Analyse de sécurité de technologies propriétaires SCADA

Jean-Baptiste Bédrune
jbbedrune@quarkslab.com
Alexandre Gazet
agazet@quarkslab.com
Florent Monjalet
florent.monjalet@cea.fr

Plan

Qu'est-ce qu'un système industriel ?

Système industriel / Industrial Control System (ICS)

Réseau informatique contrôlant un processus physique.

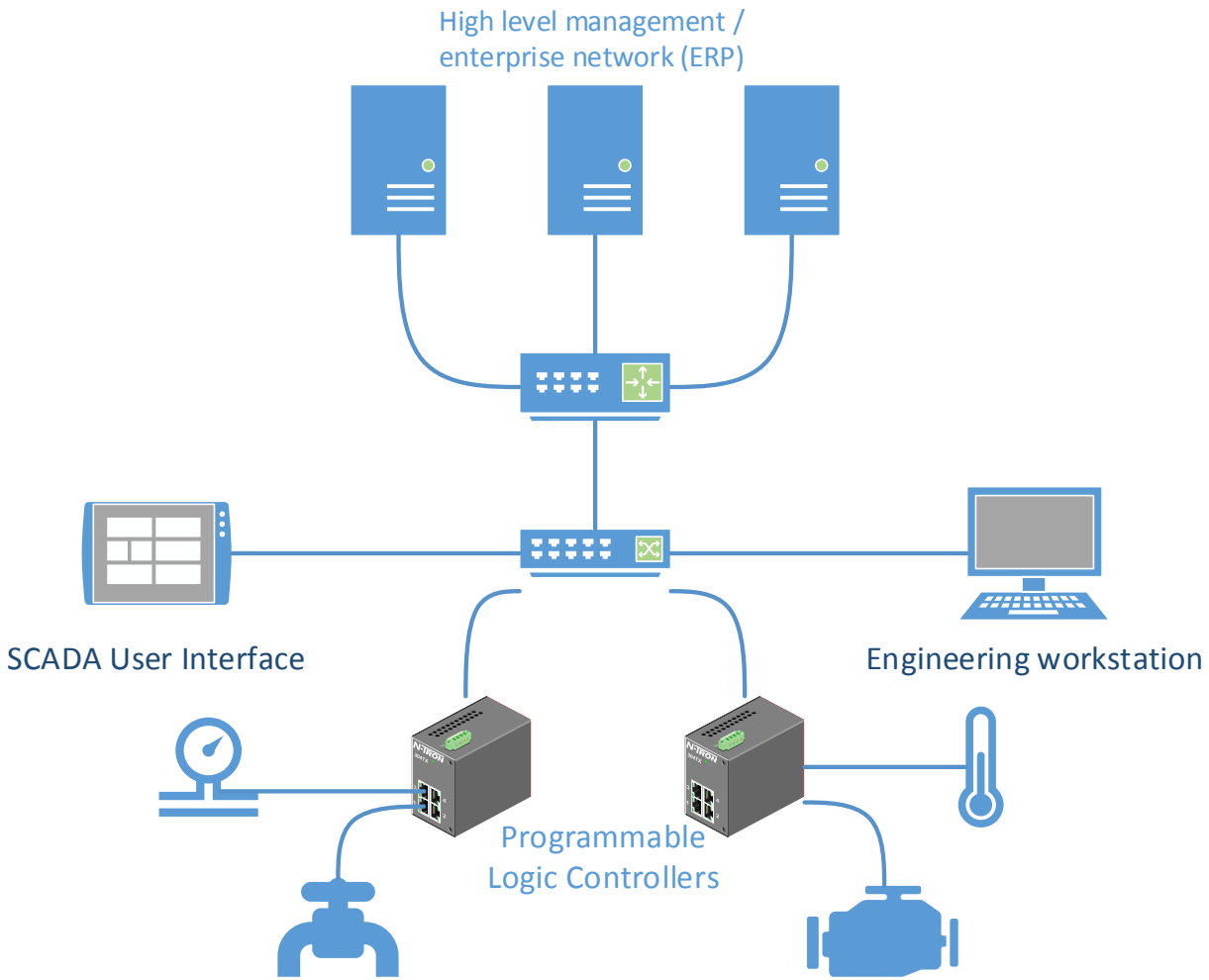
Présents dans beaucoup de systèmes critiques :

- Centrale nucléaire
- Distribution d'eau
- Aiguillage de trains
- Contrôles d'accès
- Chaînes de production

Supervisory Control and Data Acquistion (SCADA)

Partie supervision et contrôle direct du processus physique (sous-partie de l'ICS).

Composants d'un ICS



SCADA

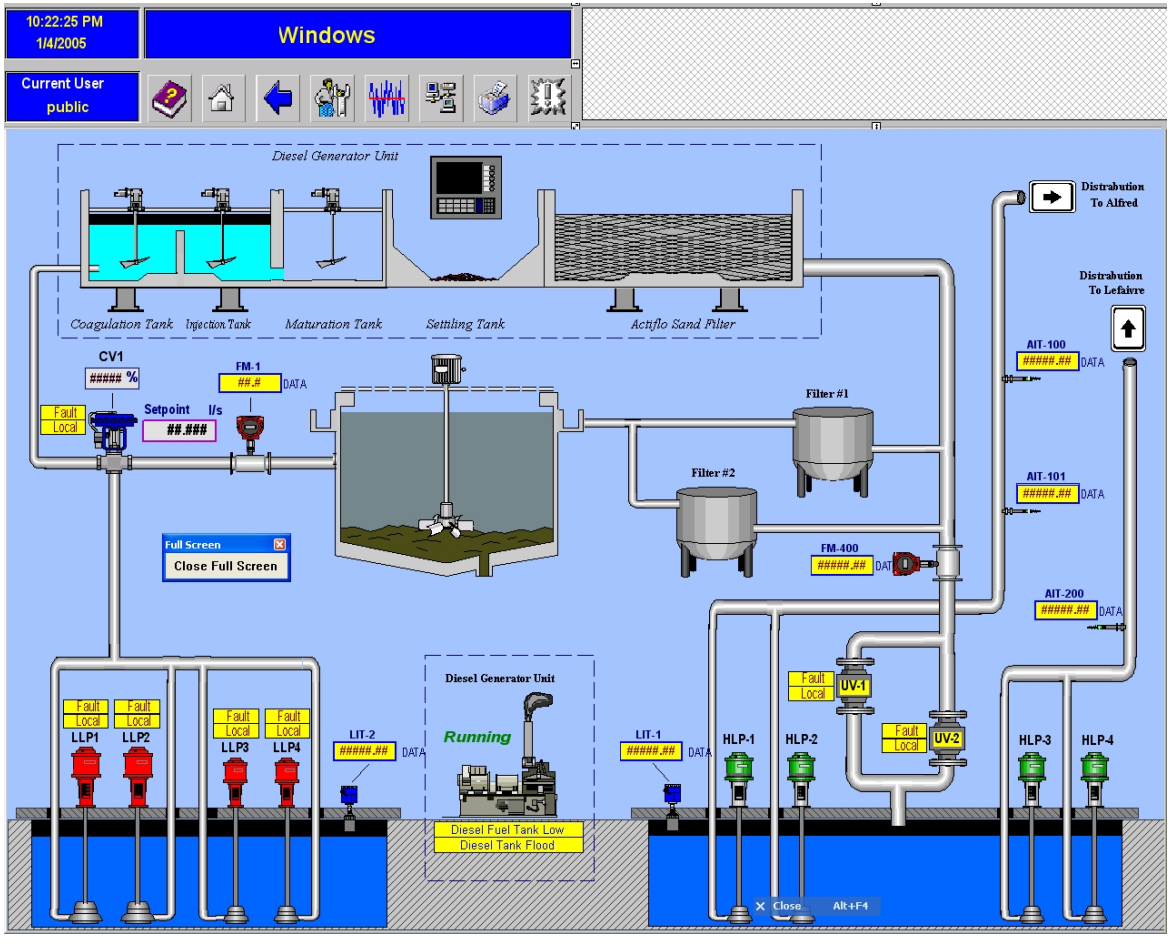


Figure: Exemple d'IHM SCADA (fastweb.it)

Pourquoi spécifiquement un protocole industriel ?

Constat : une grande partie des vulnérabilités publiées concernent des fonctionnalités non spécifiques aux réseaux industriels (serveurs Web embarqués)
Intérêt de l'étude d'un protocole métier :

- Critique : contrôle direct et bas niveau du processus industriel
- Indispensable : cœur du système industriel

Pourquoi spécifiquement un protocole industriel ?

Constat : une grande partie des vulnérabilités publiées concernent des fonctionnalités non spécifiques aux réseaux industriels (serveurs Web embarqués)

Intérêt de l'étude d'un protocole métier :

- Critique : contrôle direct et bas niveau du processus industriel
- Indispensable : cœur du système industriel

La cible :

- Protocole récent, conçu pour être sécurisé
- Propriétaire
  - Très peu de travaux publics existants
  - Beaucoup de choses à découvrir

Objectifs de l'étude ?

1. Reverser une partie du protocole pour écrire des dissecteurs
2. Évaluer la sécurité du protocole
   - Comment l'authenticité/l'intégrité est-elle implémentée ?
   - Des erreurs de conception ?
3. Évaluer l'implémentation du protocole

Plan

Analyse en boîte noire

- **Buts :**
  - Identification de la structure générale des paquets
  - Recherche de points d'intérêt
- **Méthodologie :**
  - Génération contrôlée de trafic
  - Analyse différentielle, entre paquets de :
    - Même session, hôte différent
    - Même session, même hôte, position différente dans l'échange
    - Session différente, même hôte, même position
    - Etc.

Analyse différentielle

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

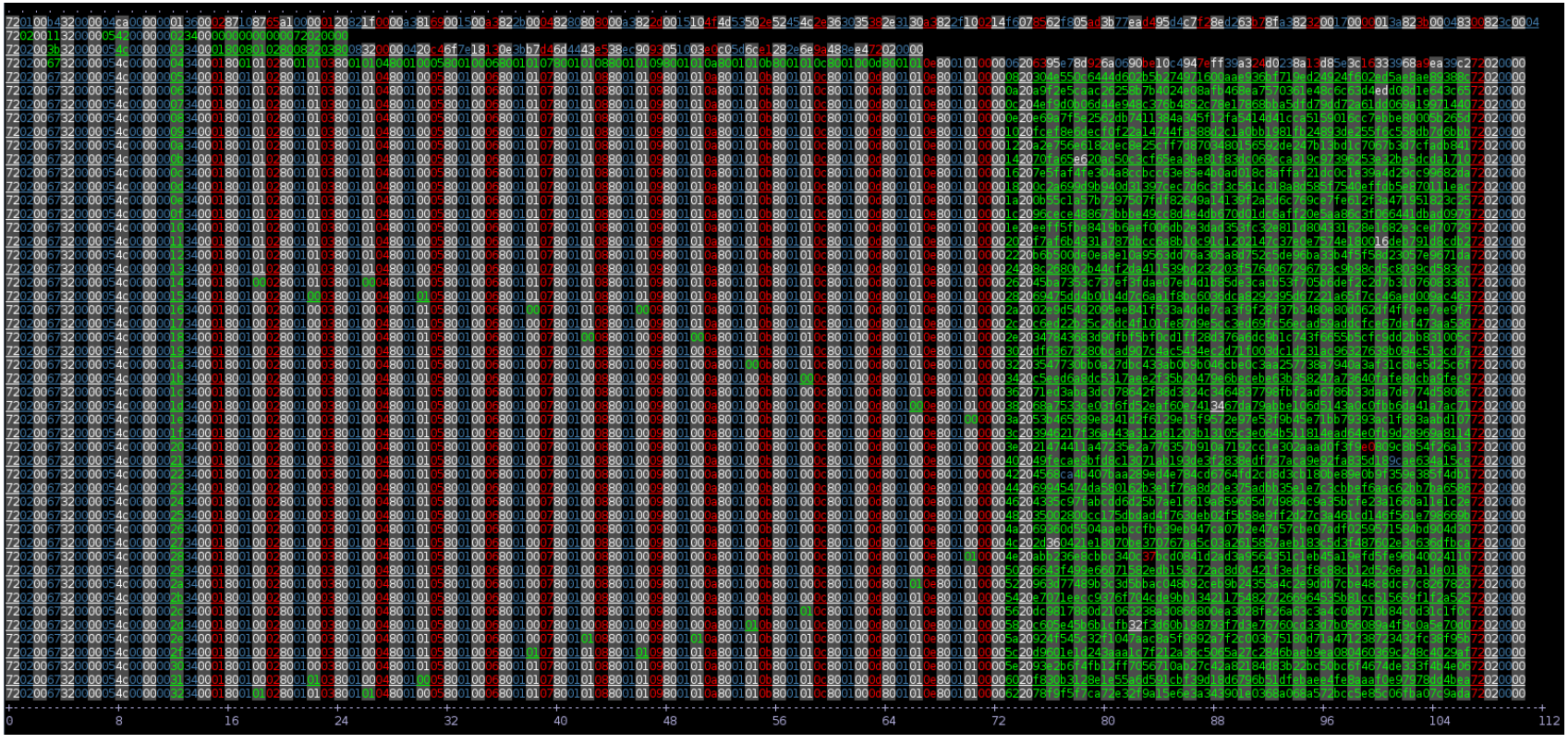


Figure: Différences entre paquets envoyés par le PLC [1]

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

Analyse différentielle

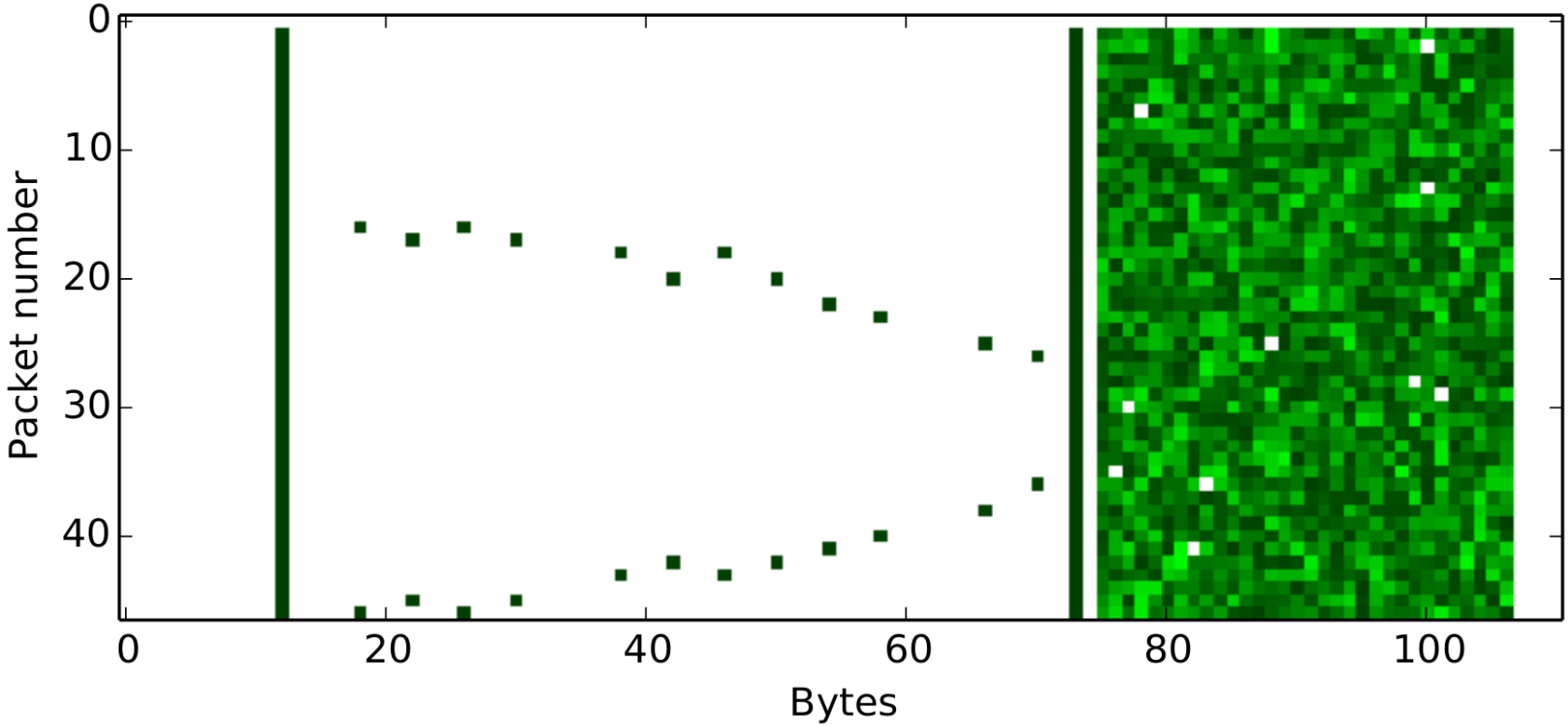


Figure: Différences entre paquets envoyés par le PLC (plus vif = plus grande différence)

Résultats

Résultats :

- Déduction d'une partie de la spécification
- Outils de dissection
- Identification des champs liés à de la cryptographie (32 octets à forte entropie)

Et maintenant ?

- Les champs cryptographiques nécessitent une analyse en boîte blanche
- Étude des binaires

Plan

Recherche des fonctions intéressantes

- Étude des clients Windows (IHM) : facile à instrumenter/débugger
- Recherche du code traitant le bloc de 32 octets

2. http://aluigi.altervista.org/mytoolz.htm

Recherche des fonctions intéressantes

- Étude des clients Windows (IHM) : facile à instrumenter/débugger
- Recherche du code traitant le bloc de 32 octets

- Première approche : suivre manuellement les données depuis `recv` jusqu'à leur lecture
  - Multiprocessing, mémoire partagée, asynchronicité, nombreuses copies...
- Deuxième approche : recherche de primitives cryptographiques intéressantes (par exemple un hash de 32 octets)
  - `signsrch`[2] : détection de constantes cryptographiques (et autres signatures) classiques
  - Dans une DLL : SHA-256, AES

2. http://aluigi.altervista.org/mytoolz.htm

Plan

Découverte du cryptosystème

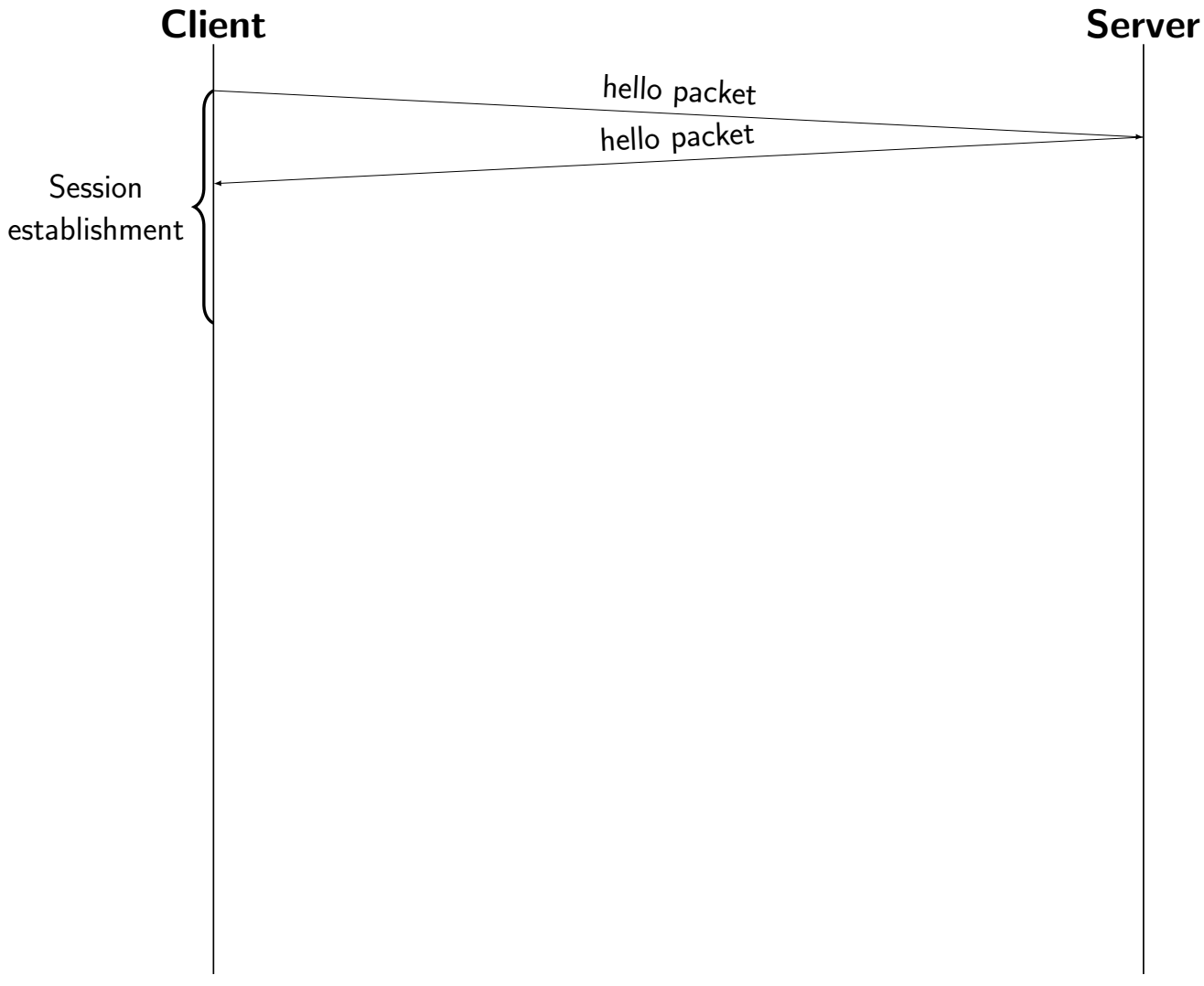
Point de départ :

1. SHA-256 appelé à chaque paquet reçu ou émis
2. SHA-256 utilisé dans un HMAC SHA-256
3. Utilisation d'une clé MAC, d'où vient-elle ?

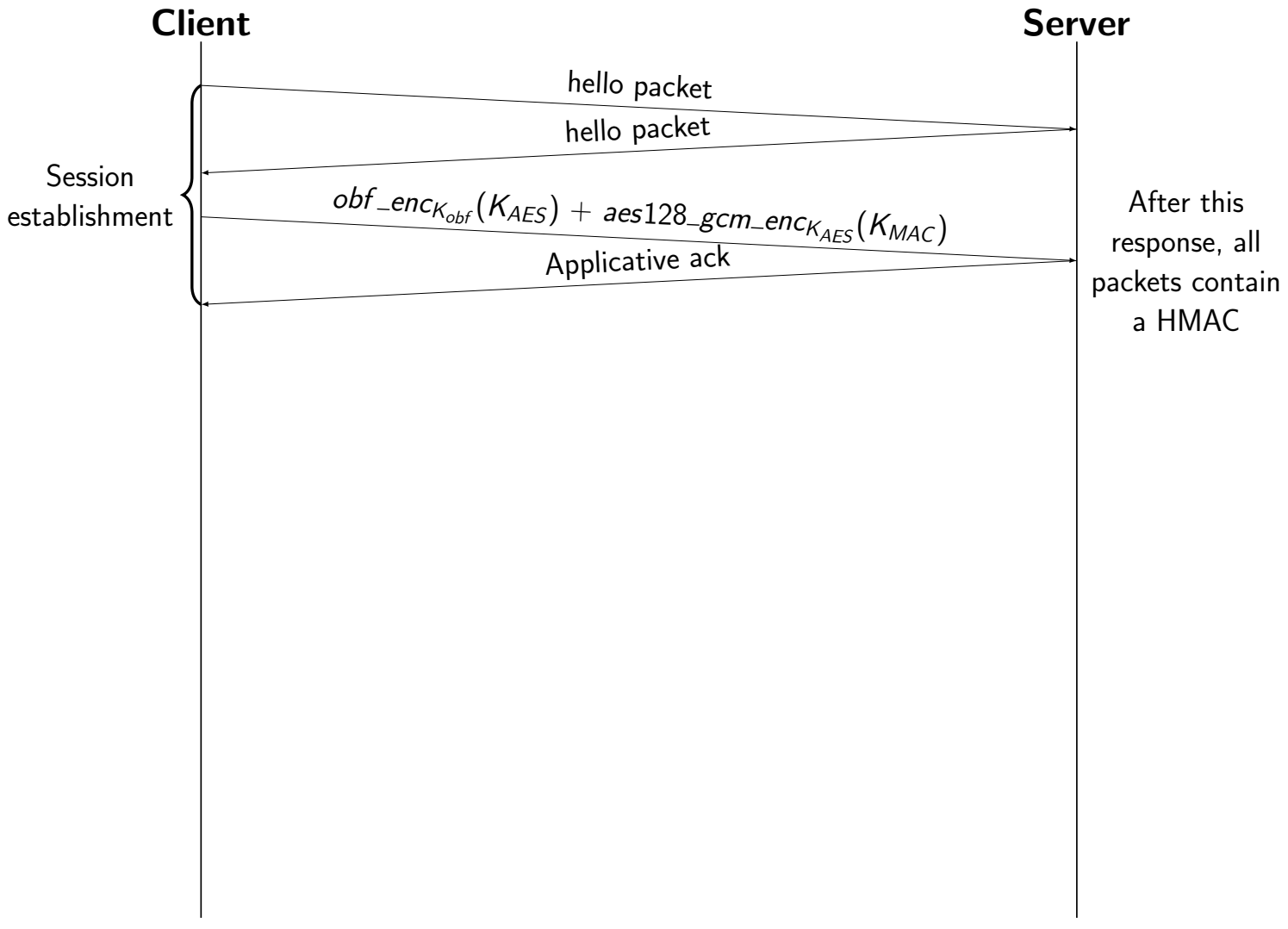
Itérations :

1. Comment la clé MAC est-elle générée ?
2. Analyse boîte noire : trouver l'échange de clés dans les paquets
3. Analyse boîte blanche : trouver *comment* l'échange a lieu
4. Etc.

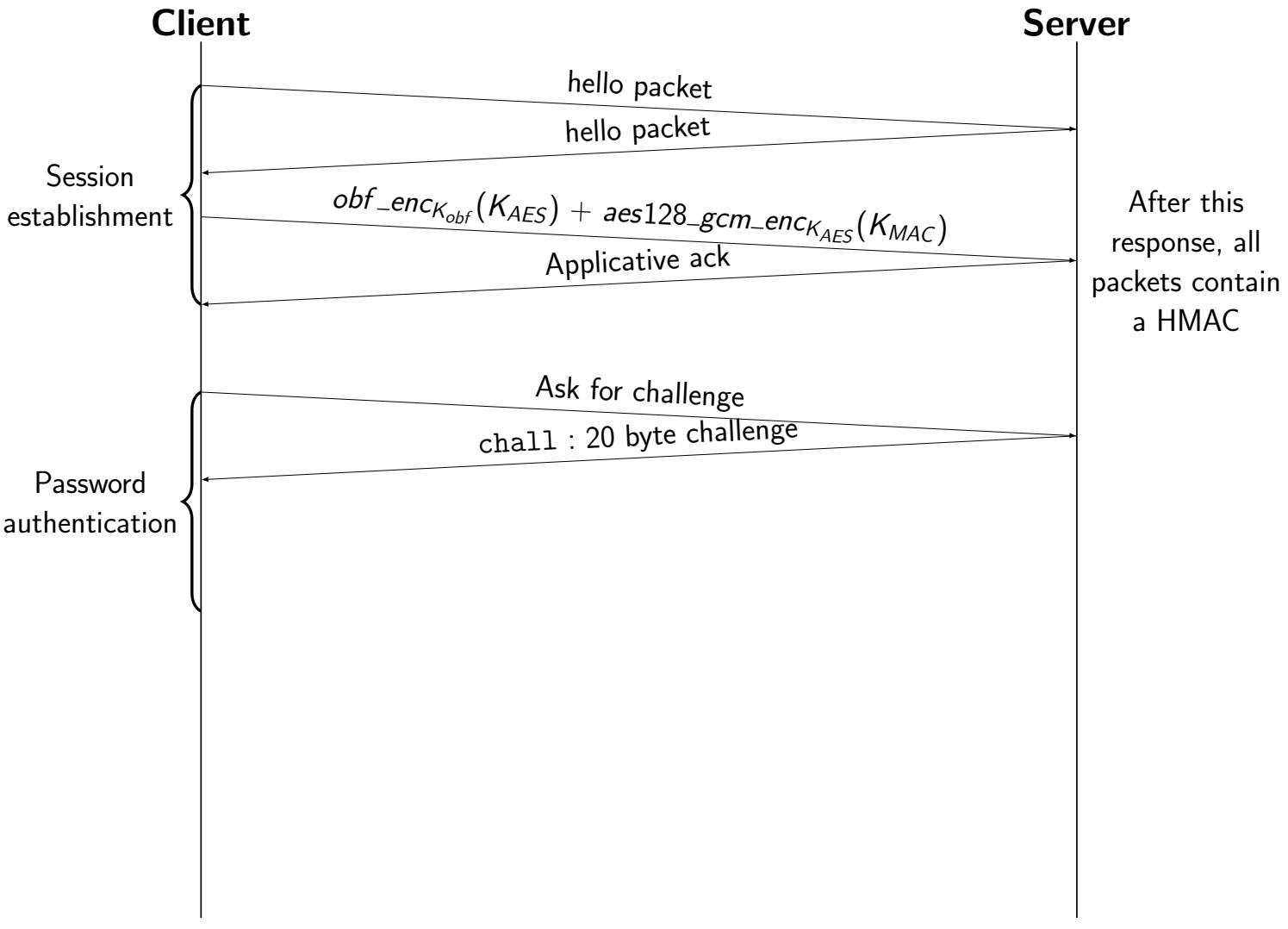
Résumé du cryptosystème



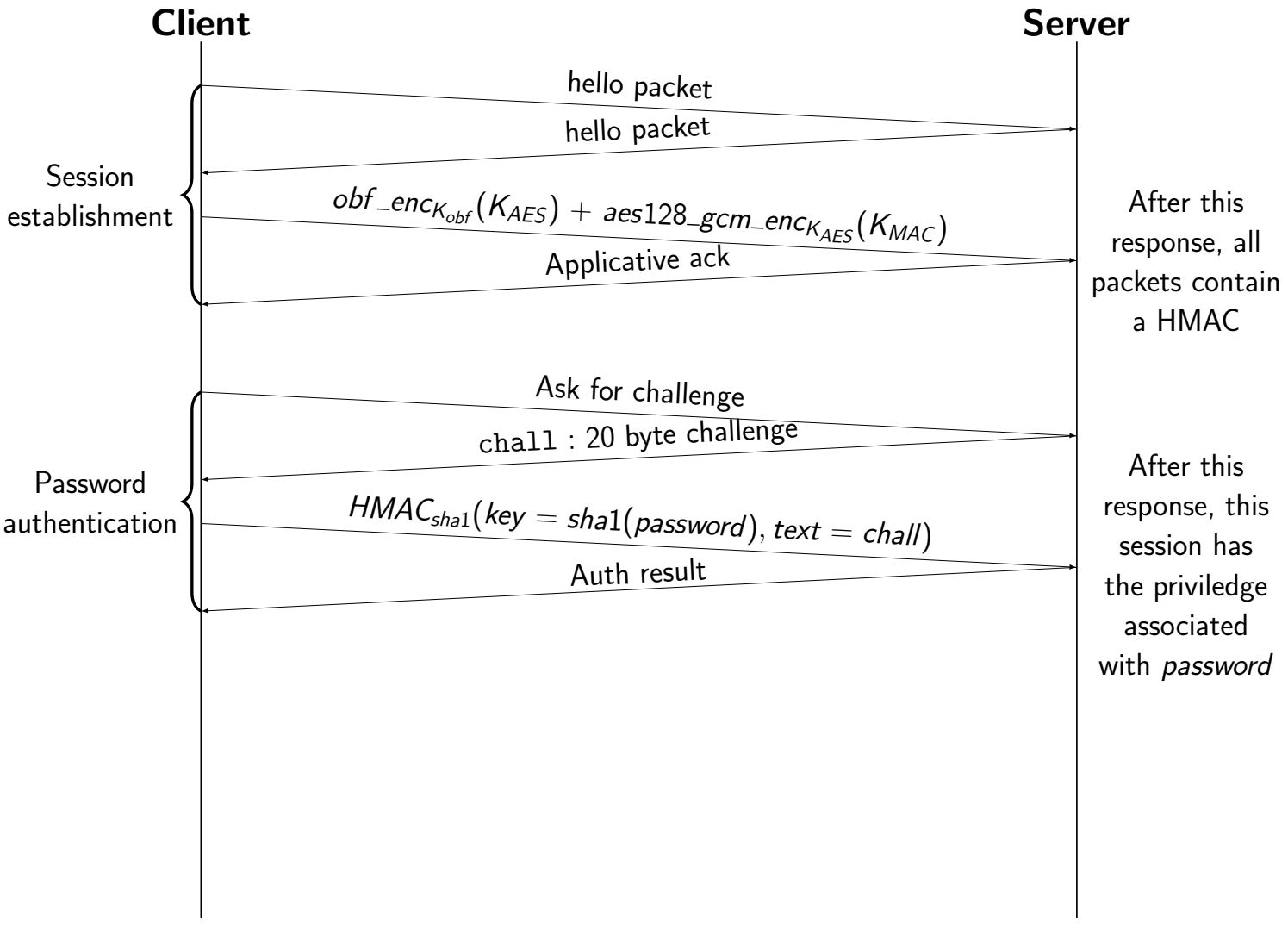
Résumé du cryptosystème



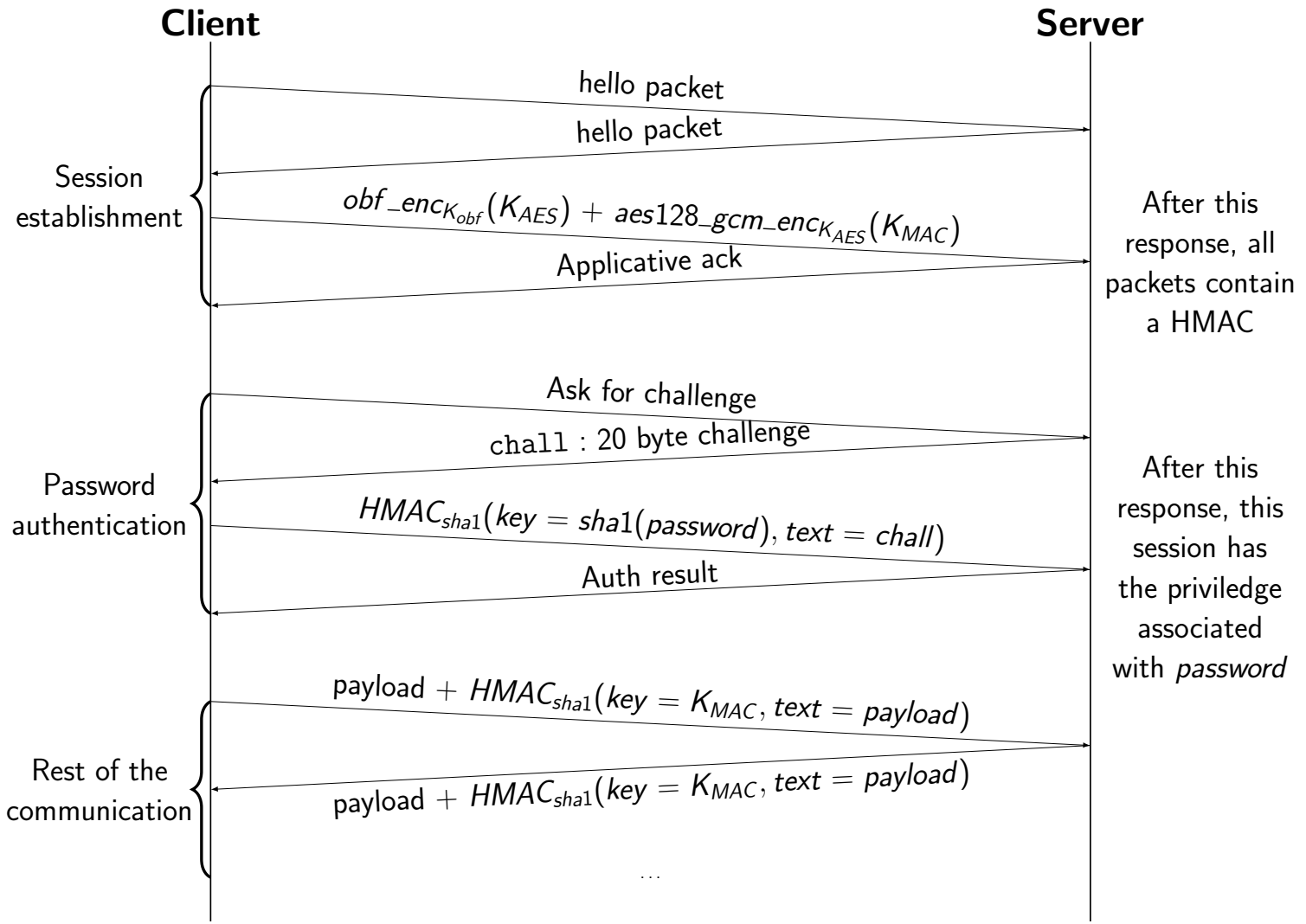
Résumé du cryptosystème



Résumé du cryptosystème



Résumé du cryptosystème



Remarques sur le protocole

- Le client utilise une clé publique (ECC ?) K_{obf} pour chiffrer le premier secret partagé (K_{AES})

Remarques sur le protocole

- Le client utilise une clé publique (ECC ?) K_{obf} pour chiffrer le premier secret partagé (K_{AES})
- La clé est stockée dans un Zip chiffré côté client (mot de passe hardcodé)
- Le Zip vient de l'installation de l'IHM SCADA

Remarques sur le protocole

- Le client utilise une clé publique (ECC ?) K_{obf} pour chiffrer le premier secret partagé (K_{AES})
- La clé est stockée dans un Zip chiffré côté client (mot de passe hardcodé)
- Le Zip vient de l'installation de l'IHM SCADA
- La clé récupérée dans le Zip ne dépend *que* du modèle de PLC
  $\Rightarrow$ Même clé privée pour tous les PLCs d'un même modèle
- **Objectif :** reverser la crypto obscurcie et extraire la clé privée du PLC (work in progress)

Plan

Mais où est l'entropie ?

- **Authenticité $\equiv$ confidentialité de la clé MAC.**
- Collisions de clés vues pendant la phase de débogage

Mais où est l'entropie ?

- **Authenticité ≡ confidentialité de la clé MAC.**
- Collisions de clés vues pendant la phase de débogage
- **Comment ces clés sont-elles générées ?**
  - `prng_init(0xffffffff)`
  - Suite déterministe d'appels à :
    - `prng_reseed("only for real entropy bytes!")`
    - `prng_gen_num(size)`
- Toujours la même séquence de clés MAC générée
- Bruteforce aisé...
- Possibilité de forger des paquets authentifiés
- Pas besoin de casser la white-box

Mais où est l'entropie ?

- **Authenticité ≡ confidentialité de la clé MAC.**
- Collisions de clés vues pendant la phase de débogage
- **Comment ces clés sont-elles générées ?**
  - `prng_init(0xffffffff)`
  - Suite déterministe d'appels à :
    - `prng_reseed("only for real entropy bytes!")`
    - `prng_gen_num(size)`
- Toujours la même séquence de clés MAC générée
- Bruteforce aisé...
- Possibilité de forger des paquets authentifiés
- Pas besoin de casser la white-box

La vulnérabilité a été patchée depuis

Exploitation de la vulnérabilité

Prérequis :

- Accès au réseau
- *Man-in-the-middle*

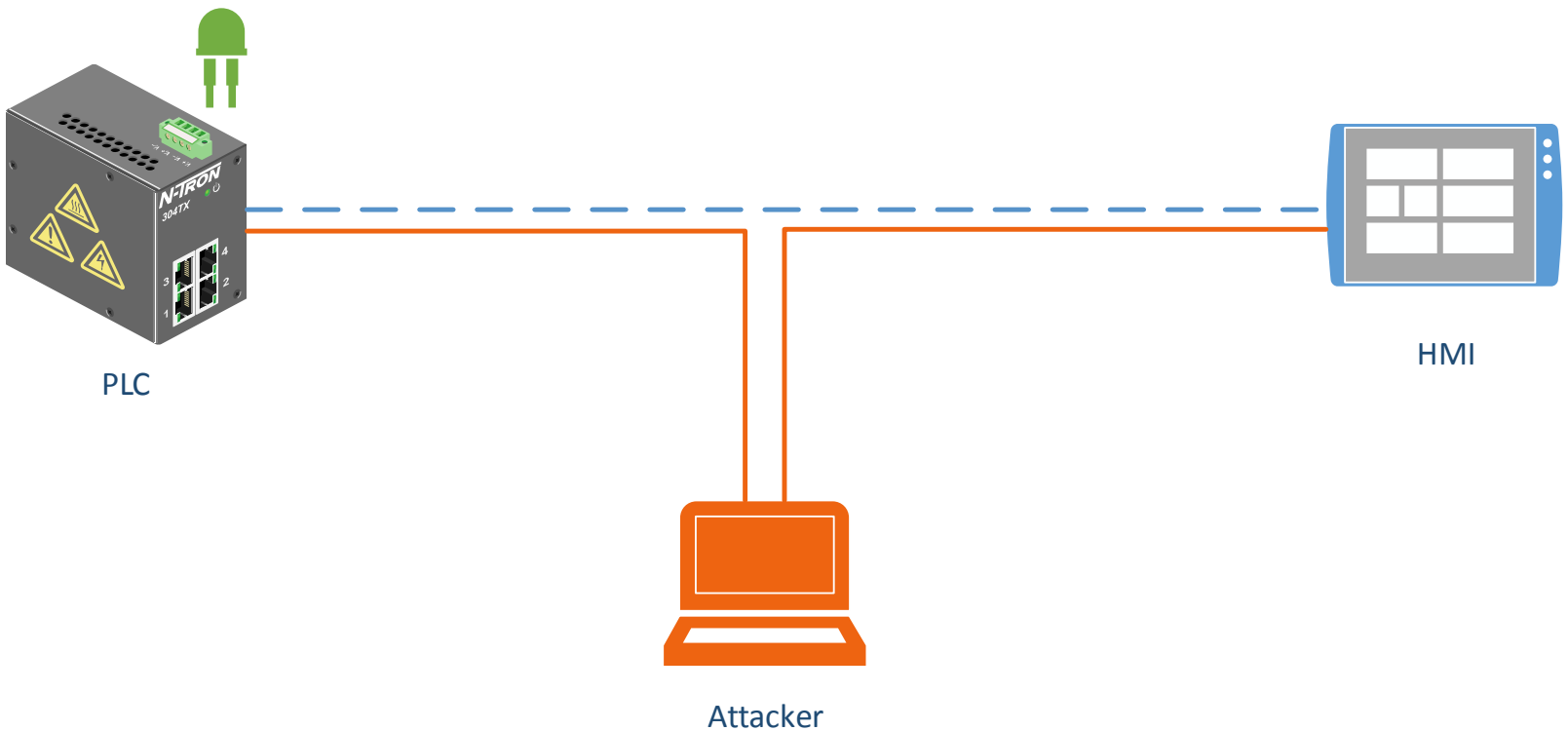
Que peut-on faire avec ?

- Voler n'importe quelle session authentifiée
- Changer les valeurs renvoyées par le PLC
- Agir avec les privilèges de l'utilisateur actif :
  - Écritures arbitraires sur le PLC
  - Reprogrammation du PLC

  $\Rightarrow$ Contrôle total du processus physique

Démonstration

Exploitation du manque d'entropie :
Man-in-the-middle entre un PLC et la supervision



Plan

Rétroconception du firmware

Motivation

- Pas de cryptographie boîte blanche ?
- Obfuscation plus légère ?

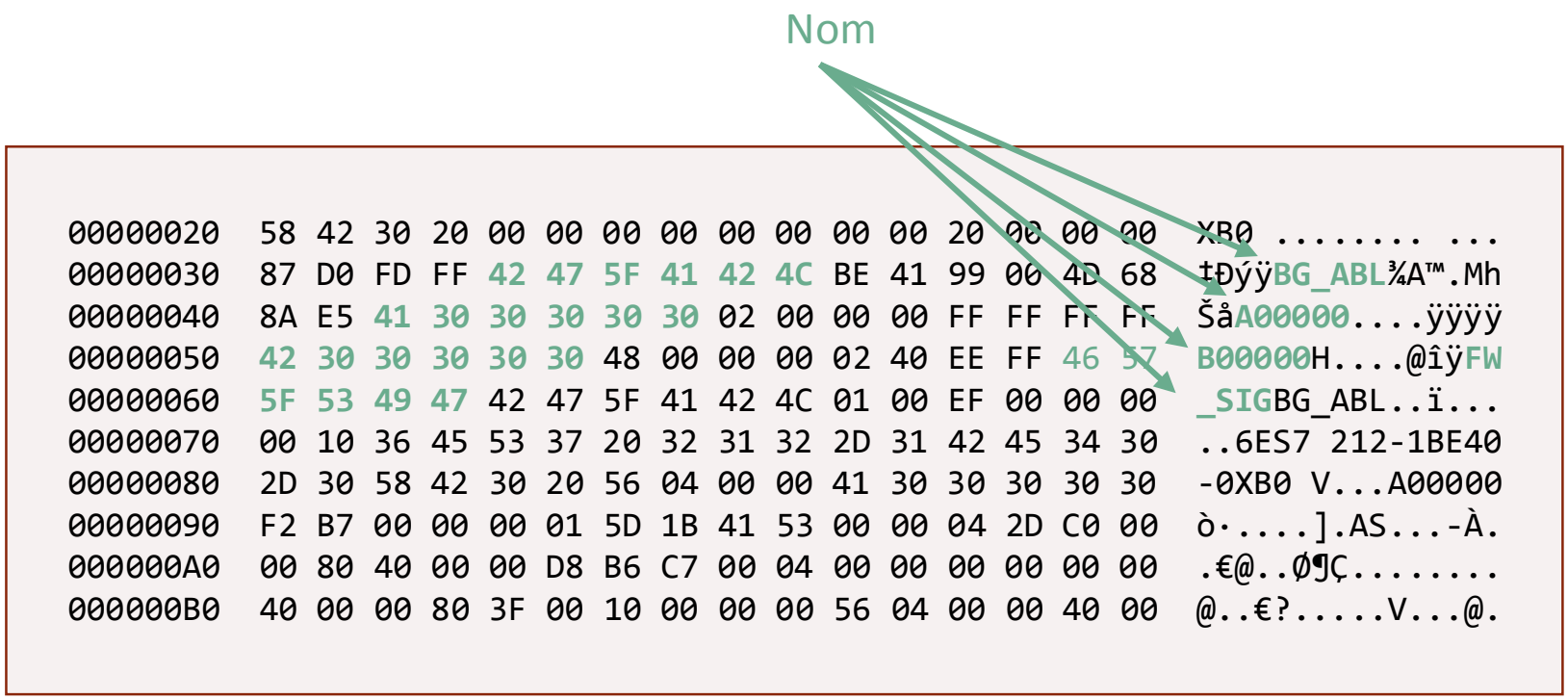
Accès au firmware

- Sur la NAND du PLC
- Sur le site de Web de l'équipementier : nécessite un compte valide

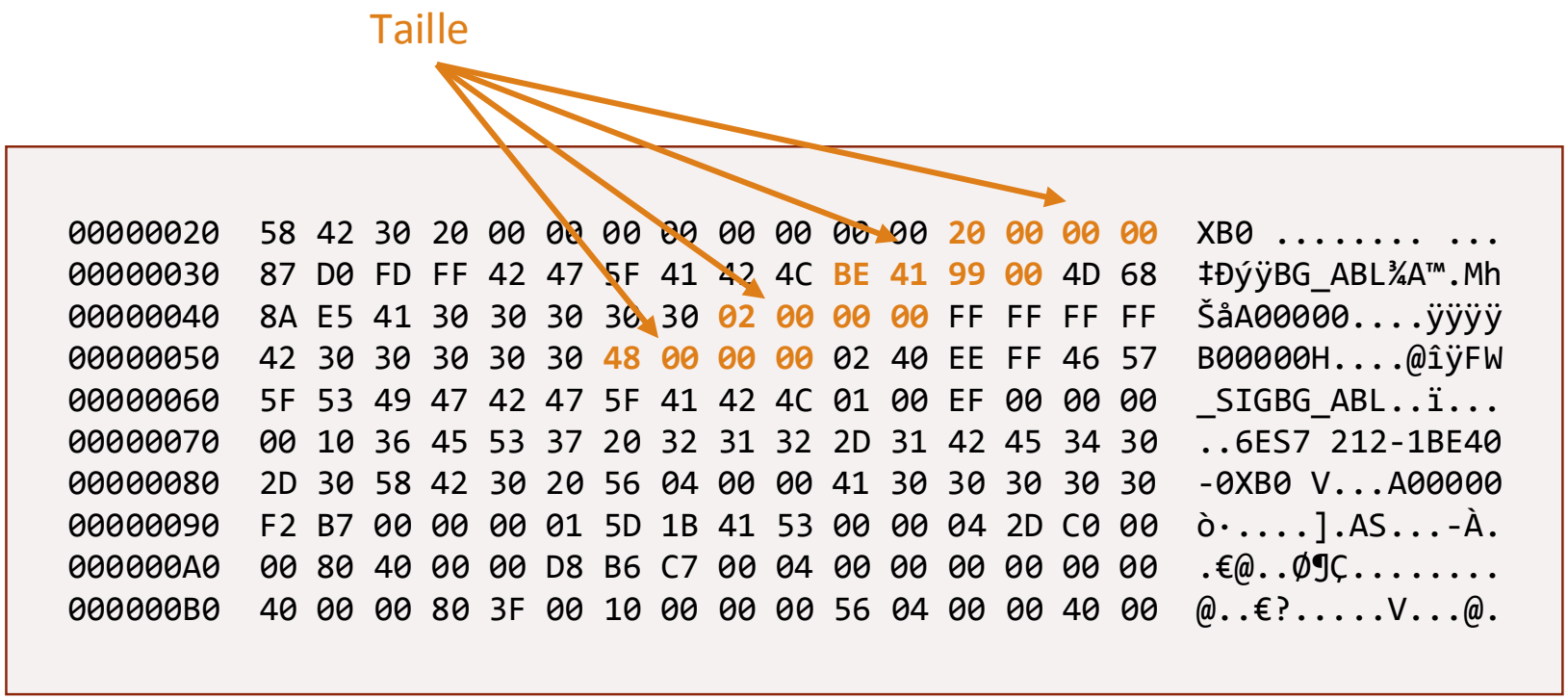
Mise à jour

- Mise à jour à travers le serveur Web ou une carte SD
- Code du firmware intégralement compressé
- Décompression réalisée par le firmware en cours d'exécution
  $\Rightarrow$ Décompression en boîte noire...

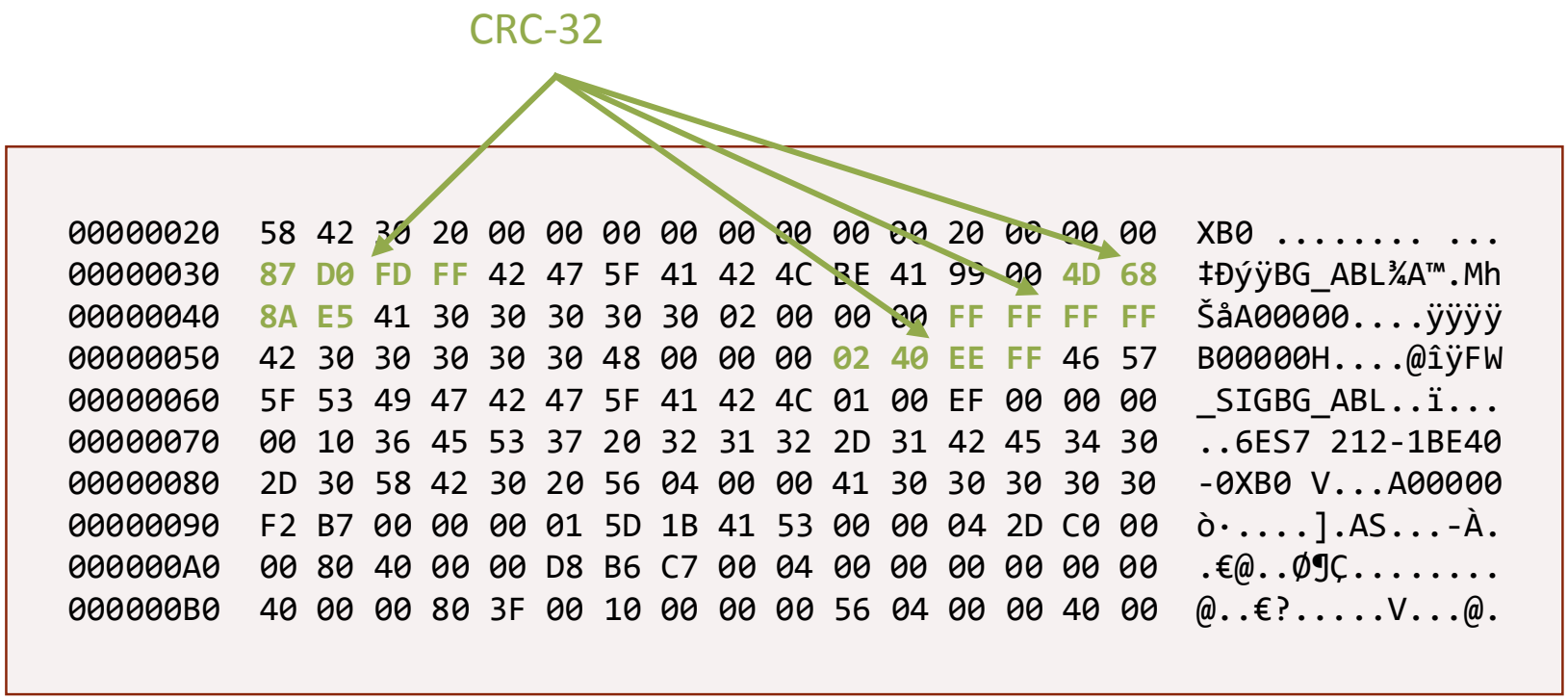
Recherche des en-têtes



Recherche des en-têtes



Recherche des en-têtes



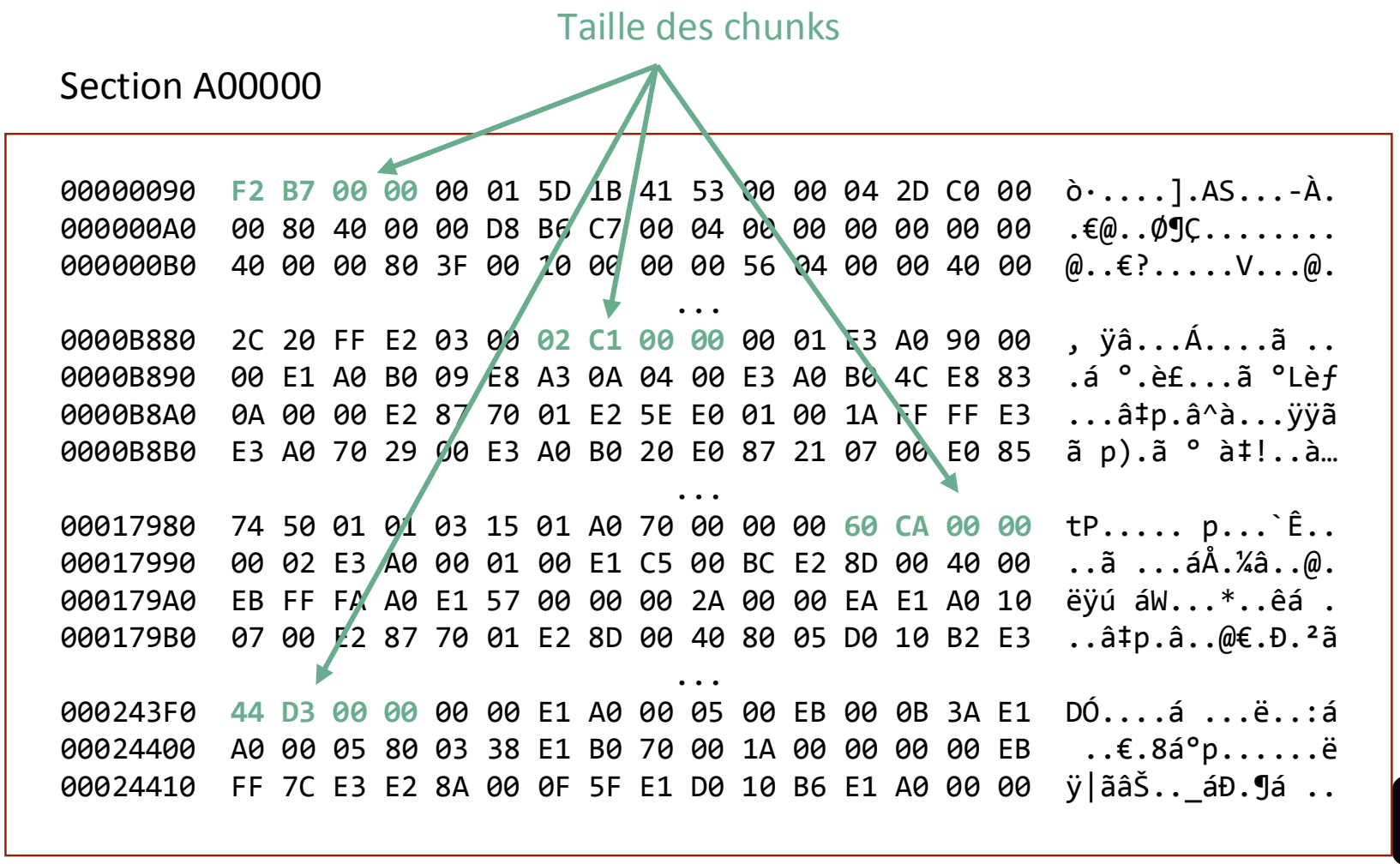
Recherche des en-têtes

```
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 20 32 31 32 2D 31 42 45 34 30 2D  ..6ES7 212-1BE40-
00000080  30 58 42 30 20 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 00 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...@.
```

BG_ABL

A00000

Organisation de la section de code



Zones faiblement compressées

```
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
```

Zones faiblement compressées

```
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
```

Zones faiblement compressées

Premier octet : masque

```
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
```

Zones faiblement compressées

Premier octet : masque

Octets rouges : longueur

```
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 :

- Blocs de 9 octets : un octet de masque, 8 octets de données
- Certaines données encodées par leur longueur
  - Longueur, mais pas de distance...
- La compression augmente progressivement à l'intérieur des chunks

$\Rightarrow$ Compression LZ

Compression : LZP

LZP

- Seul algorithme encodant seulement la longueur (WikiBooks)
- Improvement to dictionary coding/context coding
- 4 variantes. Ici, LZP3 est utilisé
- Pas d'implémentation publique fonctionnelle

Utilisation

- Décompresse chaque bloc de la section A00000. Chaque bloc fait 64 Ko
- Récupération du firmware en clair
- Un CRC-32 à la fin des données décompressées valide l'intégrité

Organisation mémoire

- Firmware décompressé : données brutes, pas un format connu
- Organisation mémoire décrite dans le binaire
- Utilisé par le bootloader
- Écriture d'un loader IDA pour charger correctement les données du firmware

Organisation mémoire

- Firmware décompressé : données brutes, pas un format connu
- Organisation mémoire décrite dans le binaire
- Utilisé par le bootloader
- Écriture d'un loader IDA pour charger correctement les données du firmware

Mauvaise nouvelle : l'obfuscation est toujours présente...

Signature du firmware

But

- Contourner le mécanisme de signature
- Injecter du code arbitraire

Vérification de signature

- ECDSA-256 avec SHA-256, courbe et générateur standard (ANSI X9.62 P-256)
- Tout le firmware est signé, sauf les 78 derniers octets (section FW_SIG, de taille fixe)
- Code propriétaire, correctement mis en œuvre. Nombres de taille fixe

$\Rightarrow$ Aucune vulnérabilité identifiée.

Travaux à venir

- Cryptographie boîte blanche
  - Authentification : clé privée du PLC. Une clé par modèle...
  - Chiffrement des programmes utilisateur (AES, plutôt simple)
- Meilleure compréhension du protocole.
  - Beaucoup d'informations dans le firmware
- Exécution de code
  - Injection et exécution de code arbitraire
  - Modification du comportement du code existant

Plan

Conclusion

Technologie pas encore mature du point de vue de la sécurité

- Erreurs dans l'utilisation des fonctions cryptographiques
- Vol des sessions rendu facile
- Schéma d'authentification non standard

De réels progrès

- Efforts pour concevoir un protocole sécurisé
- Bien meilleur que ce qui était précédemment utilisé
- Équipementier très réactif
- La sécurité semble être maintenant bien prise en compte

Questions ?

www.quarkslab.com

contact@quarkslab.com | @quarkslab.com