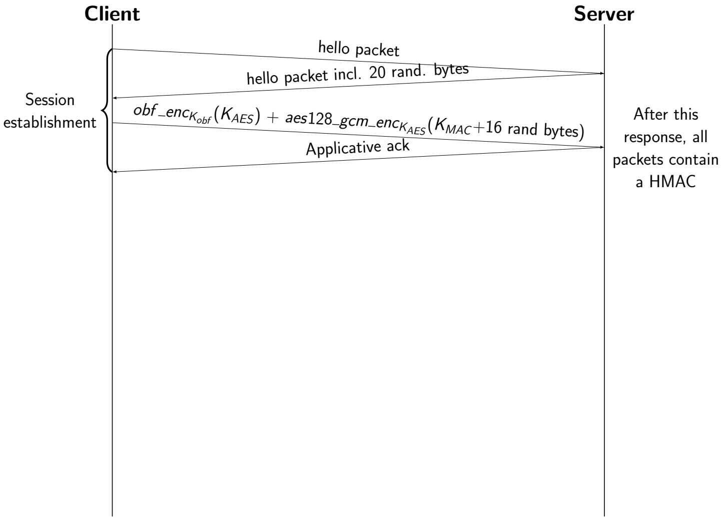
  `HMAC SHA-256(macKey, message)`

- User authentication: password (challenge/response)
- The authenticated peers are the only ones able to forge valid packets

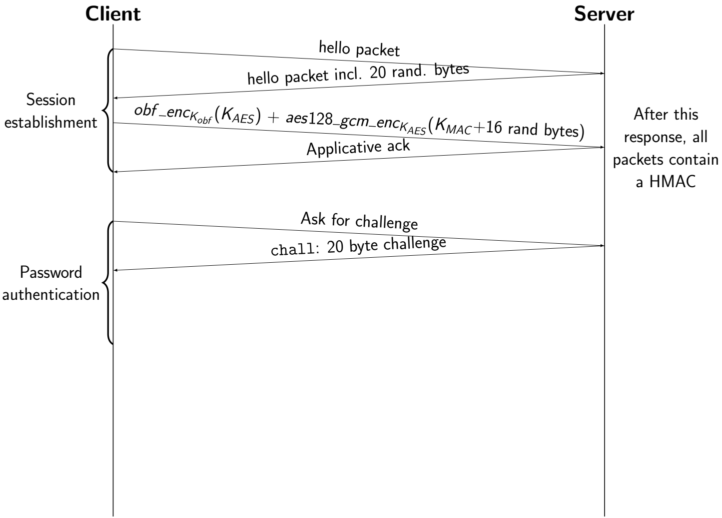
Cryptosystem Summary



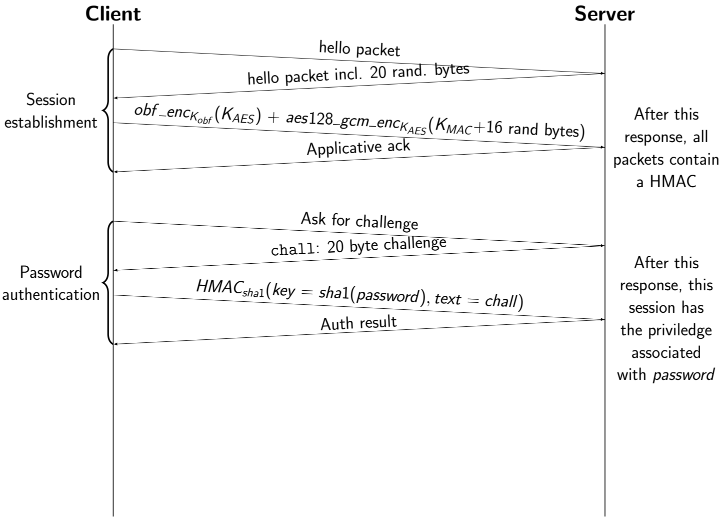
Cryptosystem Summary



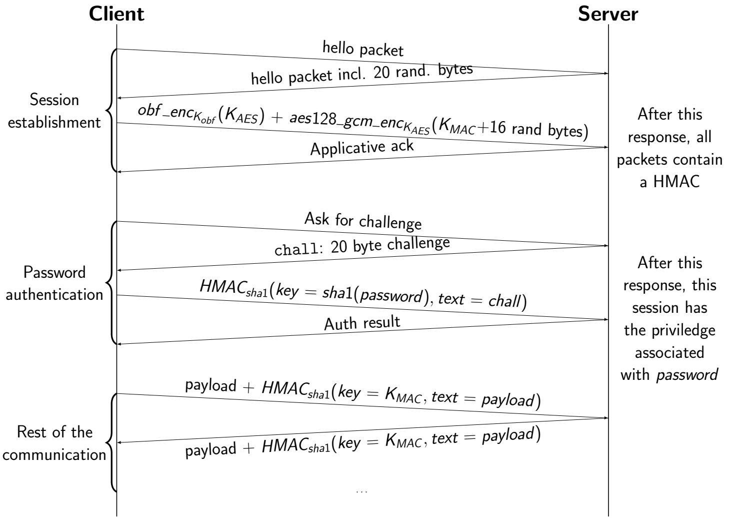
Cryptosystem Summary



Cryptosystem Summary



Cryptosystem Summary



Notes on the Protocol

- Client uses an (ECC?) public key (K_{obf}) to encrypt the first shared secret (K_{AES})

Notes on the Protocol

- Client uses an (ECC?) public key (K_{obf}) to encrypt the first shared secret (K_{AES})
- Key stored in an encrypted Zip client-side (password is hard-coded)
- Zip comes from the SCADA HMI installation

Notes on the Protocol

- Client uses an (ECC?) public key (K_{obf}) to encrypt the first shared secret (K_{AES})
- Key stored in an encrypted Zip client-side (password is hard-coded)
- Zip comes from the SCADA HMI installation
- The key retrieved in the Zip depends *only* on the PLC model
  $\Rightarrow$ Same private key for all similar PLCs
- **Goal:** reverse obfuscated crypto and recover private key from firmware (work in progress)

Plan

Vulnerability Description

- **Authenticity $\equiv$ secrecy of the MAC key.**
- Key collisions found when debugging

Vulnerability Description

- **Authenticity $\equiv$ secrecy of the MAC key.**
- Key collisions found when debugging
- **How is the key generated?**
  - `prng_init(0xffffffff)`
  - Deterministic sequence of calls to:
    - `prng_reseed("only for real entropy bytes!")`
    - `prng_gen_num(size)`
- Same MAC key sequence at every execution
- Easy brute force...
- Forge authenticated packets
- No need to break white-box cryptography

Building an Actual Attack

What can be done:

- Steal any authenticated session
- Act with the privileges of any active user
  - Arbitrary writes
  - PLC reprogrammation
- Spoof traffic (spoofed read values)
  $\Rightarrow$ Full control over the actual physical process

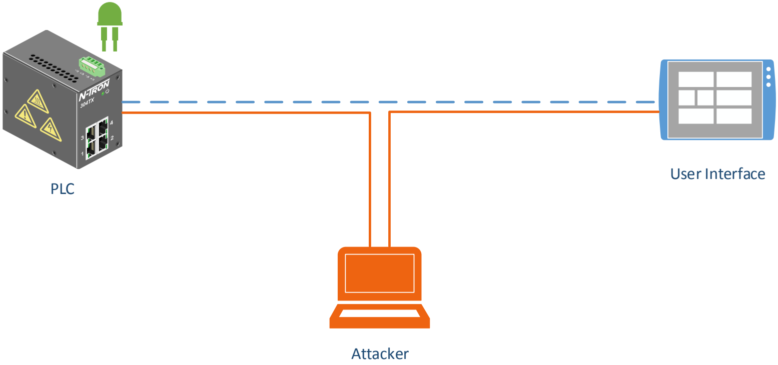
Exploiting it:

- Limited knowledge of the protocol is enough
- Differential analysis (real traffic, generated traffic)
  $\Rightarrow$ Isolate parts that need to be understood

Has been patched since this study

Demonstration

Exploiting the entropy loss: Man in the middle between PLC and supervision



Plan

Firmware Reverse Engineering

Motivation

- No white-box cryptography?
- Lighter obfuscation?

Accessing the firmware

- On the NAND of the PLC
- On the vendor's website: can be downloaded with a valid account

Update mechanism

- Update through Web server or SD Card
- Firmware code is fully compressed
- Unpacking done by the running firmware
  $\Rightarrow$ Black-box unpacking...

Looking for headers

Name

```
00000020  58 42 30 20 00 00 00 00 00 00 00 00 20 00 00 00  XB0 ........ ...
00000030  87 D0 FD FF 42 47 5F 41 42 4C BE 41 99 00 4D 68  ‡ÐýÿBG_ABL¾A™.Mh
00000040  8A E5 41 30 30 30 30 30 02 00 00 00 FF FF FF FF  ŠåA00000....ÿÿÿÿ
00000050  42 30 30 30 30 30 48 00 00 00 02 40 EE FF 46 57  B00000H....@îÿFW
00000060  5F 53 49 47 42 47 5F 41 42 4C 01 00 EF 00 00 00  _SIGBG_ABL..ï...
00000070  00 10 36 45 53 37 20 32 31 32 2D 31 42 45 34 30  ..6ES7 212-1BE40
00000080  2D 30 58 42 30 20 56 04 00 00 41 30 30 30 30 30  -0XB0 V...A00000
00000090  F2 B7 00 00 00 01 5D 1B 41 53 00 00 04 2D C0 00  ò·....].AS...-À.
000000A0  00 80 40 00 00 D8 B6 C7 00 04 00 00 00 00 00 00  .€@..Ø¶Ç........
000000B0  40 00 00 80 3F 00 10 00 00 00 56 04 00 00 40 00  @..€?.....V...@.
```

Looking for headers

Size

```
00000020  58 42 30 20 00 00 00 00 00 00 00 00 20 00 00 00  XB0 ........ ...
00000030  87 D0 FD FF 42 47 5F 41 42 4C BE 41 99 00 4D 68  ‡ÐýÿBG_ABL¾A™.Mh
00000040  8A E5 41 30 30 30 30 30 02 00 00 00 FF FF FF FF  ŠåA00000....ÿÿÿÿ
00000050  42 30 30 30 30 30 48 00 00 00 02 40 EE FF 46 57  B00000H....@îÿFW
00000060  5F 53 49 47 42 47 5F 41 42 4C 01 00 EF 00 00 00  _SIGBG_ABL..ï...
00000070  00 10 36 45 53 37 20 32 31 32 2D 31 42 45 34 30  ..6ES7 212-1BE40
00000080  2D 30 58 42 30 20 56 04 00 00 41 30 30 30 30 30  -0XB0 V...A00000
00000090  F2 B7 00 00 00 01 5D 1B 41 53 00 00 04 2D C0 00  ò·....].AS...-À.
000000A0  00 80 40 00 00 D8 B6 C7 00 04 00 00 00 00 00 00  .€@..Ø¶Ç........
000000B0  40 00 00 80 3F 00 10 00 00 00 56 04 00 00 40 00  @..€?.....V...@.
```

Looking for headers

CRC-32

```
00000020  58 42 30 20 00 00 00 00 00 00 00 00 20 00 00 00  XB0 ........ ...
00000030  87 D0 FD FF 42 47 5F 41 42 4C BE 41 99 00 4D 68  ‡ÐýÿBG_ABL¾A™.Mh
00000040  8A E5 41 30 30 30 30 30 02 00 00 00 FF FF FF FF  ŠåA00000....ÿÿÿÿ
00000050  42 30 30 30 30 30 48 00 00 00 02 40 EE FF 46 57  B00000H....@îÿFW
00000060  5F 53 49 47 42 47 5F 41 42 4C 01 00 EF 00 00 00  _SIGBG_ABL..ï...
00000070  00 10 36 45 53 37 20 32 31 32 2D 31 42 45 34 30  ..6ES7 212-1BE40
00000080  2D 30 58 42 30 20 56 04 00 00 41 30 30 30 30 30  -0XB0 V...A00000
00000090  F2 B7 00 00 00 01 5D 1B 41 53 00 00 04 2D C0 00  ò·....].AS...-À.
000000A0  00 80 40 00 00 D8 B6 C7 00 04 00 00 00 00 00 00  .€@..Ø¶Ç........
000000B0  40 00 00 80 3F 00 10 00 00 00 56 04 00 00 40 00  @..€?.....V...@.
```

Looking for headers

```
00000020  58 42 30 20 00 00 00 00 00 00 00 00 20 00 00 00  XB0 ........ ...
00000030  87 D0 FD FF 42 47 5F 41 42 4C BE 41 99 00 4D 68  ‡ÐýÿBG_ABL¾A™.Mh
00000040  8A E5 41 30 30 30 30 30 02 00 00 00 FF FF FF FF  ŠåA00000....ÿÿÿÿ
00000050  42 30 30 30 30 30 48 00 00 00 02 40 EE FF 46 57  B00000H....@îÿFW
00000060  5F 53 49 47 42 47 5F 41 42 4C 01 00 EF 00 00 00  _SIGBG_ABL..ï...
00000070  00 10 36 45 5     BG_ABL     2 2D 31 42 45 34 30  ..6ES7 212-1BE40
00000080  2D 30 58 42 30 20 56 04 00 00 41 30 30 30 30 30  -0XB0 V...A00000
00000090  F2 B7 00 00 00 01 5D 1B 41 53 00 00 04 2D C0 00  ò·....].AS...-À.
000000A0  00 80 40 00 00 D8 B6 C7     A00000     00 00 00  .€@..Ø¶Ç........
000000B0  40 00 00 80 3F 00 10 00 00 00 56 04 00 00 40 00  @..€?.....V...@.
```

Layout of the code section

Section A00000

Size of chunks

```
00000090  F2 B7 00 00 00 01 5D 1B 41 53 00 00 04 2D C0 00  ò.....].AS...-À.
000000A0  00 80 40 00 00 D8 B6 C7 00 04 00 00 00 00 00 00  .€@..Ø¶Ç........
000000B0  40 00 00 80 3F 00 10 00 00 00 56 04 00 00 40 00  @..€?.....V...@.
                                ...
0000B880  2C 20 FF E2 03 00 02 C1 00 00 00 01 E3 A0 90 00  , ÿâ...Á....ã ..
0000B890  00 E1 A0 B0 09 E8 A3 0A 04 00 E3 A0 B0 4C E8 83  .á °.è£...ã °Lèƒ
0000B8A0  0A 00 00 E2 87 70 01 E2 5E E0 01 00 1A FF FF E3  ...â‡p.â^à...ÿÿã
0000B8B0  E3 A0 70 29 00 E3 A0 B0 20 E0 87 21 07 00 E0 85  ã p).ã ° à‡!..à…
                                ...
00017980  74 50 01 01 03 15 01 A0 70 00 00 00 60 CA 00 00  tP..... p...`Ê..
00017990  00 02 E3 A0 00 01 00 E1 C5 00 BC E2 8D 00 40 00  ..ã ...áÅ.¼â..@.
000179A0  EB FF FA A0 E1 57 00 00 00 2A 00 00 EA E1 A0 10  ëÿú áW...*..êá .
000179B0  07 00 E2 87 70 01 E2 8D 00 40 80 05 D0 10 B2 E3  ..â‡p.â..@€.Ð.²ã
                                ...
000243F0  44 D3 00 00 00 00 E1 A0 00 05 00 EB 00 0B 3A E1  DÓ....á ...ë..:á
00024400  A0 00 05 80 03 38 E1 B0 70 00 1A 00 00 00 00 EB   ..€.8á°p......ë
00024410  FF 7C E3 E2 8A 00 0F 5F E1 D0 10 B6 E1 A0 00 00  ÿ|ãâŠ.._áÐ.¶á ..
```

Interesting chunks: low compression

```
00814000  00 3C 53 45 52 56 45 52 50 00 41 47 45 53 3E 0D  .<SERVERP.AGES>.
00814010  0A 3C 00 21 2D 2D 20 54 68 65 20 00 44 65 66 61  .<.!-- The .Defa
00814020  75 6C 74 20 00 6C 69 6E 6B 20 61 74 20 28 74 68  ult .link at (th
00814030  02 42 01 73 65 20 00 54 61 67 20 77 69 6C 6C 01  .B.se .Tag will.
00814040  20 62 65 20 75 73 65 02 00 77 68 65 6E 20 61 20   be use..when a.
00814050  52 05 65 71 75 65 73 02 63 01 02 75 6C 64 20 6E  R.eques.c..uld n
00814060  6F 02 62 00 65 20 72 65 73 6F 6C 76 00 65 64 20  o.b.e resolv.ed.
00814070  2D 2D 3E 0D 0A 00 3C 42 41 53 45 20 4C 4F 00 43  -->...<BASE LO.C
```

Interesting chunks: low compression

```
00000000  00 3C 53 45 52 56 45 52 50  .<SERVERP
00000009  00 41 47 45 53 3E 0D 0A 3C  .AGES>..<
00000012  00 21 2D 2D 20 54 68 65 20  .!-- The.
0000001B  00 44 65 66 61 75 6C 74 20  .Default.
00000024  00 6C 69 6E 6B 20 61 74 20  .link at.
0000002D  28 74 68 02 42 01 73 65 20  (th.B.se.
00000036  00 54 61 67 20 77 69 6C 6C  .Tag will
0000003F  01 20 62 65 20 75 73 65 02  . be use.
00000048  00 77 68 65 6E 20 61 20 52  .when a R
00000051  05 65 71 75 65 73 02 63 01  .eques.c.
0000005A  02 75 6C 64 20 6E 6F 02 62  .uld no.b
00000063  00 65 20 72 65 73 6F 6C 76  .e resolv
0000006C  00 65 64 20 2D 2D 3E 0D 0A  .ed -->..
00000075  00 3C 42 41 53 45 20 4C 4F  .<BASE LO
0000007E  00 43 41 4C 4C 49 4E 4B 3D  .CALLINK=
00000087  00 22 2F 22 20 50 52 45 46  ."/" PREF
```

Interesting chunks: low compression

First byte: mask

```
00000000  00 3C 53 45 52 56 45 52 50  .<SERVERP
00000009  00 41 47 45 53 3E 0D 0A 3C  .AGES>..<
00000012  00 21 2D 2D 20 54 68 65 20  .!-- The.
0000001B  00 44 65 66 61 75 6C 74 20  .Default.
00000024  00 6C 69 6E 6B 20 61 74 20  .link at.
0000002D  28 74 68 02 42 01 73 65 20  (th.B.se.
00000036  00 54 61 67 20 77 69 6C 6C  .Tag will
0000003F  01 20 62 65 20 75 73 65 02  . be use.
00000048  00 77 68 65 6E 20 61 20 52  .when a R
00000051  05 65 71 75 65 73 02 63 01  .eques.c.
0000005A  02 75 6C 64 20 6E 6F 02 62  .uld no.b
00000063  00 65 20 72 65 73 6F 6C 76  .e resolv
0000006C  00 65 64 20 2D 2D 3E 0D 0A  .ed -->..
00000075  00 3C 42 41 53 45 20 4C 4F  .<BASE LO
0000007E  00 43 41 4C 4C 49 4E 4B 3D  .CALLINK=
00000087  00 22 2F 22 20 50 52 45 46  ."/" PREF
```

Interesting chunks: low compression

First byte: mask

Red bytes: length

```
00000000  00 3C 53 45 52 56 45 52 50  .<SERVERP
00000009  00 41 47 45 53 3E 0D 0A 3C  .AGES>..<
00000012  00 21 2D 2D 20 54 68 65 20  .!-- The.
0000001B  00 44 65 66 61 75 6C 74 20  .Default.
00000024  00 6C 69 6E 6B 20 61 74 20  .link at.
0000002D  28 74 68 02 42 01 73 65 20  (th.B.se.
00000036  00 54 61 67 20 77 69 6C 6C  .Tag will
0000003F  01 20 62 65 20 75 73 65 02  . be use.
00000048  00 77 68 65 6E 20 61 20 52  .when a R
00000051  05 65 71 75 65 73 02 63 01  .eques.c.
0000005A  02 75 6C 64 20 6E 6F 02 62  .uld no.b
00000063  00 65 20 72 65 73 6F 6C 76  .e resolv
0000006C  00 65 64 20 2D 2D 3E 0D 0A  .ed -->..
00000075  00 3C 42 41 53 45 20 4C 4F  .<BASE LO
0000007E  00 43 41 4C 4C 49 4E 4B 3D  .CALLINK=
00000087  00 22 2F 22 20 50 52 45 46  ."/" PREF
```

Compression

Summary:

- Blocks of 9 bytes: 1 byte of mask, 8 bytes of data
- Pieces of data encoded by their length
  - No length/distance...
- Compression increases inside a chunk

⇒ LZ-based compression

Compression: LZP

LZP

- One and only algorithm coding only the length on WikiBooks.
- Improvement to dictionary coding/context coding.
- 4 variants. Here LZP3 is used.
- No public implementation has been found.

Usage

- Unpack each block of the A00000 section. Each block is 64KB.
- Got plain text firmware.
- CRC-32 at the end to confirm.

Memory layout

- Unpacked firmware: no known format, raw blob.
- Memory layout is described in the binary.
- Used by the boot loader.
- IDA loader written to load the firmware with a correct mapping.

One bad news: obfuscation is still here...

Firmware signature

Goal

- Bypass the signature mechanism
- Inject our own code

Signature check

- ECDSA-256 with SHA-256, standard curve and generator (ANSI X9.62 P-256)
- All the firmware is signed, except the last 78 bytes (FW_SIG section, fixed size)
- Custom code, will implemented. Fixed size numbers.

$\Rightarrow$ No vulnerability has been found.

Future work

- White-box cryptography.
  - Authentication: private key of the PLC. One key to rule them all.
  - Encryption of the user programs (AES, seems to be easy).
- Better understanding of the protocol.
  - Lot of information in the firmware.
- Get code execution.
  - Inject our own code.
  - Modify the behavior of the existing code.

Plan

Conclusion

Industrial technology still not mature

- Cryptography misuses
- Easy session stealing
- Non standard authentication scheme

Some real progress

- Efforts to build a secure protocol
- Way better than other what used to be done
- Very reactive vendor
- Things are going in the right direction

Questions?

INNOVATIVE SECURITY

www.quarkslab.com

contact@quarkslab.com | @quarkslab.com