Design of mechanisms for filtering and isolations of industrial protocols

Peter Rouget

THÈSE POUR OBTENIR LE GRADE DE DOCTEUR DE L'UNIVERSITÉ DE MONTPELLIER

En : Génie Informatique, Automatique et Traitement du signal

École doctorale : Information, Structures, Systèmes

Unité de recherche : Systèmes Automatiques et Microélectroniques

Étude et conception de mécanismes de rupture et de filtrage de protocoles industriels

Présentée par Peter ROUGET
Le 29 août 2019

Sous la direction de Lionel TORRES et Pascal BENOIT

Devant le jury présidé par

Sébastien PILLEMENT, Professeur, Université de Nantes	Examinateur

Et composé de

David HELY, Maître de Conférence HDR, Grenoble INP	Rapporteur
Guy GOGNIAT, Professeur, Université Bretagne Sud	Rapporteur
Lionel TORRES, Professeur, Université de Montpellier	Directeur de Thèse
Pascal BENOIT, Maître de Conférence HDR, Université de Montpellier	Directeur de Thèse
Benoît BADRIGNANS, Directeur Technique, Seclab	Examinateur

UNIVERSITÉ DE MONTPELLIER

Abstract

With the rise of Industry 4.0, many infrastructures were forced to open their networks to the Internet, mainly to meet the growing need for supervision and remote control. But where these infrastructures were previously isolated, spared from external threats, their opening has caused the emergence of new threats, particularly network ones, which were not addressed and present serious risks.

Network cybersecurity solutions, like Firewalls, Intrusion Detection Systems or Intrusion Protection Systems are commonly used to address the concern of industrial infrastructures cybersecurity. However the trend of relying on software-based systems to ensure network protection brought to light the vulnerabilities of these systems, due to their inherent software implementation. Furthermore, the industry is tied to its own specificities (low-latency, support of specific network protocols), which are rarely covered by common IT solutions.

The main goal of this thesis is to study the use of FPGA-based devices applied to cybersecurity for industrial networks. Either as support for software-based security applications, or to perform critical network analysis operations. First it presents the industrial context, with control systems, their architectures, needs, implementation rules, specific protocols and also gives two examples of control systems as they can be found in the industry. Then it highlights the security problematic, with a description of the most common threats, case studies about their applications and impact in a control system, and discussions on the state of the art counter-measures available on the market. Through the establishment of a security target, it points the vulnerability of software elements and operating systems as well as the lack of process state aware security analysis.

To address these issues, we propose, through a first contribution, to enforce the security of the software system by taking advantage of existing FPGA's protection mechanisms. Finally, to answer specific application threats, we introduce an implementation of a patterns matching architecture with time and operational-process awareness, on FPGA. This thesis was conducted in collaboration between the Montpellier computer science, robotic and microelectronic laboratory (LIRMM) and the SECLAB company.

Résumé

Avec l'essor de l'Industrie 4.0, de nombreuses infrastructures ont été contraintes d'ouvrir leurs réseaux à Internet, principalement pour répondre au besoin croissant de supervision et de contrôle à distance. Mais là où ces infrastructures étaient auparavant isolées, épargnées par les menaces extérieures, leur ouverture a provoqué l'émergence de nouveaux risques, en particulier à travers le réseau, potentiellement sérieux et qui ne sont pas couverts.

Les solutions de cybersécurité, comme les pare-feux, les systèmes de détection d'intrusion ou les systèmes de protection contre les intrusions, sont couramment utilisés pour répondre aux préoccupations liées à la cybersécurité des infrastructures industrielles. Cependant, la tendance à se fier aux systèmes logiciels pour assurer la protection du réseau a mis en lumière les vulnérabilités de ces systèmes, en raison de leurs implémentations logicielles inhérentes. En outre, l'industrie est liée à ses propres spécificités (faible latence, support de protocoles réseaux spécifiques), qui sont rarement couvertes par les solutions informatiques communes.

L'objectif principal de cette thèse est d'étudier l'utilisation de dispositifs FPGA appliqués à la cybersécurité pour les réseaux industriels, soit comme support pour des applications de sécurité logicielle, soit pour effectuer des opérations critiques d'analyse réseau. Ce travail présente d'abord le contexte industriel, avec les systèmes de contrôle, leurs architectures, leurs besoins, les règles de mise en œuvre, les protocoles spécifiques et donne également deux exemples de systèmes de contrôle comme on peut en trouver dans l'industrie. Il met ensuite en lumière les problèmes de sécurité, avec une description des menaces les plus courantes, des études de cas sur leurs applications et leurs impacts dans un système de contrôle, et des discussions sur les contre-mesures de pointe disponibles sur le marché. Suite à l'établissement d'une cible de sécurité, nous mettrons en évidence la vulnérabilité des éléments logiciels et des systèmes d'exploitation. Nous verrons aussi comment l'absence d'analyse de sécurité tenant compte de l'état des processus peut mener à certaines vulnérabilités.

Pour pallier à ces problèmes, nous proposons, par une première contribution, de renforcer la sécurité des systèmes logiciels en tirant parti des mécanismes de protection existants du FPGA. Enfin, pour répondre à des menaces applicatives spécifiques, nous proposons la mise en œuvre d'une architecture de reconnaissance de motifs, sur FPGA, prenant en considération le cadre temporel et l'état du procédé industriel. Cette thèse a été réalisée en collaboration avec le Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier (LIRMM) et la société SECLAB.

Acknowledgements

First of all, I would like to thanks my advisors, Pascal Benoit and Lionel Torres for their support and guidance throughout this Ph.D. I have learned a lot from conducting researches as from giving presentations. Thanks to Benoît Badrignans for his accompaniment during this research work and his help on many technical problems.
I thank all my coworkers who believed in me and all those who did not believe it, thank you for this warm atmosphere that allowed me to stay motivated.
Finally, I thank my family: my beloved wife Melody, my parents Christine and Daniel, my step mother Lina for their unfailing love and encouragements, and my friends: Loic, Guillaume, Yohan, Geoffroy and Mathieu for all these good moments of laughter.
This thesis was brought and supported by SECLAB whom I thank for this opportunity and for the resources they put at my disposal.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Nowadays, industry needs a high level of automation and control, whether in production environments but also in transport, energy, all require fast and reliable exchanges to perform their duties. Dedicated networks which achieve these needs are referred as industrial networks and even if they share common features with IT networks, their needs and goals are very different. The main problematic of industrial networks is *availability above all*, the worst nightmare of an industrial is losing control of its own process or being unable to access a specific equipment. In consequence those networks generally focus on availability, with low latency exchanges and a strong deterministic requirement. All these constraints make industrial networks a delicate environment where safety is the key word in order to allow a smooth operating.

With the increased connectivity of industrial control systems, numerous threats and vulnerabilities which were unknown to these systems appeared. Their criticality but also the potential consequences made them targets of choice for a wide variety of criminal organizations. Threats are generally addressed using common IT counter-measures which, although effective, lack the application or operational intelligence. In 2015 the Trojan BlackEnergy [38] plunged hundreds of thousands of Ukrainians into the dark for several hours. The attackers used spearfishing to introduce the trojan into the company network and then spread it until getting a remote control access to the industrial process and cutting the power. In addition to highlighting a lack of staff's awareness about common network threats, this attack showed that the industrial process was not ready to face application threat. The fact that a remote order asking to shut down circuit breakers, potentially leading to power failure, was not detected nor considered suspicious shows a vital need for operational oriented security devices. In this context, dedicated solutions came on the market to address this concern, but even so it is still badly addressed. Furthermore, most of the solutions are pure software applications which present flaws and vulnerabilities inherited from the software components. The aim of my work, during this

thesis, was to enforce the security of industrial control system against operational threats, through extensive usage of hardware resources. We plan to increase the security of existing network devices through boot chain protection and suggest the appeal of a dedicated hardware patterns matching to harden the security process.

This work is structured around two major contributions:

- First, it addresses protection against persistent threats for network security software application. Through usage of FPGAs, a chain of verification of each boot element by the previous one ensures that the running software application is authentic and not compromised.
- Secondly, a hardware process-oriented patterns matching engine on FPGA is proposed, instead of securing software security application, we replace it by a hardware one. It allows to perform security operations on the application level, similar to dedicated solutions of the market, but with enforced security offered by pure hardware.

Chapter 2 introduces the industrial context by describing the industrial network, its components, architecture and problematic followed by case studies. Then, this chapter addresses the security problematic with vulnerabilities, threats and attacks as well as the possible consequences through examples. We then speak of the existing counter-measures, with their advantages and disadvantages, before building the security target that we intend to address. Chapter 3 is about the first contribution, we discuss the boot process of an Linux operating system on a System on Chip FPGA's platform, with security issues and existing counter-measures. And then we introduce the contribution in two steps, with its basic concept, architecture and with a second optimized proposition as well as the results of experiments. Chapter 4 addresses the second contribution, about hardware patterns matching, firstly by discussing on industrial protocols breakage then on hardware patterns matching implementation recommendations. Then a versatile patterns matching solution is suggested, in both concept and architecture, before implementation experiments and results. Finally, we conclude this thesis through discussions about further works.

Chapter 2

Industrial Networks

Historically Industry has always attracted malicious intents as it presents important economic and human stakes. Where industrial infrastructures were isolated, separated from others, and so hard to access and attack, today needs for remote control and supervision [24] have required the opening of their networks to the Internet along with the appearance of new problematic and threats. The emergence of connected objects is one of many reasons these infrastructures "open their network access". But as these networks have specific needs and peculiarities which differ from Information Technology's standard, network cybersecurity solutions are rarely sufficient to answer to these emerging threats. In order to understand these needs and to propose adequate solutions, it is necessary to understand what is an Industrial network.

2.1 Definitions

Industrial Network is a common term used to qualify a network whose usage is dedicated toward industrial purposes [2]. It is generally architected in 3 levels, the SCADA ensures supervision and remote control over automatons which control the operating field level. The manufacturing process is automated, then controlled and supervised through Control Systems, which may be potentially geographically distant. The following sections highlight and discuss the peculiarities of this type of network, whether in terms of equipments, traffic, architecture and implementation or in terms of vulnerabilities and protection.

2.1.1 Information Technology (IT) versus Operational Technology (OT)

It exists a wide variety of industrial contexts for use of industrial networks, but they are generally used when machinery needs monitoring, remote control or automation. Despite this diversity and different sets of requirements, great implementation principles stay unchanged. Resources and devices must stay available, industrial environment imposes a strict control to ensure no failures or accidents because their severity is far more serious. Furthermore, industrial networks are real time, so communications must respect fixed delays, with deterministic answer times. These needs are reflected by a network architecture different of common IT one. The following work describes in more details these specificities, from the network architecture, traffic and its components, to implementation problematic and failures severity.

Components and Devices

Industrial networks are made of specific devices whose functions are centered around the application needs and the manufacturing process. It is to ensure the communication

between and toward these equipments that is built the industrial network. Following is a list of the primary devices which constitutes these networks, with their functions, while Figure 2.1 locates them according to the levels of the control system.

Human-Machine Interface (HMI) is a hardware and software element allowing human operators to visualize and control a process through modifying settings, objectives, sending requests, override automatic controls, emergency stops. It is the primary communication interface between a human and the automated process. The control engineer can access process information, status and historic as well as directly querying controllers. HMIs are generally set up on the application or supervision layer of the network, on various platforms, like laptops, dedicated desktops, browsers and even through Internet. Exchanges between equipments, part of the industrial system, are not made to be understandable by humans. On the fieldbus level, they are logical values (1 or 0) and electrical levels, and the more it goes up in the architecture, the more they become addresses and values. It would require tremendous knowledge about the network architecture, devices configurations and characteristics to be able to understand these exchanges. The HMI gives the necessary amount of information to the user with graphical representations and allows him to simply act on the process.

Data Historian stores process information, operations and logs of the control network. It is a database which collects and stores information through the network. These data can be used for statistical analysis about the control process as well as production planning. In case of attacks or failures, stored logs are precious for diagnostics and upgrades.

Programmable Logic Controllers (PLCs) are small industrial computers running custom and dedicated Operation Systems. It is one of the primary controller resources of industrial networks with the ability to control complex and automated processes. It is very versatile as it offers numeric and analog I/Os, time control, digital interfaces. It controls actuators, sensors and the field process. PLCs are configurable devices, programmable through a dedicated interface, located on engineering workstations of the Local Area Network (LAN). Users will implement specific functions, stored in the user-space memory of the devices, while backed up on Data Historians.

Remote Terminal Unit (RTU) is a data acquisition and control field device whose purpose is to support SCADA remote stations. It is basically a PLC designed for specific control applications. It is generally deployed in low connectivity environments, with wireless or radio communications, both being features rarely present on industrial equipments unless specific cases.

Intelligent Electronic Devices (IEDs) are "smart" field devices able to perform data acquisition, network communications and local processing. It is intended to be relatively autonomous, to perform simple processing based on acquired data, without the need of

a PLC, and to refer directly to Control Servers. These devices are generally close to the field process if not in it.

Supervisory control and data acquisition (SCADA) is a control system, used by industrial processes, which uses network data and HMI for high level supervision, as well as PLC, RTU and IED to interface with the process. It is a privileged solution to interface devices from different manufacturers allowing access through standard automation protocols. It is one of the most commonly-used type of industrial control system, used in power plants, railways, water distribution, gas and oil as well as in variety of factories. SCADAs are designed to abstract communications and data collect, from the fieldbus and other controllable devices, and to display it understandably (graphically or textually) to the operators. It allows, as well, actions, operations, modifications of the process through its interface. It is a mixed solution made of software applications and hardware components. Generally speaking, a Master Terminal Unit collects and stores information from Control Centers, while PLCs and RTUs control and monitor devices on the field bus. IEDs directly converse with the SCADA master unit. The combination of software and hardware allows data exchanges back and forth between the SCADA, PLCs and RTUs with a control over the permitted actions (allowed operations, data ranges, values). The SCADA acts as a high level understandable representation of a whole system, potentially geographically extensive, for supervision and punctual actions, while the regular industrial process is automated by PLCs, RTUs and IEDs.

Distributed Control System (DCS) is a control system for industrial process, with autonomous controllers distributed along the system and a central operator supervisory control. It is very similar to SCADA but more focused on continuous process control with local control instances. DCSs are made for control systems within the same geographical location, they are made for continuous control and supervision, through usage of similar devices to the SCADA (PLC,RTU,IED) but with lesser automation.

Control Server hosts SCADA and DCS applications. It is basically a medium to access low level fieldbus devices as well as subordinates control modules and other devices inside the control system.

Furthermore common network devices are present into the industrial environment, such as routers and modems for interconnection and routing in the network, firewalls for security purpose, access points for wireless data exchanges and servers for data storage and calculation.

Industrial networks rely on specific devices on each level for its control system, while the key cores of the automated process are PLC, RTU and IED. The environment made of server, data historian, HMI and DCS allows to make a complete remote controlled and supervised process. But this is the specificities of all these equipments and their needs, in

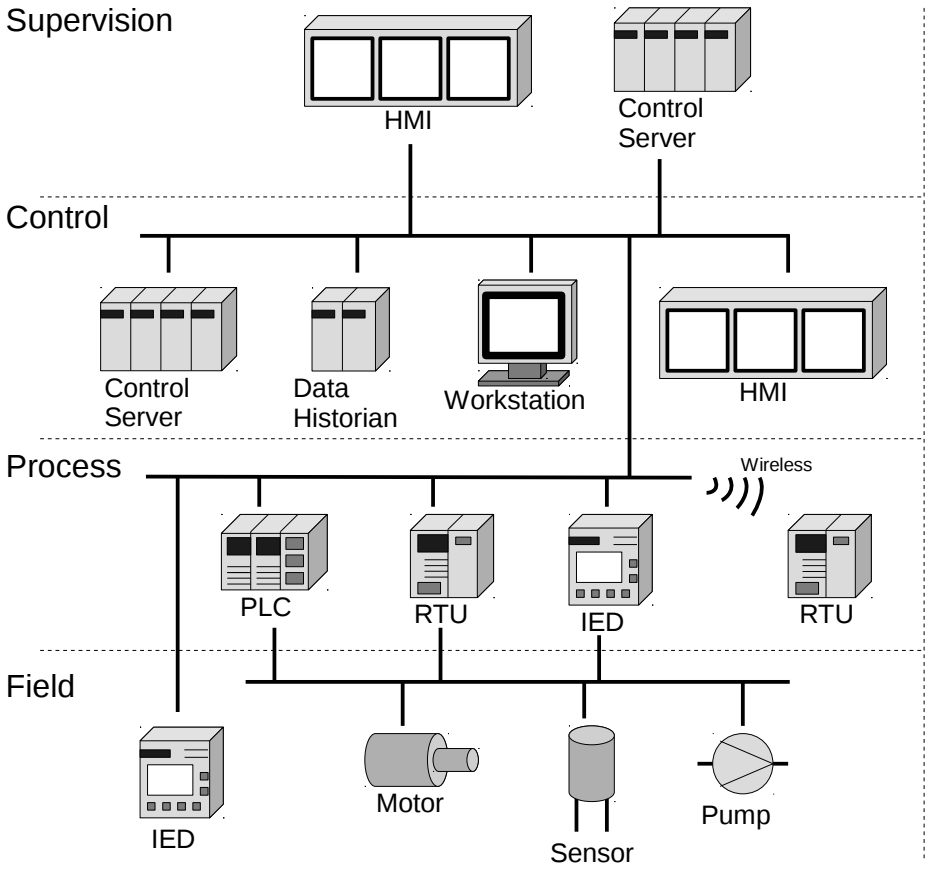


Figure 2.1: Industrial Network Architecture

terms of control and proximity to the field, that will influence the network architecture and explain the differences compared to common IT architectures. And as this architecture differ, the attacks methods and associated vulnerabilities evolve accordingly.

Network Architecture

Industrial control network architectures are much more hierarchical that common Commercial networks. Organized in layers, generally 3 or 4, each corresponding to one process level, they are interconnected through gateways, routers and modems. The connection between instruments, actuators and controllers is made on the lower layer, then controllers interconnections, followed by HMIs and control commands and finally the upper layer for Supervision and remote control. Unlike Commercial Networks which are more flattened and made of interconnected LANs. This difference is explained by the multiple

connections and communication mediums of devices in the industrial control system. Field devices generally discuss through serial protocols or raw analog data exchanges, they are rarely directly connected to the network (except for IEDs). Instead, they are connected to PLCs and RTUs which are able to use common network protocols (TCP/UDP over Ethernet for example). Furthermore Control systems and supervision are commonly geographically distant and it is not uncommon that they are connected through the internet.

To give a representation of the complete company network, with interconnections between Industrial and IT networks, the most common standard is the Purdue Enterprise Reference Architecture or Purdue Model [81]. This is a reference architecture which gives a model of the network through multiple layers and stages. In Figure 2.2 we give an example of a classical Purdue Model for a company which owns an administrative network and an industrial control network.
The levels 5 and 4 are dedicated to the Enterprise zone with Intranet, emails and Internet connection while the lower levels are for Industrial purposes.
We find in layer 0 the field level, the level 1 is dedicated to automated control, with PLCs and RTUs, while the level 2 is the first layer of process control, with control command and HMIs.
The layer 3 is for supervision, it hosts the manufacturing process of the company. Both networks (industrial and IT) are generally connected, through firewalls, to a Demilitarized Zone (DMZ) which allows communication and embedded most application servers and services of the company.

But beyond a dedicated network architecture, Industrial networks are characterized by specific implementation problematic which prioritizes the **Quality of Service over the security**. Security devices will need to take account these constraints, as they will face network architectures which were purposely not made taking the security into account.

Implementation

Industrial process requires good reactivity due to being in direct contact with physical equipments, as well as it needs to take decisions and send control in real time. These requirements resulted in changing the primary needs of the network while the Quality of Service (QoS) became dedicated on availability. [50] presents several requirements of industrial networks in comparison with IT networks, but following are what we consider as the two major implementation rules of these networks.

Availability is one, if not the first, implementation rule of industrial networks, the concept of security is predominantly associated to safety operating. It means that resources

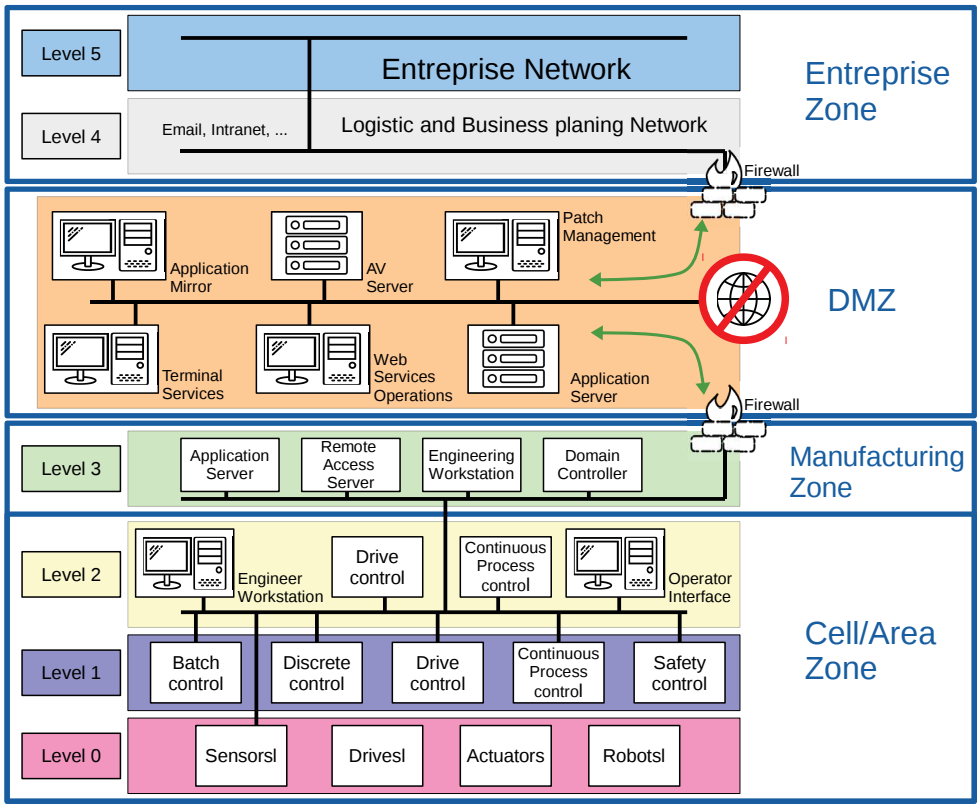


Figure 2.2: Purdue Model

must stay available, remain controllable and observable. Being unable to access a device or to properly monitor the process, may lead to failures. Due to the fact that industrial control networks are directly interfaced with physical devices, the failures severity is far more greater. Availability in an industrial control system means devices availability, equipments must be robust and keep working in any cases. If a hardware or software failure occurs, the device is not available for configuration, maintenance as well as for the process. That is why, Industrials often store spare equipments to be able to quickly restore availability.

The network, its architecture and state, may also impact and degrade the availability. A ill-conceived network architecture or overloader network will cause increasing latencies and congestion, while dedicated attacks, such as denial of services ([82]), can completely sever accesses to devices. But resorting to high availability architectures with redundancy, loadbalancing methods and adequate security allow to reduce the impact of these problematics on the network availability.

- For the process to be deterministic means that devices state must be predictable

based on their Inputs and Outputs, operations and behaviors must be consistent under the same conditions. It is a criteria of good construction and operation of the industrial control system, the typical working flow should stay predictable and must not present inconsistencies.

- Regarding the network, Industrial control systems are real time, so it must have bounded latencies and low variances. The time between a transmission and its reply should be low and predictable. Furthermore, where IT networks accept high data loss and retransmissions, it must not occur inside industrial networks as it will greatly degrade the determinism.

As QoS principles fundamentally differ from IT networks, network exchanges need to be adapted for this purpose with priority to short messages with high efficiency.

Network Traffic

In order to satisfy availability and determinism needs, data exchanges inside an industrial network prevail efficiency and low latencies. The physical medium is, in many case, common Ethernet, in accordance with widely used transport protocols such as TCP and UDP, but serial line as well as industrial Ethernet, and in some uncommon cases wireless, are also in used. Beyond those layers, and focusing more particularly over TCP/UDP over Ethernet, data are expressed through specific communication protocols which consist of small messages, only a few bytes long. Such transmissions are often single binary value, up to one or to bytes registers, the lower in the network architecture, the smaller packets are. It is in complete opposition with network exchanges within IT networks which are far bigger with regularly kilobytes of data sent.

The common network traffic of an Industrial System is deterministic, it consists of commands or periodically sampled data. But one particularity of these networks is the existence of aperiodic events such as alerts, failures or production rushes which cause sudden changes in the network traffic. It is called an "avalanche" and is characterized by a sudden and very intensive network traffic with maximum priority. When it happens, the availability of the network and the devices is more than ever critical. Even if unusual, these events are to be taken in account when dimensioning the industrial process and control system as it is fundamental for network equipments to support it, even with aging technologies.

Failure

In common IT networks, failures (criminal or accidental) can induce various consequences, from network or resources unavailability, devices compromise, data leakages. It usually

results on financial losses but rarely material or human. For industrial control systems, this remains true, but, as the network is closely linked to the industrial process, any failure directly impacts devices underneath. It is translated by heavy material and potential human losses, and in some uncommon cases industrial disaster.
Taking as example a power plant, a failure can mean malfunction of a relay or a power coupling device, locally speaking it causes material losses. But when taking into account the power shortage ensuing, it implies tremendous financial consequences, potential human losses, operating losses for surrounding companies, traffic accidents and more.

Environment

Industrials devices and machineries working conditions are very harsh as it can include high temperatures, high humidity, vibrations, dust or electromagnetic waves. Networks devices integrated in these environments need to match drastic working constraints. Whether in the design (welding, components choice) or in integration (protection against fluids, dust), it contrasts strongly to commercial networks which are located in clean, temperature controlled environments.

In fact industrial and IT networks are fundamentally different, as devices and QoS priorities differ the whole network communications are mend to adapt to these needs. Furthermore industry imposes very uncommon and harsh conditions, on the contrary to the well known air-conditioned network rooms. Among the multiplicity of domains and industrial applications which require the use of industrial control systems, a small number of infrastructures presents a critical risk for the security of a country. These infrastructures are referred as "Critical infrastructures" and their control systems are referred as "Critical networks".

2.1.2 Critical Network

The term "Critical infrastructure" is evolving but it was used for the first time in terms of National Security (President of the US 1996). One of the definition given by the Commission of the European Communities (European Union 2008) is : " *An asset, system of part thereof located in Member States which are essential for the maintenance of vital societal functions, health, safety, security, economic or social well-being of people, and the disruption or destruction of which would have a significant impact in a Member State as a result of the failure to maintain those functions*". Among existing networks, European critical infrastructures classification includes (Commission of the European Communities 2004):

- **Energy**: It addresses all the infrastructures that produce energy, whether it is electricity, oil or natural gas. This category includes for example coal centrals, nuclear

plants, dams, as well as drilling wells and oil platforms. But in addition to primary production sites it also covers processing and distribution infrastructures. All the other sectors depend on the energy one, as they virtually all need electricity, thus making the energy a key point of all critical infrastructures.

- **Communications and information**: This sector is central to the economy, academia, security, health and many others. In the era of the "All Connected" where more and more services rely on communication infrastructures this sector criticality never ceased to increase.
- **Finance**: The challenges of this sector are easily judged as it covers all economic assets, publics and privates.
- **Health care**: It covers a nation's healthcare and public health infrastructures such as public and private clinics, pharmaceutical companies, and distribution circuits. This sector protects all others from hazards but as most of its assets are owned by private parties it requires great collaboration and information sharing.
- **Food**: Food distribution and production is a vital asset for the well being and health of a population. Associated infrastructures come from agricultural sector but also transformation and distribution assets such as factories, shops and group catering companies.
- **Water**: Drinkable water is one of the first necessity goods, if not more important than food, but this sector addresses as well wastewater storage and treatment as it is vital for diseases prevention.
- **Transport**: The transportation sector includes aviation, freight and passenger rails, maritime transport, postal and road infrastructures. It presents an huge economic stake as well as a convenience of use for the population.
- **Production, storage and transport of dangerous goods**: Many sectors such a Energy and Health care reject dangerous wastes that need to be handle with caution to avoid potential dangers to the population and the environment. This sector covers mostly infrastructures needed for the transportation and storage of these wastes.
- **Government**: And finally one of the most obvious critical sector addresses governmental assets, whether it is military infrastructures, administrative buildings and schools.

Despite multiple other definitions, Critical infrastructures play an essential role in all main functions of modern society, [3] provides further information and researches on

these infrastructures. Most of it relies on an automated control process which inherited the criticality of the infrastructure, in this case we speak of Critical networks. These networks answer to the same problematic and specificities as defined previously but with a criticality level way higher which justifies a greater interest in safety. This work focuses mainly on these particular networks as they require stronger security measures.

As industrial networks fundamental principles target availability and determinism, communications and data exchanges between network devices are designed to promote these needs, hence the use of dedicated network communication protocols. But like in IT's context, the communication protocol is another risk factor in a network and to measure this risk it is necessary to understand how these protocols are constructed and how do they work.

2.2 Industrial Communication Protocols

Communication protocols are sets of rules and syntax used by interlocutors to standardize information exchanges. Today communication systems use well-defined protocols, in network communications based on Ethernet, which is the physical medium, we found the physical layer (MAC) then in most cases IP and TCP (or UDP) both being well known communication standards. In the industrial context, networks rely on dedicated communication protocols, either above the TCP layer or using completely different standards and medium.

To communicate, equipments of the industrial network use industrial protocols. They are communication protocols which were created and specified to address industrial needs. It exists industrial protocols over serial line, Ethernet, TCP, UDP or virtually over any layer of the OSI model (Figure 2.3), but in the field of this work we will center the study over TCP/UDP protocols. Figure 2.4 shows a timeline of release of the main industrial protocols still used in the industry. Looking at the OSI Model Figure 2.3, industrial protocols to which we will be interested take place on the upper layers (Session, Presentation, Application), among these some are proprietary (generally made by PLC vendors) and other are open source. As years go by, numerous industrial protocols were introduced, but in the field of this work we will present four of them, presented below, ModBus, S7, BacNet and OPC-UA, with a major focus of the first two.

We chose those Modbus and S7 because they are both very common in the industry and have no concern about security. Even so they are old and tend to be replaced by more recent ones, in newly made industrial networks, existing networks which use those protocols are mend to last for many years before being updated. One major point of industrial networks, and more particularly for critical one, is "if it still works there is no reasons to

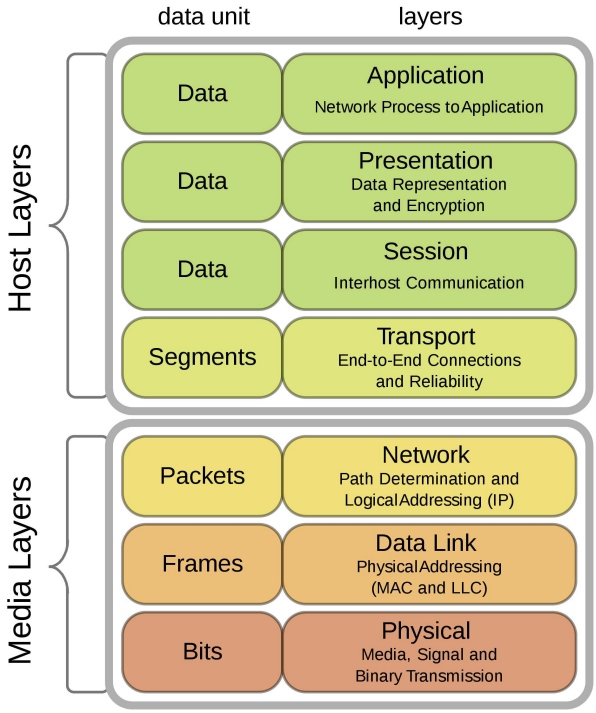


Figure 2.3: OSI Model

change it", which means equipments will not be changed as they are still working and even so there is always some identical spare equipments for quick replacement. For an industrial replacing its equipments by different or newer models is a complex and expensive operation. It would require, in prior, months of test on a simulation network which emulates the real one. Then it would be necessary to stop a part or the totality of the industrial process to perform the change. Those problems are even more true on critical industrial processes. For these reasons it is important to address the security of existing industrial protocols without systematically resorting to changes for newer and more secure ones.

2.2.1 ModBus

ModBus [39] is a communication protocol created in 1979 by Modicon, to address communication between industrial equipments, such as PLC and DCS, over serial lines. Due to its simplicity yet efficiency it became the de-facto standard for automated industry, but at the time it was designed, security and networks interconnection were not existing problematics. It allows a master to communicate with many slaves (up to 240 devices in the ModBus system), the master initiates the connection then performs a request and the targeted slave answers. ModBus allows variety of medium such as serial line, wireless but the most common types are :

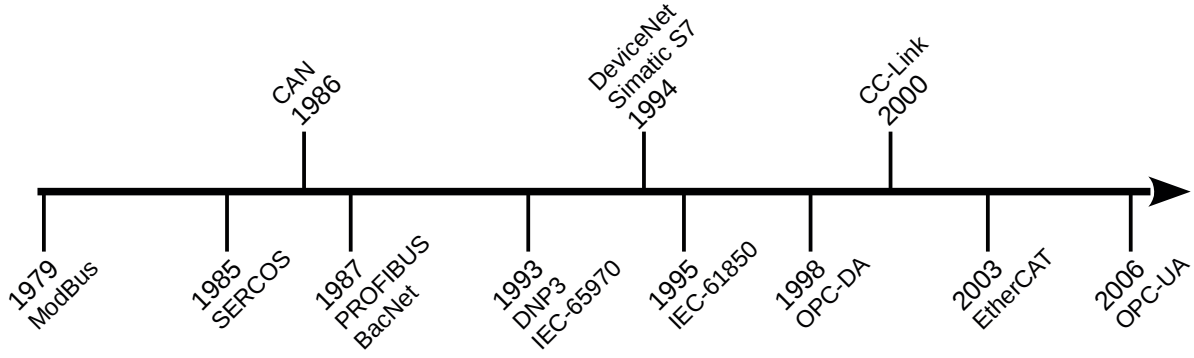


Figure 2.4: Chronology of major Industrial Protocols release

- *ModBus ASCII*, messages are coded in hexadecimal, using 4-bit ASCII characters. It is the slowest of the modBus protocols, each byte of information needing two communication bytes. Therefore it is privileged for wireless communication, such as Radio (RF) or telephone modems, because it allows rather long time intervals between characters wihtout causing errors.

- *ModBus RTU* is the ModBus standard for serial line, twisted pair links, and the more popular of the ModBus protocols. The most common speeds are 9,600 and 19,200 baud.

- *ModBus TCP* is simply ModBus over Ethernet using TCP/IP standard, the data is encapsulated inside a TCP/IP packet. Any Ethernet network is able to support this ModBus standard making it very popular among more recent industrial control systems where the Ethernet norm became standard. For these reasons the following work is focused over ModBus TCP but remains true in some part for all modBus standard as the Memory model and principle are shared by all.

Over TCP the addressing between equipment is done using the Ethernet and TCP/IP, through MAC Address and IP address, the default communication port should always be the 502. The Unit identifier present in the ModBus packet is not always used, as it was necessary for serial communication, depending of the device some use it as supplementary check for addressing. The modBus Application Data Unit (ADU) is maximum 260 Bytes long, it consists of 253 bytes of Modbus protocol data and 7 bytes specific for Modbus TCP addressing. Figure 2.5 shows the complete ModBus ADU which contains the following fields :

- *Transaction Identifier* serves for synchronization purpose, the request and the answer share the same transaction identifier allowing the master to match both.

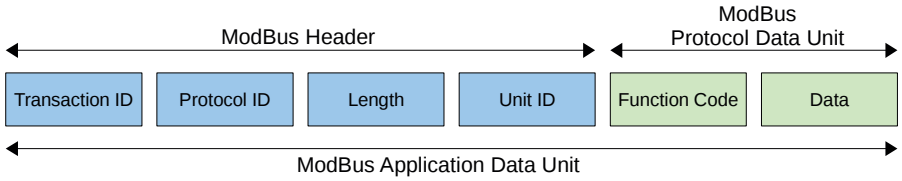


Figure 2.5: ModBus TCP Application Data Unit

- *Protocol Identifier* is a field set to 0, unused in ModBus TCP.
- *Length* indicates the number of remaining bytes in the modBus message. It is a common type of field in network protocols, allowing devices to know how many bytes they must expect in the frame.
- *Unit Identifier*, natively ModBus TCP devices ignore this field, setting it to 255. But in some cases of composite equipments (ModBus TCP to ModBus RTU gateway) it is used to give the slave address of the targeted device.
- *Function Code* is a one byte hexadecimal value from 0x01 to 0x02B which codes the operation related to the modBus message. The basic operations, read and write, are performed over single or multiple elements, and followed by the address, and eventually a data in case of writing. The corresponding answer, from the slave, uses the same function code with the requested information or acknowledgement. ModBus allows exception code for return, differentiated by an offset of 0x80, supplementary to the function code, used to indicate a processing error or a unknown request. Further functions are available (Figure 2.1) mainly for management, control, update, and unused function codes are reserved to the user for specific needs.

The ModBus memory model consists on 4 basic types of data which are : Coil (single bit, read-write), Holding Register (16-bit word, read-write), Input Register (16-bit word, read-only) and Discret Input (single bit, read-only). Generally a device has 4 memory spaces known as Blocks which work as memory pages, each one managing one type of data. Internal data (coil, register, input) accessible by their address, are either directly mapped to the PLC I/O or used to perform combinatorial operations as expected in the automaton program.

2.2.2 S7

S7 is a proprietary protocol, over TCP, belonging to Siemens. Similar to Modbus, it is a request/reply protocol with one master and multiple slaves. One of its specificity is that

Function Code (hex)	Operation
01	Read Coils
02	Read Discrete Inputs
03	Read Holding Registers
04	Read Input Register
05	Write Single Coil
06	Write Single Register
07	Read Exception Status
08	Diagnostic
0B	Get Com Event Counter
0C	Get Com Event Log
0F	Write Multiple Coils
10	Write Multiple Registers
11	Report Server ID
14	Read File Record
15	Write File Record
16	Mask Write Register
17	Read/Write Multiple Registers
18	Read FIFO Queue
2B	Read Device Identification

Table 2.1: ModBus TCP Function Codes

multiple motherboards can be installed in the same device (PLC). It requires supplementary routing information, in the form of rack and slot numbers, to address the correct target. Considering that it is the standard protocol for Siemens PLCs, it is widely used. More recent PLCs now use S7+ which is an upgraded version of S7 including security features. S7 ADU consists in a S7 PDU wrapped in TPKT [46] and ISO-COTP protocol which is define in RFC1006 [44], based on the ISO8073 protocol [45]. The S7 PDU 2.6 is made of:

- *Header* which contains protocol constants and references as well a length information.
- *Parameters* are supplementary data necessary for the correct application of the operation, such as addresses.
- *Data* carries the data relative to the operation and parameters, typically memory values, firmware, memory blocks.

The Header is 10 to 12 bytes long and embeds :

- *Protocol Identifier*, constant always set to 0x32.

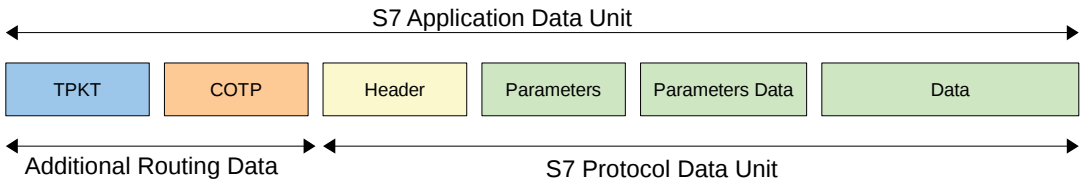


Figure 2.6: S7 TCP Application Data Unit

- *Message Type* is a first indication of the requested operation, combined with the Parameters it forms the complete operation identifier.
- *Reserved*, data always set to 0 and probably ignored or reserved for future use.
- *PDU Reference* works as a tracking number used by the master to link the request and the answer.
- *Parameter Length* is the length of the Parameter field.
- *Data Length* is the length of the Data field.

Additionally, the header may contain Error Code and Class in case of failure, this field is not mandatory. The rest of the S7 message, Data and Parameters, content depend on the Message Type. The S7 memory model works using Db (for database), those are memory pages which data are addressed inside either individually (through address) or multiple, up to the whole page. This protocol shares many common points with ModBus, in the header many fields have similar functions or meaning, and in the way of addressing data.

2.2.3 BacNet

BacNet [41] is a communication protocol for Building Automation and Control, supported and maintained by the American Society of Heating, Refrigerating and Air-Conditioning Engineers (ASHRAE). It is a very popular ISO standard, especially in America and Europe. BacNet was designed to allow communication of building automation and control systems for applications such as heating, ventilating, and air-conditioning, lighting control, access control and fire detection systems and their associated equipments. A BacNet equipment is defined by an Object identifier composed of an object type and an instance number. In the same way, a bacnet equipment possesses a number of object defined by object name and identifier, while those objects possess parameters. The ANSI/ASHRAE 135-2016 defines 60 standard object types among which are Calendar, Lift and many others, but it also allows the user to define its own. In short, a bacnet equipment is like a

meta-structure embedding a number a lower structures, each one made of multiple fields. One of the consequences of this construction is that the protocol is very verbose and requires lots of text strings to ask for an operation. Due to its use, often confined to building automation, Bacnet has very low concern toward security.

The BacNet network has its own representation made of interconnected devices, it leans on the following concepts:

- The *segment* is made of devices connected, eventually through repeaters, under the same physical layer.
- The *network* is made of connected segments, homogeneous of the bonding layer.
- The *internetwork* is made of connected BacNet network, through routers, with no constraints over physical and bonding layers.

The addressing in the BacNet protocol appeals a number of mechanisms:

- On the physical layer, it depends of the specified technology.
- On the network layer, it uses a combination of MAC address and network identifier. Multiple devices inside the same BacNet network must share the address domain, else they must be divided between multiple networks with different network identifiers.
- On the application layer, the addressing uses the common addressing mechanism of the BacNet protocol, the Object Identifier.

Bacnet is a more recent protocol and, unlike Modbus and S7, its data model departs from a proximity to the material to a more abstract representation with complex objects having multiple properties. This difference is also felt in the message construction, data is intended to cross multiple bacnet *networks* or *internetworks* with a wider variety of functions, which explains a more complex protocol structure. Figure 2.7 shows the representation of a typical bacnet packet, it does not cover all possible cases but gives a solid example of how are organized the fields in the bacnet message. The Bacnet Virtual Link Control helps to identify the type of Bacnet protocol we are using (Bacnet IP for example) and gives a first indication on the message type. It is in the NPDU that are found most of the routing data, with the Control field that indicated which data will appear in the remaining NPDU. And in the APDU we find the remaining message type indicator while the rest of the follow-up differs in function of this type.

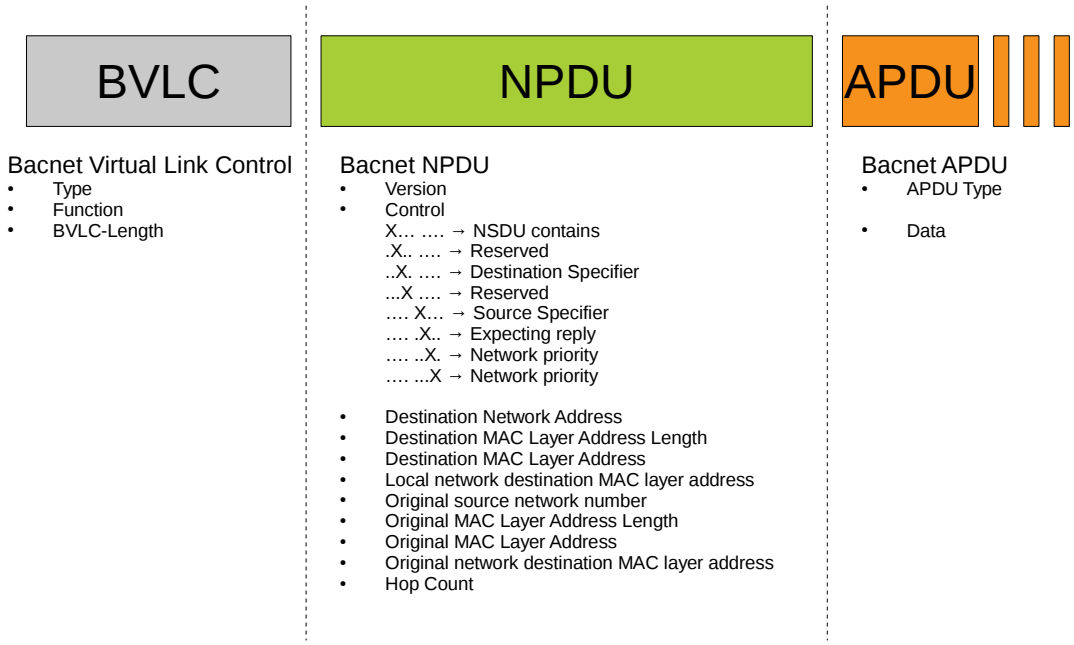


Figure 2.7: Bacnet IP packet structure

2.2.4 OPC-UA

OPC-UA [42] is a machine to machine communication protocol for industrial automation developed by the OPC Foundation and released in 2006. It was designed to be the new generation of industrial protocols, including multi platforms support, security features, flexibility and scalability toward the network architecture and devices. OPC-UA uses concepts from recent object oriented programming, it works with a so-called Full Mesh Network using nodes: a node is an entity embedding any kind of data or metadata, it can be data, attributes (Read access), as well as complex commands or events. It allows OPC-UA devices to manipulate abstract objects but it also allows more complex operations than in most of the common industrial protocols. Devices can perform basic operations as well as Subscription to an object, discovery, heartbeats. Its versatility makes it a privileged choice for new industrial standards, it is usable in nearly all industrial contexts (energy, transport, manufacturing) as it addresses needs from the managing level to the lower level of the operational process.

2.2.5 Profinet

Even if it will not be discussed in this work, it is interesting to mention Profinet [40] as it is one of the most popular protocol, the most used in the automotive sector. It is an open

communication standard for industrial automaton, created by PROFIBUS and PROFINET in 2001, and designed to address tight time constraints in data delivery. It is optimal over industrial Ethernet with support of TCP/IP, while its versatility allows integration of variety of devices from operating to management layer. Profinet leans on three protocol levels :

- TCP/IP for non-critical data with moderate time constraints.
- Real Time with Profinet IO for short time constraints.
- Isochronous Real Time with Profinet IO is the fastest protocol level for very tight time constraints.

Profinet relies on a combination of MAC address, IP address and device station name for addressing while the connection establishment is made using UDP.

2.2.6 Synthesis

According to [48], among existing industrial protocols, ModBus is one of the most popular, 42% of usage, it is well known and easy to use, making it a standard adopted by most industrial controllers. Profinet follows with 29% of usage and Bacnet is a bit less popular with 9% of usage, because it is mainly used for building automating and not much else. S7 is not represented in the study, this proprietary protocol is fairly discrete in its information disclosure, but because it is a standard for most of the Siemens PLCs, we can assume that it is quite common. Today with the emergence of new protocols, more versatile and secure, the old ones are destined to be replaced but it will take time.

In terms of security it appears unsurprisingly that older protocols, such as ModBus and BacNet, do very little or no state of security. In fact these protocols were designed at a time when the cybersecurity context was almost nonexistent and as a result they are particularly exposed to cyber attacks. Today most recent industrial protocols make good progress in order to integrate many security features such as encryption and authentication mechanisms. Among them we can find OPC-UA which is the most famous and others like S7+ which is a newly and more secured version of the Siemens's protocol. These protocols, although still not widespread, are much more resistant to attacks. Table 2.2 summarize the specificities of each industrial protocol previously addressed.

Industrial protocols listed before are examples of what can be find in today industrial networks, it is not exhaustive, as it exists numerous other ones. In the field of this work we will focus on S7, Modbus and will keep in mind Bacnet and OPC-UA for extensible parts of the study.

Industrial Protocols	Security	Flexibility	Interoperability
ModBus	Null, modbus integrates no concepts toward Authentication and Confidentiality.	The protocol is closely linked to the memory model and information must match one of the basic data types.	It does not handle interoperability with other protocols or network types.
S7	Null, as modbus, S7 takes no account of Authentication and Confidentiality.	The protocol is closely linked to the memory model and information must match one of the basic data types.	It does not handle interoperability with other protocols or network types.
BacNet	This protocol takes not care toward Authentication and Confidentiality.	It offers possibility for the user to define its own object types and labels, through string, even beyond existing specifications.	It does not handle interoperability with other protocols or network types.
OPC-UA	It offers many features toward security with encryption and challenge responses.	OPC-UA's data model is fully customizable by the user, allowing to declare its own data types.	This protocol was design to be compatible with administrative networks and many data representation standards (XML,JYSON).

Table 2.2: Industrial protocols specifications synthesis

2.3 Threats

The target of this thesis is to study the threats and risks which can affect an industrial control system and then to work on counter-measures in accordance with the state of the art. To do so it is essential to classify and understand the vulnerabilities and associated attacks these systems are exposed to. The vulnerability of common network infrastructures is not to be proven yet, and [4] shows that industrial networks are not spared. The definition of the various threats which endangered these networks is orchestrated on multiple layers.

Firstly, considering the hardware platform, it concerns the physical integrity of the

	Vulnerability	Impact
Network	Inadequate log mechanisms	Poor detection of failures and attacks.
	Missing security devices	Open security breaches and access points for attackers.
	Misconfigured security devices	Allows transmission of unnecessary or dangerous data, leading to data leakage or malwares spreading.
	Inadequate network architecture	Increase the possibility of security flaws.
	Inadequate Security Perimeter	Unknow security flaws and missconfiguration of security devices.
	Unpatched security Vulnerabilities	Vulnerability to known threats and exploits.
Application	Communication protocols Integrity	Protocol Messages compromise and malicious modifications.
	Communication protocols Authentication	Facilitate messages interception and impersonification of interlocutors.
	Unpatched security Vulnerabilities	Vulnerability to known threats and exploits.
	Lack of dedicated security solutions	Not adapted and unsufficient security.

Table 2.3: Table of Threats and Vulnerabilities

devices as well as the control of the peoples inside the infrastructure and the security policies, excluding network related ones. The access control as well as the physical protection of the devices is one of the crucial points if not the first which may concern an infrastructure. It includes protecting the power supply and protection against Radio and Electro-magnetic interferences. Physical access to equipments is a grave problem, USB ports are often left accessible, and a malicious USB device can easily cause damages. The lack of backup or redundant equipments as well as non-existing test facilities are aggravating factors. Security policies play a critical role in the vulnerability to the threats of the devices. Password policy, remote access protection, configuration's backup and unHardened OS or applications are representative weaknesses of wrong security policies. And obviously, unpatched security vulnerabilities are the most evident threats. Event if critical these threats will not be discussed further as they concern problematic not addressed in this work.

Table 2.3 summarizes the threats which concern either the network or the applications. These are addressed in more details, with their consequences, below, starting with the network ones:

- *Inadequate log mechanisms*: the use of logs from network equipments (security equipments as well as process devices) is one of the most efficient way to detect network

failures or attacks. Furthermore, after the occurrence, they are useful to track causes and consequences. The non emission of logs from a device generally results of a bad configuration, but missing storages and non monitoring of these logs are also problematical.

- *Missing or misconfigured security devices*: The lack of security equipments, or their misconfigurations, allows transmission of unnecessary or dangerous data. It opens security vulnerabilities which may lead to malware spreading, data leakage and any other potentials attacks. It is the most obvious and common security threat as it is based on the lack of security.

- *Inadequate Network architecture*: Inadequate or poorly built network architecture offers more security flaws, makes more difficult to define proper security perimeter and policy. Furthermore it can impact the operational process as it may induce loopholes and delays.

- *Inadequate Security perimeter*: The security perimeter, or Information Systems Security Policy (ISSP), is used to validate that necessary security controls are deployed, it defines which are the assets and devices needing protection, against what and how they need to be protected. Without a proper one it may lead to unknown security flaws. Furthermore, it increases the difficulty to properly configure security devices.

- *Unpatched security vulnerabilities*: One of the scourge of the industrial network cybersecurity are the existing, known but unpatched vulnerabilities. Industrial infrastructures have a big inertia toward security updates as the standard delay between the publication of a security patch and its implementation on the network is between 3 to 6 months, when the patch is accepted, which is not always the case. It is common for those patches to be voluntary ignored as long as their effects on the whole network is not fully known. Furthermore, some vulnerabilities require scientific monitoring to be aware of and patched, this work is not always done as it needs specific skills and time. The problem of updates is also related to how they can be applied. Not all assets are necessarily connected to a network. Some updates can only be deployed through USB, with the security problems inherent to this protocol.

- *Denial of Services*: Denial Of Services or Distributed Denial Of Services are very common attacks where a targeted device or network is flooded by network traffic to render it unavailable. It is a security threat that targets specifically one of the major problematic of the industrial network, the availability of the resources.

These threats are a sample of the most common threats targeting a network, as new

ones emerge every day, they are not specific to industrial networks and affect all types of network.

Furthermore the communication protocol used from the presentation of the application layer of the OSI model brings its own vulnerabilities.

- Industrial protocols address availability and determinism of data exchanges in the industrial network, they rarely integrate security features such a encryption or authentication. It facilitates greatly the work of attackers as message exchanges are easy to intercept (often in clear text mode), understand, and it becomes easy to impersonate one of the interlocutors.
- Some Industrial protocols have their own integrity checking, through hash and Cyclic Redundancy Check (Modbus), but most of them rest on the transport layer integrity checking. It facilitates the compromise of a protocol's message.
- As well as for network devices, RTUs and PLCs are rarely updated, even for security updates. As they directly handle the industrial process, any patch or update that may change their functions or behaviors is regarded more dangerous that the security breach itself.
- The network security of industrial control systems is generally handled by common IT solutions (firewalls). But even if these devices are able to analyse the network traffic, it rarely goes up to the application layer. Understanding and analysing the application layer of the network packet is fundamental to assess the incoming threat. The commercial state of the art lacks of dedicated industrial network supervision devices (there are still companies that offer these kinds of solutions, but they are rare).

These threats are common to most of the industrial networks, but each industrial protocol brings its own sets of exploits [6], depending of its specifications and implementations. Simple protocols, such as Modbus and S7 are particularly vulnerable to fuzzing, as they are relatively compact and made of small sized fields, this basic brute force techniques brings results with relatively low implementation complexity. [5] illustrates this example and gives a complete taxonomy of Modbus flaws and attacks. The more complex is the protocol, the more the attacks complexity increases [7].

2.3.1 Incident Scenarios

From the vulnerable machine, left inadvertently accessible (open session), to complex attacks involving propagation through other networks and remote control, passing through

the insertion of a malicious program, it exists numerous scenarios with increasing degrees of difficulty. Following are chosen failures, or incident scenarios, for Industrial Control Systems, based on the previously listed vulnerabilities and the threats we intend to address:

- **Man in the middle**: A malicious entity manages to access the network between the Control System and the PLCs/RTUs. Using simple network manipulation techniques (ARP spoofing), it is able to usurp both the identity of the Control and one or more PLC. So data and commands sent by the control system to destination of the PLC will be intercepted by the attacker and the same goes for answers from the PLC to the control. From this point multiple possibilities are open:
  - Simply listening and analyzing the network traffic. The data received are transferred to their legitimate recipient, the attacker operates a simple translation without modification, but it allows to retrieve information about the functions of the process and map the network. It is a stage of learning and discovery.
  - Modification of information sent by the control. The attacker will intercept the messages sent by the control and modify their content. It can lead to potential damages on the targeted device, modifications or serious malfunctions of the operational process. To correctly perform, it needs knowledge about the current process and devices as well as about the industrial protocol used, else it is possible to perform random alterations on the data, leading to unknown behaviors.
  - False information sent to the control system. In this case, data sent back to the controller are intercepted and modified. Without correct information the controller will not perform needed operations in case of alert or failure. Furthermore it can be used to cover malicious changes or actions on the PLCs/RTUs.
- **Denial of Service**. The network traffic is disrupted, slowed down or interrupted, on a targeted device or on the whole network. It impacts the availability and causes a denial of resources. It is a very common attack, particularly for IT network, as it requires relatively low skills and resources.
- **Unauthorized modification of the program** of any components of the control system (DCS, SCADA) or the process (PLC, RTU). A malicious entity manages to modify the program of a device inside the Industrial network, related to the process. It can be achieved through automated malware, transmitted using wireless devices (portable phone, computer), mass storage (USB key, hard drive), pishing

campagne, or through remote access. Doing so, it is possible to change the process behavior, damage equipments, slow down or stop the production. This method was used in the well know Stuxnet [1] case, on a centrifuge, leading to accelerated degradation of the device.

In general, regardless of the scenario of attack, the hacker needs to make an inventory (mapping, discovery, scan) of the network architecture and devices. Then he will seek to perform at least one of the following actions :

- Modify a file or a program (directly or not).
- Send wrong or malicious information to its target.
- Observe or register the behaviors of the network and devices, or the impacts of its own actions.

2.4 Case Study

To illustrate industrial network architectures, protocols and definitions, following are two examples of industrial context where control systems are commonly used. The first describe a electric post which is a component of a power plant control system. Power generation networks being wide and geographically ramified it commonly resorts to control system to supervise and manage the whole process. The majority of companies that manufacture large quantities of products, dedicated to marketing, resort to automated production systems based on one or more control systems. In this context, the second example describes a bottle filling production line, an automated control system that can be found in many food and beverage industries. Neither of the two are a complete industrial process, they are steps, positions, of the whole process which is itself composed of many. In the case of an automated production line for example, it can count several dozen of steps. Both examples are followed by a representation of the address plan by the industrial protocol and superficial security analysis.

2.4.1 Electric Post

A power plant is a complex aggregation of multiple devices, peoples and places, its representation if far too complicated to be addressed in this work, a case study on an electric post is proposed. This post is made of multiple physical implantations, geographically different and constituted of controllable elementary objects:

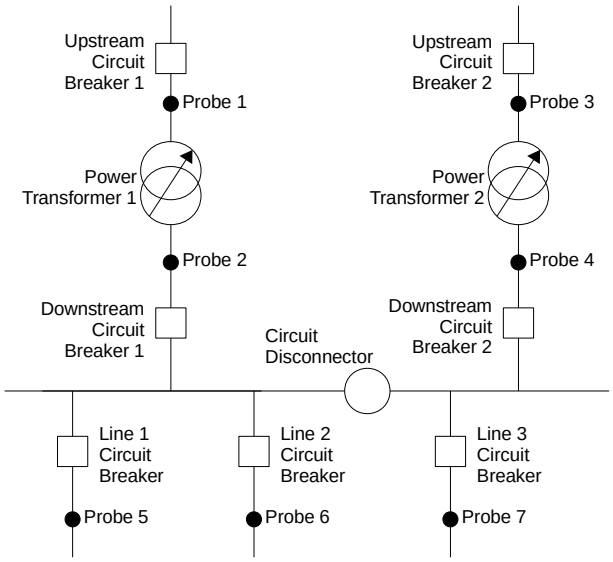


Figure 2.8: Example of an electrical Post

- **Circuit Breaker and disconnector** are breaking organs remotely controllable and searchable. They have generally two controls for state changes and two for reading the state. Circuit breakers are able to break the power while disconnectors can not.
- **Power transformers** are controllable power transformers with an impulse command to vary the output voltage.
- **Voltage probe** allows to measure voltage.
- **State probe**, an electric post includes a variety of other probes and sensors to ensure the correct behavior of the process, from fire detectors, state probes, default pass sensors, failure detectors, and many others.

The initial state of the electrical post is made of two power distributions, each power transformer is framed by an upstream and downstream circuit breaker. Each power line has its own circuit breaker and both power distributions are separated by a disconnector, see figure 2.8 for a representation of the post. It is locally operated, then refers to multiple distant supervisions of different levels.

Following are rules and scenarios concerning the electric post:

- Upstream and downstream circuit breakers must be opened when working on power transformers (maintenance operations, replacement).

- The downstream circuit breaker can not be closed, while the upstream one is open, to prevent power trace back.
- Circuit disconnectors can not be opened while charged.
- Circuit breakers and disconnectors must not be closed more than 2 times per minute.

All these rules exist to prevent failures and malfunctions on the electric post, they are part of the controllers program and are known by any supervision instance.

In terms on industrial protocol usage, IEC61850 [78] and ModBus are both the most used in electric power systems, in the context of this work we will use ModBus to give an example of representation and addressing from the control system point of view. Table 2.4 summarizes the representation of each controllable component through ModBus addressing, with their functions, data type they are mapped on and a memory address map proposal. In the industry, circuits breakers and disconnectors have one control input modeled by a coil (refer to ModBus data model) to open or close and one status output modeled also by a coil. These inputs and outputs are logical values, either 0 or 1. Voltage probes only have one analogic output with the measured value, generally mapped to one register. Power transformers have two control inputs, one logical to turn it on and off, and an analogic one to specify the output level, the logical one mapped to a coil while the analogic to a register. Then, one logical output for the status mapped to a coil, the register used to specify the output level is readable too. Generally speaking, probes and sensors among the control systems have one output, either logic or numeric, to return the measured value or state. When working on complex industrial process, the control part is split between multiple control devices, such as PLCs, RTUs and IEDs along the process, each driving its own set of actuators, sensors, motors. In our case study, due to the simplicity of the example, one control device is enough to drive it all, but to give better understanding we split the process in control groups, each representing a dedicated part of the control. These control groups have either a dedicated control device or one for all of them.

Rules and dependencies between registers, coils, inputs and outputs, in terms of value and state, are either handled inside the controller (PLC/RTU) program or by the upper level of control.

- The first case allows a higher resilience toward application attacks, under condition that the controller program is cleverly written.
- While the second case lowers the load on the local control device but increases the exposure to man in the middle attacks.

Control Group	Equipment	Function	ModBus Type	Address
Ctrl 1	Upstream Circuit Breaker 1	Open/Close	Coil	0x00
		Status	Coil	0x01
	Probe 1	Value	Input Register	0x00
	Power Transformmer 1	On/Off	Coil	0x02
		Set output power level	Holding Register	0x00
		Status	Coil	0x03
	Probe 2	Value	Input Register	0x01
	Downstream Circuit Breaker 1	Open/Close	Coil	0x04
		Status	Coil	0x05
Ctrl 2	Upstream Circuit Breaker 2	Open/Close	Coil	0x10
		Status	Coil	0x11
	Probe 3	Value	Input Register	0x10
	Power Transformmer 2	On/Off	Coil	0x12
		Set output power level	Holding Register	0x10
		Status	Coil	0x13
	Probe 4	Value	Input Register	0x11
	Downstream Circuit Breaker 2	Open/Close	Coil	0x14
		Status	Coil	0x15
Ctrl 3	Circuit Disconnector	Open/Close	Coil	0x20
		Status	Coil	0x21
Ctrl 4	Line 1 Circuit Breaker	Open/Close	Coil	0x30
		Status	Coil	0x31
	Probe 5	Value	Input Register	0x30
Ctrl 5	Line 2 Circuit Breaker	Open/Close	Coil	0x40
		Status	Coil	0x41
	Probe 6	Value	Input Register	0x40
Ctrl 6	Line 3 Circuit Breaker	Open/Close	Coil	0x50
		Status	Coil	0x51
	Probe 7	Value	Input Register	0x50

Table 2.4: ModBus Control Table of the Electric Post

On this kind of industrial example, there are two application attacks which are easy to consider:

- The first concerns interactions between devices, such as circuit breakers, where two or more of them must have conditional states to each others. For example, two circuit breakers, *Downstream Circuit Breaker 1* and *Downstream Circuit Breaker 2* should not be in open state both at the same time because it would cause power break. This type of attack is very easy to perform with minimal knowledge of the process, as it can be made through fuzzing or random commands.

- The second attack requires deep knowledge of the current process, as it involves attacking the power transformer by setting its output power level outside of the functional levels of the power grid. It may cause equipments destruction, power failures, and this is hard to counter unless the controller program is well written and allows to set limits on memory values (which is not the case on many commercial controllers).

From this example of industrial process which concerns the energy and public resources we will now discuss of another very popular process which is used as well in both private and public industries, the automated production line.

2.4.2 Bottle filling line

Most of the industries which resort to automated production lines, have a dedicated Control System which supervises and ensures the smooth running of the production. There is no defined model for these lines, as each industrial has its own Control System with its own architecture to answer its needs. But generally, these processes make intensive use of PLCs and RTUs to manage a wide variety of sensors, probes, motors, valves, actuators. To illustrate it, we propose the representation of a generic bottle filling line, as it can be found in any wine, water, juice or soda factory. It is made of a conveyor to carry empty bottles under the filling system, this one uses a solenoid or a pump to deliver the liquid, sensors to control the amount of available liquid, and either level sensors, flow sensors, or both to check the filling level of the bottles.

In practice, the process is composed of :

- A sensor allowing to know the amount of available liquid for the pump in the tank. It can be a unique probe, with only one readable state which indicates if the critical level of liquid is reached, or a multitude, allowing to know with more details the level of liquid.

- A pump or solenoid with an impulse command to control the flow.

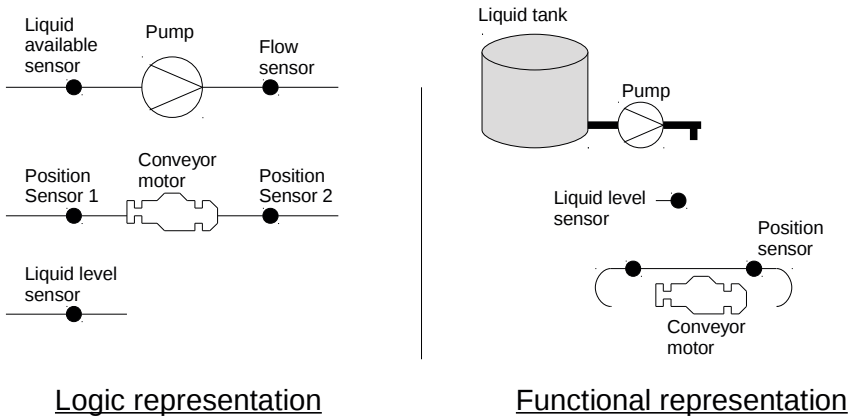


Figure 2.9: Example of Bottle Filling Line

- A flow sensor, it measures the flow that has passed through, to know the volume of elapsed liquid. Typically, it has a readable status with the flow metering, a reset control and eventually a writable level cap to send alerts. It is also possible to imagine liquid level sensors inside the bottle to detect filling level.
- A conveyor, to carry bottles on the filling line a mean of transport is necessary. Taking as example a belt conveyor, it is generally controllable through a motor with either impulse or analog level command to control speed and move. To know with precision the movements of the conveyor, in order to determine the position of the bottle, various methods exist :
  - To extrapolate the position using the conveyor characteristics, prior experiments and knowing the control command sent.
  - To use hall effect or mechanical sensors on the motor frame or conveyor mechanisms.
  - To use various kind of sensors (mechanical, infrared) to detect and determine the positions of the elements transported by the conveyor.

This kind of line is an example of what can be found in many industries when speaking of production line, it induces lots of steps each with its own set of devices and automated processes. But as the diversity of control systems and automated production processes is explained by the fact that each answers to specific needs, and every need is different, these processes present a wide diversity of industrial protocols. For the purpose of matching with the problematic of this work we will use ModBus to further detail this bottle filling line control architecture but, in the same way, S7, OPC-UA and various other protocols could be used.

Equipment	Function	ModBus Type	Address
Liquid Available Sensor	Value	Input Register	0x00
Pump	On/Off	Coil	0x00
Flow Sensor	Value	Input Register	0x01
Position Sensor 1	Value	Input Register	0x02
Position Sensor 2	Value	Input Register	0x03
Conveyor Motor	Set Rotation Speed	Holding Register	0x00
Liquid Level Sensor	Value	Input Register	0x04

Table 2.5: ModBus Control Table of the Bottle filling line

As in the previous example, basic sensors, whatever their type, give only an output value. This information is mapped on a coil, when it is logical, or on an input register when speaking of analog value, such as voltage and temperature. Electric motors such as the conveyor one, are indexed in two categories: Alternative Current (AC) and Direct Current (DC). AC motor rotation speed is controlled through the frequency of the input supply while DC motor rotation speed depends of the level of the input supply current. In both cases the speed control is symbolized by a register, motors don't require an on/off control input as a non-null supply current fulfills this function. For pumps too it exists more than one kind, basic one only requires one control input for on/off (supply current or not) with a non-controllable and constant output flow. While more complex pumps allow flow control, in the same way as for motors rotation speed. In our case, we will take into account of a basic pump with only one control input mapped on a coil. See table 2.5 for ModBus mapping control table of the current bottles filling line.

Due to the simplicity of the example there is not much of application attacks available regarding only one component of the process, only the conveyor belt motor offers the possibility to stop or increase the rotation speed causing discrepancies and failures. But the production line correct function is heavily based on the interactions and timing between devices. When the position sensor gives the right value the conveyor motor should stop, then the pump must be activated for the liquid to flow until the liquid level sensor reaches the right level. It is in the smooth running of each step that the attack possibly appears. Activating the pump randomly or falsifying sensors values through man in the middle attacks are both easy to deploy application attacks.

While simple, both previous examples, Electric post and Bottles filling line, give good

understanding of how industrial networks are architected to allow automated processing and remote control. It also gives an idea of the diversity of industrial control systems in the industry, as nearly all kind of applications make use of it. But while we show attacks possibilities and vulnerabilities in these examples, the following section provides state of the art analysis on protection mechanisms to address these attacks.

2.5 Protections

Today Industrial networks are complete part of every day's life, as most of the industries and infrastructures rely on them. While their problematic and implementations differ from classical IT networks, with dedicated components and protocols, they are still exposed to common challenges. But industrial networks vulnerabilities rely on the application layer as well with specific problematics which require dedicated solutions.

2.5.1 Isolation

Among the techniques used by industrials to secure their networks this one is the most common and most simple one, while it may impose heavy constraints. It consists in isolating the network from the Internet or other connected networks. The isolation can be physical, without any network gateways, relying on the use of a physical media to transfer data, or logical through specific equipments as diodes, data-diodes. While it seems efficient, in reality it poses three great problems:

- It requires a good control over the human factor. From access control to the control of physical medias (usb, mobile phones), any data, device and person that enter the network needs to be controlled.
- It increases the difficulty to retrieve information and logs from the network, which requires additional measures, such as a dedicated person who checks manually logs and alarms.
- It is based on the postulate that if it's disconnected it is not accessible. In fact the state of the art of hacking techniques shows that it is wrong, nowadays it is possible to exchange data with an unconnected computer using side channels (EM, light, sound).

Beyond accessibility problems, updates have to be taken into consideration, as it represents a necessary access and potential threats to the control system.

On the other hand, isolating the network increases the difficulty for an attacker and requires little network security equipments. This method was popular and efficient many

years ago when wireless connectivity was not so common. Today the popularity of IOT inside industrial networks, promotes the setting of networks made of a multiplicity of small interconnected devices, often wireless, and complicates heavily the control over data isolation. Furthermore, the generalization of smartphones and portable computers is a challenge for physical control of devices in contact with isolated equipments. Physical isolation is still interesting and used in highly critical infrastructures where the supplementary cost and organization are already necessary (access control, staff audit), such as power plants and nuclear plants. The efforts needed to ensure complete isolation are lower than the cost in case of a potential failure/attack, but they remain specific cases and physical isolation is generally a deprecated method to ensure industrial control system security. Despite this, logical isolation stays a privileged solution in numerous infrastructures [51].

2.5.2 Firewall

Firewalls are the most common way to ensure security over a network. They are equipments which scan the network traffic, monitor and authorize applications and flows. It typically looks at the IP/port of the TCP header or other transport data such as MAC address. Additionally, firewalls process routing rules to manage the network. Firewalls, as they exist in IT networks, work as a first security barrier which protects against a majority of networks threats, but it is not sufficient when addressing protocol specific threats. Specific industrial firewalls are able to look at the payload of the network packet to perform analysis at the application layer, but as we know these are quite uncommon. It allows to cover the flaws about specific industrial applications of common IT firewalls.

Among all the firewalls on the market we can notably find the company Check Point [54], Fortinet [55], Cisco with their ASA solution [56] as well as products that target the industrial environment such as Stormshield [57] and Tofino [58]. But these solutions are proprietary and deployed turnkey, so they offer little if no space for research. In this context it is worth mentioning open source firewalls such as netfilter [79], pfsense [80] or iptable [59], the one used natively by Linux, as they offer possibilities for work, such as [52] and [53].

One of the points that is worth mentioning when speaking about firewalls is the implementation criteria. It is said that there are commonly two types of firewall, the software and the hardware one. The software is generally an application running on a host machine, either a dedicated one or on each workstation, sitting in the network. While the hardware one is a dedicated device which sits directly behind the router or the network interconnection and is a point of passage for network traffic. In practice, both firewall types are software applications running on different platforms. True hardware firewall

which performs network security analysis without resorting to software means, with pure hardware resources, are very poorly represented outside of the research [61] [62].

One of the popular techniques that firewalls use to perform threats recognition is patterns matching. The incoming data is compared with a set of rules or signature to find a corresponding match. This method is quite efficient, even if depending of the used implementation some errors may occur, as it allows to quickly detect incoming threats. But it is based on a list of previously identified threats and there is a type of attack called *0-day* which means a new or recent one, not yet identified, studied and classified. The time for those threats to be added to the list of patterns checked by the firewall represents a window of attack not insignificant. Furthermore, security rules of the firewall are written and entered by an administrator, which inevitably induces a human risk factor.

It is worth mentioning that besides security, firewalls offer features for network architecture and management, such as redirection or network address translation.

2.5.3 IPS / IDS

Intrusion detection and protection systems are dedicated applications whose work is to detect and prevent incoming threats. They are very similar to firewalls, they perform analysis on incoming data, but are typically designed to work on the payload of the packets to perform finer analysis. Furthermore these applications are generally designed to work with a SIEM and send abundance of logs. Nowadays with the improvement of firewalls, IDS, IPS and firewalls tend to be grouped under the acronym UTM for United Threats Management and are increasingly difficult to differentiate.

Among existing IPS and IDS, Snort [60] is one of the most well known. It is an open Source network IDS, under GNU GPL license, with a very active community, and integrated in commercial devices by the Sourcefire company. Snort is able to perform real time analysis on the network traffic at all the levels of the network packet (up to the application and protocol layers), string matching as well as detection of various kinds of probes and attacks. Its popularity is due to its versatility but also because the community has the opportunity to write and enrich the security rules database. This allows to be more quickly and effectively updated on the recent threat. Today some rules sets available with Snort are considered so exhaustive that they are used as references for many commercial and research topics [37] [47].

As this work addresses Industrial Network Cybersecurity, we consider an ideal security target which fills all the threats and vulnerabilities we intend to cover. This very exhaustive target will be refined using the state of the art to highlight uncovered security flaws.

2.6 Threat Model and Security Target

First of all, to allow the correct definition of the security target and to list all the needs, it is necessary to describe the context of usage. We consider a security device allowing to interconnect two networks, with different levels of security, while ensuring their partitioning. Typically this matches the interconnection between the IT network and the industrial network, refer to Figure 2.2 in the industrial network architecture description. We also assume that it takes place in a network context which already has the main components of an industrial architecture, especially a SCADA and logs retrieval mechanisms (such as a SIEM for example). Concerning the format, we use the *First Level Security Certification* (CSPN) standard, promoted by the ANSSI, to construct the security target, it is a derivative of the *Ebios* method. This is an alternative to the Common Criterion, more rigorous than *Ebios* and with structures, methodologies and a process developed by the ANSSI. But we are taking the liberty of not using all its codes, in fact we are not writing the target of an existing device before commercialization but we list security needs in relation to the state of the art.

In the following of this section we will discuss the assets that we are targeting to protect, whether they are local, on the device, or on the whole network, threats that we intend to address and features used for protection. In a concern for formalism and to facilitate the recognition of what we seek to address, we will use the prefix *A_* followed by a number for the assets, *T_* for the threats and *F_* for security features. So the operating mode of the targeted device should ensure :

- A physical partitioning of networks, despite a logical interconnection.
- A network breakage.
- A protocol breakage.
- The filtering of industrial network protocols according to their standards.
- The semantic filtering of the application load in the protocol payload.

According to the operating modes that are targeted, sensitive assets needing protection are identified. These will generally take form of features on a security device and as we are speaking of network security we will consider that the device, this target addresses, must be configurable and administrable in terms of network and security policies. Table 2.6 summarizes the assets and gives an analysis of their purpose compared to the four fundamental security principles.

- **[A1.LOGGING]** The logging policy and function must stay operational.

Assets	**Confidentiality**	**Authentication**	**Integrity**	**Availability**
[A1.LOGGING]		X	X	X
[A2.USER_AUTHENTIFICATON]	X	X		
[A3.CONFIGURATIONS]		X	X	
[A4.DEVICE_INTEGRITY]			X	X
[A5.PROCESS_INTEGRITY]			X	X
[A6.APPLICATION_CHECK]		X	X	X
[A7.RULES]	X	X		
[A8.BOOT_CHAIN]		X	X	

Table 2.6: Sensitive assets synthesis

- **[A2.USER_AUTHENTIFICATION]** The authentication of the users is protected by a password, its integrity and confidentiality must be protected as well as the authentication mechanism.
- **[A3.CONFIGURATIONS]** The configuration must be protected in integrity and authentication.
- **[A4.DEVICE_INTEGRITY]** Integrity of network devices which constitute the control system (PLC, RTU, IED, SCADA, DCS), facing a protocol threat.
- **[A5.PROCESS_INTEGRITY]** The smooth progress of the industrial process according to the automated program and nominal functions.
- **[A6.APPLICATION_CHECK]** The process of checking the protocol's network flow according to the enforced security rules.
- **[A7.RULES]** The protection of enforced security rules.
- **[A8.BOOT_CHAIN]** The authentication and integrity of all running software elements.

In order to make sure that security functions researched in this work cover the needs and requirements of the security target, it is necessary to define the threats we are addressing along with their environment. The context of usage of the target as well as the hypotheses about its environment will allow to refine the security target and ignore some threats. In fact the following hypotheses remove some of the threats mentioned in the *Threats* section, including logs related and hardware related threats, as we are addressing network and application threats:

- Logs are regularly read, an administrator consults local events journals of the device, so we consider that any logged event is taken into consideration.

- The configuration by an administrator is considered trusted, the staff who configures the device knows what he is doing and can not introduce an error in the configuration.
- The device must be in a secured room with restricted and controlled access. By controlling the physical environment of the target, we ensure that no physical deteriorations will be performed as well as we exclude electronic circuit modifications, electronic chips replacement and many others side channel attacks.
- The device is well sized and suitable for the desired security needs, it ensures that the security functions are well adapted to the needs.
- All the staff who will handle and work with the target is trained and well informed of all the security prerequisites to the use of the device.

Then it is necessary to define the threats applicable to the target, they take into account an attacker without any specific rights on the network as well as an attacker which privileged accesses to any device of the network (login/password to a machine, shell on a device). Attack means are not taken into consideration, attackers may use their own material as well as a malicious usage of one of the network equipment. Attacks we are considering in this work are the following:

- **[T1.RIGHTS_POLICY_BYPASS]** An attacker manages to obtain privileged rights on the target. From this point he is able to perform various operations, from configuration modification to complete modification of the system.
- **[T2.IDENTIFIERS_THEFT]** The attacker manages to steal connection secrets (login/-password) of a user. This generally leads to T1.RIGHTS_POLICY_BYPASS.
- **[T3.AUTHENTICATION_BYPASS]** The attacker identifies himself on the target without connection secrets. This too leads to T1.RIGHTS_POLICY_BYPASS.
- **[T4.DENIAL_OF_SERVICE]** An attacker manages to perform a denial of service on one network to the other, through the device. The denial of service of the target itself is not considered.
- **[T5.CONFIGURATION_CORRUPTION]** The attacker manages to modify temporarily or permanently the configuration of the target.
- **[T6.FILTERING_BYPASS]** By managing to bypass the filtering process, an attacker can transmit illegitimate data through the device and induce dangerous behaviors on the network.

- **[T7.PARTITIONING_VIOLATION]** The attacker manages to bypass the network breakage and partitioning.
- **[T8.LOG_COMPROMISE]** The attacker manages to compromise, delete or add log in the system journal.
- **[T9.SYSTEM_COMPROMISE]** The attacker manages to alter or compromise a part or an element of the system. It will cause persistent attacks which will stay and occur even after reboot.
- **[T10.INDUSTRIAL_PROCESS_COMPROMISE]** By sending specific, but not malicious, controls to the industrial process, at critical moments or during critical operations, an attacker can badly damaged and compromised the industrial process.

The attacks previously listed refer to the *Threats* section, with more detailed version targeting network and application security. From there, it is important to list the security features that must respond to these threats. In order to focus on the thesis problematic, this work makes use of an existing platform from an industrial partner. It allows to inherit its architecture, security features and threat model, eliminating the need to address some of the vulnerabilities as it is already done.

2.6.1 Denelis

As part of this work, the threat model takes into account an existing network security device marketed by the company SECLAB [43], the DENELIS. It advocates for network isolation through network and protocol breakage. It takes place at the interconnection between two networks, generally the IT and the industrial one (see Figure 2.10). OSI layers (Figure 2.3) up to the fourth of incoming network messages are scrap and reconstructed on the other network (Figure 2.10). Network protocols, which take place on the upper layer of the OSI model (Figure 2.3), are inspected to check the concordance with their specifications. The threat model is focused around network and protocol threats, ensuring secure interconnection between networks with no concerns about the message content. Furthermore the physical security of the device is not addressed as it is allegedly placed in highly monitored environments with limited access control.

DENELIS architecture is based on 3 electronic layers, ingrown by three separate electronic circuits, each possessing its own FPGA. The first one is exposed to the first network, ensuring exchanges, the third is connected to the second network. The last circuit, called *Core* is located between the others two, it handles data transfer between them and is a guarantee of non-propagation of attacks. The two circuits connected to the networks (on

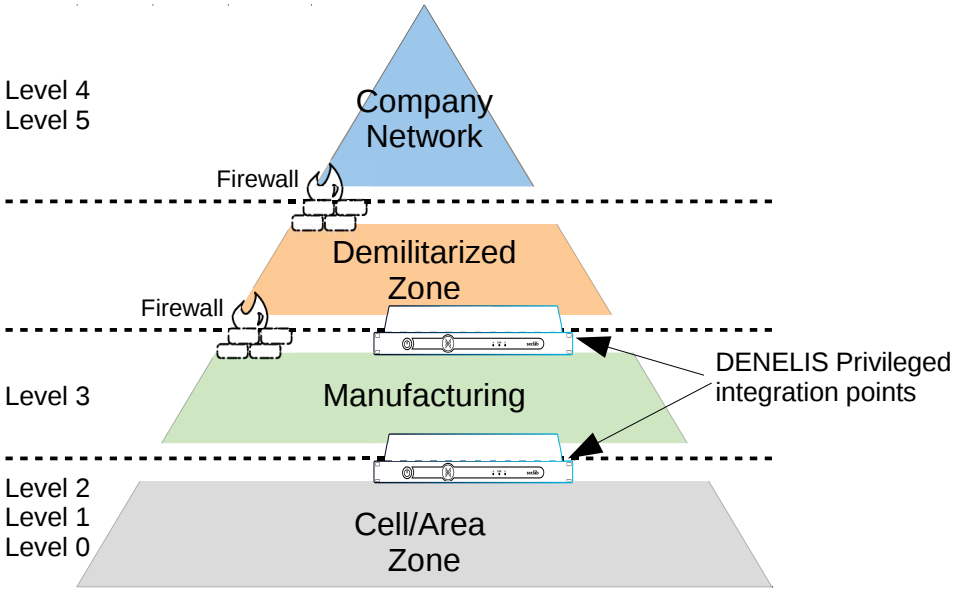


Figure 2.10: DENELIS integration in the network architecture

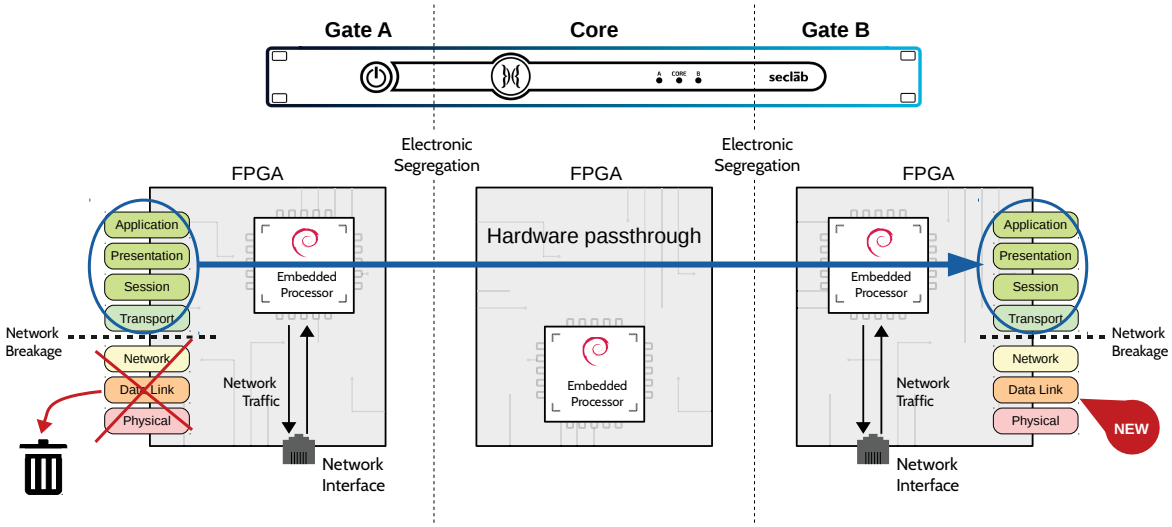


Figure 2.11: DENELIS high level Architecture

each side), called *Gates*, embed Linux running systems, and all three circuits are connected through high bandwidth bus. Figure 2.11 shows the high level architecture of a DENELIS.

The DENELIS is deployed in critical environments where the physical security and access control are assumed insured. It is installed in server rooms, whose security is handled by the client. It is therefore considered that the safety and physical integrity of the

device are not to be taken into account. It will not be degraded, broken or opened, which means that internal components (electronic chips, wires, welding) are not accessible. It avoids lot of attacks and vulnerabilities such as side channel, malicious replacement of electronic chips, direct probing on data bus and many others. It does not mean that all these vulnerabilities and attacks are ignored and will never append but it means that the security issues addressed by this device are not these ones. The DENELIS addresses network and protocol security issues and so its threat model is as follows:

- It addresses network and transport level threats, as OSI layers up to 4 are destroyed, attacks based on and requiring these layers are addressed.
- *Gates* are exposed to compromise, from simple denial of services to full control loss, but the spread to the *Core* and then to the other *Gate* is not possible. By making use of a proprietary protocol for inter-system communications and redundant hardware verification mechanisms, the compromise is restricted to the *Gate* and the attacker is not able to spread to the rest of the device.
- Incoming industrial protocols are checked on each system (*Gate* and *Core*). After destruction of the lower layers of the OSI model, the industrial protocol used in the layers five (Session), six (Presentation) and seven (Application) is analysed and checked to ensure that it matches its specifications. This analysis is redundant on each system in case of compromise.
- Any interactions on the device, other than through the network, that shows a risk is not covered by the threat model. It means that physical threats, side channels, serial access, are not considered in the model. The device may include features to cover these threats but not in the scope of the security target.

If we look at the list of attacks and threats, the DENELIS answers to some of them through the following security features and the Table 2.7 matches the security functions with the attacks.

- **[F1.CONFIGURATION_INTEGRITY]** An unauthorized user can not modify the configuration, furthermore the integrity is checked.
- **[F2.MANAGEMENT_OF_MALFORMED_INPUT]** The device is designed to handle correctly malicious or malformed network entries.
- **[F3.SECURE_STORAGE_OF_SECRETS]** Authentication secrets such as users and administrator passwords are securely stored, compromising the device does not allow to retrieve them.

	[F1.CONFIGURATION_INTEGRITY]	[F2.MANAGEMENT_OF_MALFORMED_INPUT]	[F3.SECURE_STORAGE_OF_SECRETS]	[F4.SECURE_CONNEXION]	[F5.SECURE_LOGGING]	[F6.NETWORK_BREAKAGE]
[T1.RIGHTS_POLICY_BYPASS]			X	X		
[T2.IDENTIFIERS_THEFT]			X	X		
[T3.AUTHENTICATION_BYPASS]			X	X		
[T4.DENIAL_OF_SERVICE]		X				X
[T5.CONFIGURATION_CORRUPTION]	X		X	X		
[T6.FILTERING_BYPASS]		X				X
[T7.PARTITIONING_VIOLATION]						X
[T8.LOG_COMPROMISE]				X	X	
[T9.SYSTEM_COMPROMISE]						
[T10.INDUSTRIAL_PROCESS_COMPROMISE]						

Table 2.7: Security features coverage over identified threats

- **[F4.SECURE_CONNEXION]** The connection to the administration interface is secured, ensuring integrity and confidentiality.
- **[F5.SECURE_LOGGING]** The device locally stores logs, securely sends them to a remote server and allows export by an administrator.
- **[F6.NETWORK_BREAKAGE]** Each network is protected against network attacks over integrity and availability from the other network. Compromise of a *GATE* can not be propagate to the rest of the device.

Looking at Table 2.7, it appears that threats **[T9.SYSTEM_COMPROMISE]** and **[T10.INDUSTRIAL_PROCESS_COMPROMISE]** are not covered by the actual security

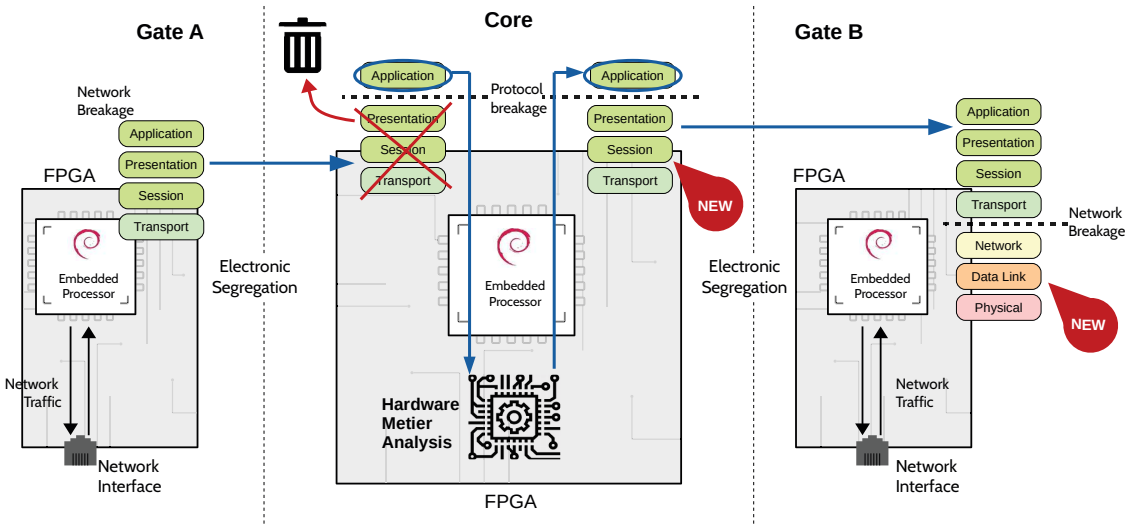


Figure 2.12: Research Platform high level Architecture

target of the DENELIS. This work aims to cover these threats and provides security solutions to overcome these critical flaws. To do so we setup a research and experimentation platform based on the DENELIS architecture.

Research Platform

As it is described in Figure 2.12, the need is to keep the three FPGAs architecture, but as the target is to perform extensive security operations on the *Core*'s FPGA, we need to release as much resources as possible of this FPGA. To do so the protocol checking performed is removed from the *Core* and established on the *Gates*, but the overall *Gates* functions are kept untouched. Furthermore to reduce parasitic processes on the *Core* and to work with the greatest number of industrial protocols, a stage of protocol breakage will be added on each *Gate* (this is discussed in the second contribution).

Protection against network issues is a key point of industrial networks cybersecurity, firewalls and IPS/IDS are the most represented of the solutions used but, while they generally resort to patterns matching over network data, they are insufficient when addressing specific issues. The security target puts in light these issues which concern application filtering and protection against persistent threats. The company Seclab, with the DENELIS, proposes an uncommon approach which arises as a valuable base platform, while it is still insufficient on the addressed application and persistent threats. This work targets to contribute on the state of the art while proposing innovative approaches using FPGA's platform.

2.7 Problematic

Industrial networks, in the same way as IT networks, are exposed to various threats, but this is the applications specificities and the probable consequences of a failure that make them critical targets. Among commercial and scientific solutions to address these networks cybersecurity, a recurring safety issue is that they are mainly software applications running on dedicated, or not, systems. But when considering the security of such applications, there are two sides to take into consideration. Firstly, the proper configuration of the security solution, as we explained when addressing the various threats that target a control system, an improper or incomplete configuration is a serious security threat. And secondly, these applications, as firewalls and IPS/IDS, run on systems that offer other features, such as management interfaces, web services. All these supplementary features are vulnerable to other forms of attacks, such as buffer overflow [63] for management interfaces and SQL injection [64] for web services, but it exists even more [65]. On a software platform, when an attacker manages to gain an access point, he can easily divert the operations and disable processes, thus invalidating the security application hosted. Consequently, these applications are considered intrinsically less secure than hardware only devices. It is therefore necessary to secure such applications in order to protect access and modification by an attacker.

Furthermore, even if specialized equipments (Industrial Firewalls, IDS/IPS) are able to handle network and protocol security, they rarely take into consideration the application aspect. Looking at a network message inside an Industrial Control System, it has three security topics, the network (transport), the protocol (communication) and the application. The first two topics were covered in the section concerning the threats, what remains is the application characteristic which is the meaning of the command and the impact on the operational process below. But to understand the application component, one needs to have the knowledge and understanding of this process. In fact, taking a legitimate operation, which is commonly requested by the control part, depending of the process state and behavior, it can cause disastrous consequences. If we take again the example of the electrical substation, with two power sources which supply a number of lines, each source can be disconnected by a circuit breaker. Wether it is the field operator or the control system, they know that both circuit breakers can not be activated or deactivated at the same time, it would either damage the power sources or cause power shortage. But a device which does not possess this application knowledge, does not know or understand the relation between equipments in the industrial process, and thus can not judge if an incoming operational order is damageable to the process. So in conclusion to completely acknowledge that a network packet does not present risks for the Industrial Control System, security devices need to understand the process and track its state, which

is rarely supported by classical and popular solutions.

From the observations made when looking at protection mechanisms and based on the security target, it appears that two security requirements are not currently addressed. **[T10.INDUSTRIAL_PROCESS_COMPROMISE]** concerns the application knowledge that is required to perform complete network analysis and **[T9.SYSTEM_COMPROMISE]** addresses system vulnerabilities toward persistent software attacks which aim to disable the security operations on the security device. These two security requirements match both topics discussed above and so they will be the two main problematics we intend to address in this work. To answer these needs we introduce two contributions, one centered around a secure boot chain for operating system using FPGA as root of trust. It addresses the problematic of existing software compromise, we intend to target the long-term compromise of the system and applications that will put the security analysis in danger. While the other concerns both hardware application filtering and software compromise, we propose to handle the security analysis through hardware resource and introduce an operational-process aware architecture.

2.7.1 Secure Boot

This contribution aims to answer the threat **[T9.SYSTEM_COMPROMISE]**, and to address the problematic of non-compromise of the security analysis. To ensure that the actual running system on the processor is secured, the research platform provides some features, such as **[F1.CONFIGURATION_INTEGRITY]**, **[F3.SECURE_STORAGE _OF _SECRETS]**, and it exists many solutions, such as the native firewall of the system or reduced access rights. But what about persistent threats, they typically tend to implement malicious elements which persist even in case of a system reboot. Doing so an attacker may install a backdoor on a device and durably compromise an application or system. To cover this threat, this work introduces a Secure boot chain which ensures that software elements running on the FPGA embedded processor are legitimate and not compromised. From the boot parts (spl,uboot) up to the actual running system (kernel, rootfs), each boot element is verified by the previous one. On the research platform the target is to raise the security of the *Gates*, but it is relevant to any System on Chip running Linux, as we tend to ensure the legitimacy of the system and its applications. We intend to promote the FPGA as a root of trust to create a chain of trust from one boot element to the other, without need for external components (secure elements) or modifications to the FPGA configuration logic.

2.7.2 Application Filtering

To answer the security threats **[T10.INDUSTRIAL_PROCESS_COMPROMISE]**, one needs to understand the incoming industrial process and to perform security analysis on the requested operation. One of the method privileged to perform quick and efficient analysis between a data flow and stored constraints is the patterns matching. It allows to match a stored operation to incoming data and so to authorize it. But [28] has demonstrated that simple pattern matching is not enough, advanced application threats need the security analysis to be aware of the operational-process state to ensure correct filtering. This work tends to introduce a time and operational-process aware patterns matching engine based on multiple dedicated Finite State Machines. Furthermore it introduces industrial protocols non-dependency through a common model for industrial protocols representation. But as we tend to propose an operational-process aware architecture using hardware resources, we address at the same time the problematic of security analysis compromise. Hardware architectures have a higher resistance to compromise than software so we raise the security of the security analysis. Where the first contribution tends to cover security needs for existing platform or systems through FPGA's usage, this second one proposes an alternative architecture. Furthermore as we rely on software mechanisms to perform industrial protocol non-dependency, we make use of the first contribution to ensure the non-compromise of these mechanisms.

Chapter 3

Secure Boot

Among solutions to secure industrial networks, most of them are pure software applications or eventually Systems on Chip (SoCs), using combination of a CPU and reconfigurable architecure. One of the most basic condition for those devices to ensure the efficiency of their security processes is the no-compromise and integrity of their software system. [8] has shown that security has become a concern for embedded devices, particularly since their proximity to the users, because they offer easy physical access. Multiple alternatives are available for a malicious entity to compromise a software application, through well known attacks to alter the functions, persistent attacks which remain even after reboot or directly by replacing some part of the system. This chapter discuss issues related to operating system boot security on FPGAs and particularly how to ensure authentication and no-compromise of the system through its boot elements.

3.1 Operating system boot on FPGA

The boot process of an FPGA-based embedded system does not differ that much of a classic architecture, although it introduces a crucial step when working on reconfigurable platform, the loading of the matrix. The bitstream is loaded by the configuration logic on the matrix, the method and the source depend on the configuration. It is either directly loaded from a memory, typically a flash support, or by the embedded processor (when there is one) and in this case the sources are multiple, from a memory, from the network or more generally from any devices or links available to the processor. This additional boot step only concerns the FPGA, the operating system which runs on the embedded processor has its own boot sequence which is commonly independent to the FPGA loading. The sequencing of these steps depends on the boot method choice, in one case the bitstream is loaded first and then the operating system boots and in the other case the operation system boots first and then loads the FPGA.

Concerning the boot chain of the operating system, it happens as following, while Figure 3.1 summarizes the whole boot process from the FPGA loading to the operating system.

- The Boot ROM is the very first software boot step, it is not accessible to the user as it is written in a dedicated Read Only Memory (ROM) by the FPGA provider. On power up this short software element handles very low level clocks setup (CPU clocks setup), reads the PIO configuration which indicates where to find the first preloader, loads it and jumps to its start.

- The first bootloader is the first boot element customizable by the user, it handles the basic configurations of physical resources (clocks setting, serial interfaces, network

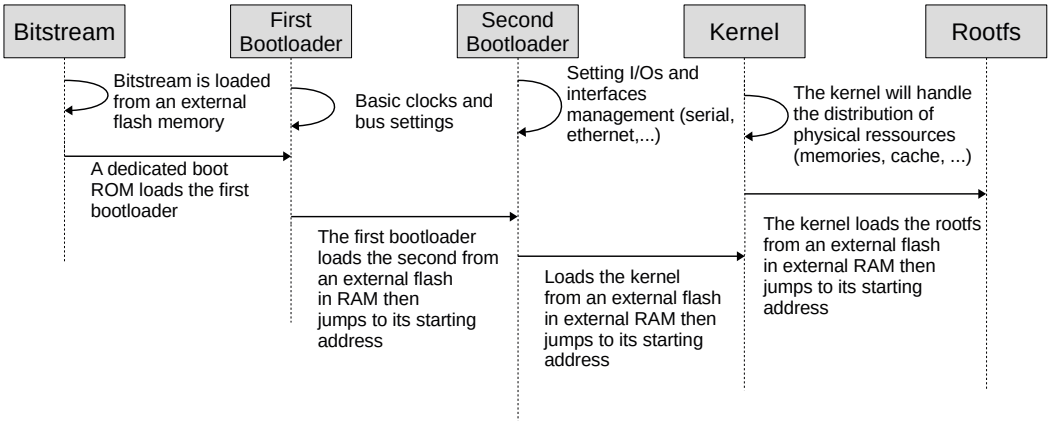


Figure 3.1: Boot chain of a FPGA based System on Chip

interfaces, ...). This stage takes place in two steps, a part of the bootloader is loaded in cache by the Boot ROM (L1 and L2), it configures the DDR RAM, then the rest of the bootloader is loaded in RAM and makes further setting. Its last task is to load the second bootloader in RAM and jump to it.

- The second bootloader is an optional step of the boot process, it handles similar configurations as the first bootloader but in more depth and where the first one focusses on the physical settings, this one focusses on the software configurations of the resources. Both bootloaders can be combined in one unique element which performs all the setup. Finally, it loads and starts the operating system.

- Operating system boot timing and operations depend on itself, it is the last step of the boot chain which leads to the system being ready to use. In case of Linux (which is the OS we will be using) the kernel allocates resources (memories, cache, ...) and loads the rootfs.

Most, if not all, of the elements of the boot process are loaded from external supports, such as flash memories.

Knowing the boot process of the System on chip, it is easy to understand that the compromise, voluntary or accidental, of one of the elements will impact the whole boot and will prevent the system from working correctly. In this context we construct a threat model to only consider the attacks which may lead to persistent threats on the software boot elements.

3.2 Threat Model

Attacks and threats that can target the boot elements are numerous, some take place when the system is running, while others during the boot process, but virtually, nearly all the attacks can potentially target the boot elements. As well, the flash storages on the device are a vulnerable locations: a malicious entity might replace or alter memories content through logical or physical access.
For example, if at runtime an attacker modifies the content of the Flash memory containing the kernel or rootfs, he is then able to install a backdoor that can persist after reboot. Even the RAM or the communications between parts of the system can be probed to retrieve or alter information.
Attack means are various, as the attacker needs to access with writable right, it can be achieved through software methods, side channels and many others. The same goes for the update of the bitstream or software elements, it can be done remotely through the network or locally with a physical access. In both cases, an attacker can either send malicious elements, or during a remote update, he can intercept and modify the data. All these attacks and threats can lead to a specific case that we intend to address in this work, the persistent threats.
Whatever the attack and its consequences, the most basic and efficient way to break or revert it, is the reboot of the system. Indeed it guarantees to put back the system in its original state, it does not patch vulnerabilities but it will stop all running processes and cut all connections. Persistent threats are, in the other hand, threats and attacks that will persist even after a reboot. Whether it is backdoors or system compromise, which make it starts in a state different from that expected, there is multiplicity of existing persistent threats. The most common vector is the modification of the boot elements.
It is the sequence of the boot steps that leads the system in its started state, so if one of these steps is altered, the state of the whole system is compromised too. The most common example is the replacement of one bootloader, as it loads the operating system, an attacker which replace or alter a bootloader can change the operating system that will start on the device and doing so completely change the behavior.

Our threat model, Figure 3.2, considers that the FPGA is secured. In accordance, bitstream encryption is used and then its freshness is checked using the manufacturer mechanisms, if available, otherwise the solutions described in [15] and [16] can be employed. Physical and side channel attacks are not considered in the scope of this work while the update of the bitstream is done by a certified entity and in a secured environment. The security of the external DDR RAM is not considered at runtime but approaches based on Merkel-tree [68] concept or through encryption and checksum [69] may be considered. We mainly address the security of the kernel and other boot elements stored in the

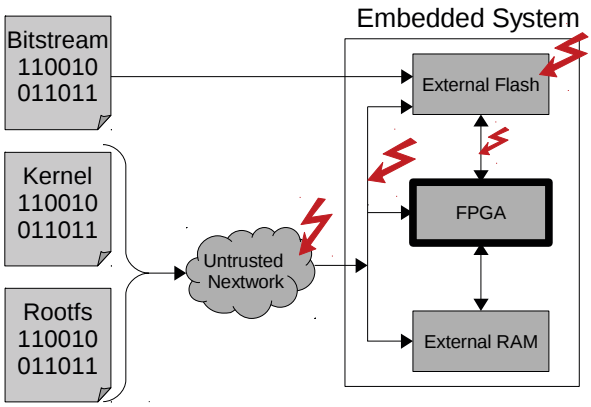


Figure 3.2: Threat Model

external flash memories.

3.3 Related Works

Literature addresses authentication, integrity and confidentiality of boot sequence elements, but most of the proposed solutions are expensive and complex: they generally require user resources, the modification of the configuration logic or supplementary devices (Crypto Processor, Trusted Platform Module) to operate. Using FPGA security mechanisms and configurable logic, in a cost effective way, it is possible to ensure a root of trust between consecutive stages of the boot sequence.

3.3.1 Operating System Security

The growing transistors integration enables FPGA to embed processor and run software applications. Their integrity and authentication are fundamental to ensure the security of the embedded device, and to prevent advanced persistent threats. By using Hash algorithms [18] and [19] propose to verify the integrity of the kernel loaded on the processor. Then, signing the hash and verifying it on the device allows to ensure the authentication of the kernel like described in [20], [21] and [16]. [23] uses a similar mechanism to check the kernel and performs additional memory measurements to ensure that the binary Image matches the expected size and memory location. To allow each boot stage to verify the next one, [22] offers a boot chain using a Trusted Platform Module (TPM) to ensure the authentication and integrity from the lowest elements to the kernel.

3.3.2 Bitstream Security

Compared to software only programmable systems, the reconfiguration capability offered by FPGAs present additional risks because it requires to also protect the bitstream. FPGA manufacturers (Altera [9], Microsemi [10], Xilinx [11]) offer possibilities such as encrypting the bitstream generated with their tools. A dedicated hardware engine handles the decryption of the firmware on the device. [12] proposed to add a CRC check for the bitstream to ensure its integrity and [10] achieves the same purpose with their AES-based MAC. If the bitstream check fails, [13] and [14] propose to fall back on a golden firmware stored in a dedicated read-only flash memory within the FPGA. To ensure the freshness of the bitstream and to prevent replay attacks, [14] offers the possibly to add a version number which is compared and incremented at each update. [15] and [16] suggest a similar approach by adding TAGs to the bitstream. With the SecReCon architecture, [17] submits a method based on a Root of Trust (RoT) and requiring a trusted authority, but the implementation is heavy. Finally to address the problematic of authentication [15] introduces a process of challenge response between the FPGA and the programmer.

3.4 SecBoot - Secure boot using embedded boot parts

3.4.1 Concept

Embedded systems require highly optimized designs and, to achieve the desired security level, this contribution aims at developing a new method that strikes the balance between both area and boot sequence integrity. We extend the approach in [16] using Open source tools to perform the kernel verification and take advantage of FPGA dynamic reconfiguration, as proposed in [70], to reduce the resources overhead. Rather than checking each element of the boot chain, all the vulnerable elements are embedded in the bitstream. In the presently proposed boot sequence, the first and second bootloader stages are stored in embedded BRAMs located in the user logic of the matrix. This allows to take advantage of security features made available by FPGA vendors for the bitstream, such as encryption and integrity checking. To reduce the overhead, caused by this method, over the memory resources, we take advantage of the reconfiguration capabilities of the FPGA and use two bitstreams. The first (Boot bitstream) with the BRAM memories for boot and the second (User bitstream) without these BRAMs.

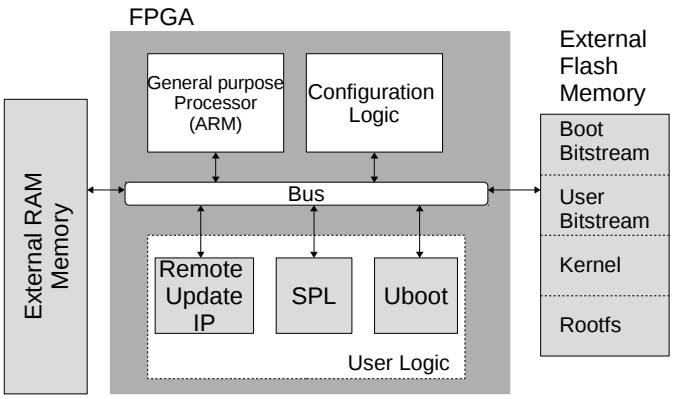


Figure 3.3: FPGA boot elements locations

3.4.2 Architecture

Embedded early boot stage

For FPGA's embedded CPU, booting on an embedded bootloader is generally natively supported. But the first bootloader still needs to be modified to get the correct memory mapping of the second bootloader. In fact, the first bootloader is not intended to locate the next one on the FPGA's matrix, pre-loaded addresses it knows to retrieve the second bootloader match standard interfaces such as network and serial. Uboot (Universal Bootloader) is an open source bootloader which natively supports kernel verification. It needs an **Image Tree Source (ITS)** file, which contains the information of the kernel image (path, compression, Image format, load address, ...) that will be verified, and the algorithms chosen for hash and sign. This file is compiled with a **Device Tree Compiler** and with the given set of private keys to generate a **Flattened Image Tree (FIT)** file. It contains the binary of all the given images and their signatures. Then Uboot is recompiled to embed the corresponding public keys. By default Uboot supports up to RSA 2048 for signature and SHA 256 for hashing, but to harden the signature and improve the security level, support for RSA 4096 and SHA 512 was added. The generated FIT file is stored in the external flash in place of the Kernel. The reader may refer to [71] for further precision about Uboot verification feature. Finally to set up a secure boot chain from FPGA to OS, the support for signature verification by the initramfs [72] embedded in the kernel is added. It ensures that the rootfs is authentic and not corrupted by verifying its GNU Private Guard (GPG) [73] signature. This verification is not in the scope of this work as the focus is on FPGA to kernel security.

Reconfiguration

Embedding early boot elements in the FPGA matrix consumes lots of RAM blocks. It reserves a part of the resources normally made available to the user and thus limits the possibility of implementation for the rest of the design. To avoid this overhead we take advantage of the reconfiguration capabilities of the FPGA. We use two bitstreams, the first (Boot bitstream) with the reserved RAM resources for boot and the second (User bitstream) without these RAMs. Figure 3.3 depicts the system overview with the location of each boot element. On FPGA's power up the first bitstream is loaded from the default address. During the boot process after kernel verification we reconfigure the FPGA matrix with the second bitstream. The reconfiguration is triggered by Uboot through a specific gpio. A dedicated IP accesses FPGA's configuration registers, rewrites the bitstream source address and then launches the reloading (on an Intel FPGA this IP is the Remote Update, on a Xilinx it is the Partial Reconfiguration Controller). The reconfiguration only impacts the user logic, the embedded processor will pursue system's boot during this time. The IP used for dynamic reconfiguration must be present in both bitstreams. In case of a shutdown request from the processor, we want to rewrite the default address in the FPGA configuration registers before shutdown, to be able to load the first bitstream on the next power-up. Figure 3.4 summarizes the new boot sequence with the actions of each element. Another way of freeing the RAM blocks used for the boot is to remap them, to make them re-usable by other design elements. This involves making these memories writable and raises a security question. Moreover the feasibility depends entirely on the design and the FPGA. For example, on a small FPGA, the size of the memory that embeds the Uboot imposes heavy constraints over the timing of the design and greatly complicates the addition of other IPs. On the contrary, on larger FPGAs, the constraints imposed by this memory are negligible and remapping is a feasible option.

3.4.3 Results

To demonstrate the occupied resources and time cost of the proposed boot chain we use an Intel Cyclone V 5CSEA5 FPGA with an EPCQ memory, for bitstream storage, and an EMMC memory, for kernel and rootfs storage. The first bootloader we are using is a simple preloader (SPL), based on Uboot, generated through Intel tools kit SoC EDS, and for the second bootloader, we use Uboot as mentioned previously. For the dynamic reconfiguration of the FPGA we use an Intel IP, the Remote Update (RU), coupled with a custom state machine.

The FPGA resources consumption of each element needed to implement the proposed boot chain is displayed in Table 3.1. At the end of the boot, when the OS is operating,

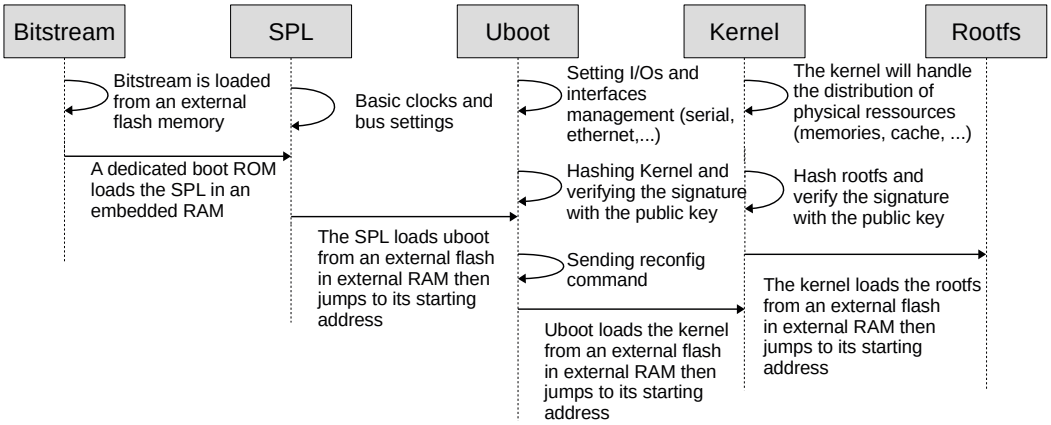


Figure 3.4: Secure Boot Chain with FPGA reconfiguration and kernel verification

		Block Memory Bits	M10Ks	Registers	LUTs
Before Reconfiguration	SPL	224,352	28	3	30
	Uboot	1,349,960	168	6	213
	Remote Update	0	0	74	125
	Total	1,574,312	196	83	368
After Reconfiguration	Remote Update	0	0	74	125

Table 3.1: Resource Usage Summarize

only the Remote Update IP and its state machine remain on the matrix. For the FPGA used in the experiment the resource cost represents a bit more than 0,3% of the available LUTs which may be considered negligible. In comparison with prior work which relies on the use of FPGA configuration logic in order to perform security checks over the boot chain, this work still has an overhead on the user logic. On the contrary, compared to asymmetric cryptography and hardware acceleration, the resource cost is lowered, we move the resources overhead from the FPGA to the flash, as we need to store 2 bitstreams instead of one. But nowadays flash memories are large and cheap, much less expensive than FPGA resources.

The boot time of the overall system from FPGA to rootfs, using a light kernel, takes several seconds. The verification of a RSA 4096 signature takes at most 1 or 2 milliseconds which is negligible in comparison. Furthermore the FPGA reconfiguration, in addition to being extremely fast, is done in discrete time while the kernel is booting so it will not impact the boot delay.

3.5 Discussion

Even if the experimentation was carried using an Intel FPGA, it remains applicable with any manufacturer supporting dynamic reconfiguration. For example on a Xilinx target: [74] and [75] show how to embed the early boot stages. Furthermore, by using the Partial Reconfiguration Controller IP it should be possible to achieve a similar work as this contribution. RAM blocks are basic IPs and are embedded in all FPGA families. The use of an Open source software allows free customization of boot elements. All these points allow a great flexibility in the choices of the system elements. This contribution is focused on reducing the resources reserved by security mechanism as describes in [16]. Using FPGA reconfiguration optimizes the area used by the boot elements to a barely negligible level. It does not require any additional device or processor to perform security mechanisms.

The Kernel integrity and authentication is checked by the Uboot. If an attacker manages to modify the flash content and alter the Kernel, the signature will differ and Uboot will stop the boot process. At this moment the system is locked and waits for a reboot or a physical intervention to update the kernel. We can imagine falling back on a golden Kernel stored in a Read Only memory to perform security checks while waiting for an intervention. In case of a replay-attack, an attacker could be able to retrieve an older version in order to downgrade the Kernel. Uboot verification will end successfully and the boot process will continue on the downgraded Kernel. In order to limit the risk, it is necessary to change the key used by Uboot to verify the signature and therefore to update the bitstream. This solution is recommended for security updates, each Kernel version is signed with a different key to prevent downgrade. All of this depends on the anti-replay protection of the bitstream. An attacker able to load a previous version of the bitstream with the Uboot verification key corresponding to the downgraded Kernel signature will successfully perform a replay-attack on the device. For example with the used Cyclone V FPGA and the previously described threat model we are vulnerable to replay-attacks. It is necessary to use other FPGA families or to implement solutions as suggested in [16] and [15]. Table 3.2 exposes the criteria discussed above in relation to related works.

Secure boot is fundamental to ensure the security of embedded devices. In this contribution we have presented a quick and easy way to secure a Kernel, by capitalizing on already existing FPGA security, using Uboot. Thanks to the reconfiguration capabilities we achieved nearly negligible resources and boot time overheads. The presented solution is compliant with other security features as the protection over early boot stage increase with the security of the bitstream. It allows building a stronger boot chain, with reduced development time over boot security. Instead of protecting the software system and applications, against the compromise through signature checking, the next chapter suggests to replace the software security analysis by a hardware IP. More precisely by a

	Kernel Integrity	Area optimized	Flexible	FPGA oriented	Anti-replay
Discretix 2006 [18]	Yes	Yes	Yes	No	Yes
Atmel 2006 [19]	Yes	Yes	No	No	No
ARM 2009 [21]	Yes	No	Yes	No	No
Apple 2009 [20]	Yes	Yes	Yes	No	No
Devic 2011 [16]	Yes	No	Yes	Yes	Yes
Kai et al. 2012 [23]	Yes	Yes	No	No	Yes
Microsemi 2014 [22]	Yes	Yes	No	Yes	No
SecBoot 2017	Yes	Yes	Yes	Yes	FPGA dependent

Table 3.2: Related Works Comparison

patterns matching architecture that will check the incoming network traffic and integrates application knowledge to perform operational-process awareness. By construction hardware applications are less vulnerable and more resistant to compromise (not immune), besides being faster. Furthermore, this work provides a proposition for protocol agnosticism which relies on a software process, so it will use this first contribution to ensure non-compromise on the current application.

Chapter 4

Versatile patterns matching for application analysis

Network cybersecurity solutions, like Intrusion Detection Systems (IDS), Intrusion Protection Systems (IPS) or firewalls, extensively resort to pattern or signature matching to identify threats among network data. However, it presents vulnerabilities inherent to the software implementation, and furthermore, industrial networks have specific constraints that are not always covered by solutions coming from the IT world (such as low-latency or support of specific industrial protocols). In parallel with the researches carried out on the protection of these software elements (see previous chapter on SecBoot for example), hardware solutions are more and more investigated [61]. To achieve similar security features (network filtering, protocol filtering) but on reconfigurable hardware (FPGA) a versatile solution is described. It uses a dedicated memory space to handle patterns storage, while FSMs are in charge of the comparisons.

4.1 Industrial Protocol Common Model and Protocol Breakage

This work targets a versatile solution to perform application analysis over network exchanges in industrial networks. To do so, incoming network messages will need to match sets of rules to ensure that there is no risk for the industrial process. But, as the research platform ensures network breakage and protocol checking in software, to analyse and understand the application load embedded in the industrial protocol using hardware resources, one need to understand the protocol. There are lots of different industrial protocols, to perform application analysis it would either require to understand every protocols and to adapt the process, or to abstract the protocol and to have a matching process which works the same for every protocols.

The unification and homogenization of industrial, IT and IOT networks under a common network protocol is not a recent topic, but as the interest around the IOT increases, it is a need that is becoming more and more present [49]. Two examples of this need are the Common Information Model (CIM) [66] and the Common industrial protocol (CIP) [67].

- The first (CIM) is an open standard for representation and modeling of networks. It allows to describe equipments of the network as objects with attributes, parameters and relationships in order to exchange management information. Currently maintained as a UML model and allowing to derive XML schema, the CIM was officially adapted by the International Electrotechnical Commission (IEC) as a standard for electric power industry.

- On the other hand, the CIP is an open industrial protocol, supported by ODVA and used for industrial automation applications. It was designed to provide a unified

communication architecture, is media-independent and is supported by many vendors.

- OPC-UA [42] is another protocol which targets the unification of communication standards inside control networks. Its platform independent architecture was designed to overcome standard OPC limits and to be particularly versatile toward information management and exchanges. It should be able to handle any kind of meta information allowing to match existing protocols as well as future ones.

But all these three are communication protocols which induce the requirement for every devices on the network to speak using the same protocol to permit understanding. Unfortunately none of these unification methods are popular or used enough yet, so resting on one of them would be very reducing, with respect for the diversity of used industrial protocols, and so reducing experiment possibilities. What we introduce instead is the use of a format of data organization instead of a complete language. Format and language are two different things while one make use of the other.
Format addresses data organization the same way as a structure in high level programming languages, it is a structured way to sort data with sizes and limitations. While a language is a way to express information, it commonly relies on a dictionary to perform translations with its own vocabulary and makes use of formats to organize data within.
As a format can remain simple or become very complicated when needed, the definition of a language requires defining its vocabulary, grammar and potentially a dictionary for each other language needing translation. In practice, languages are much more expensive to define but are very complete in terms of usage. In our case industrial communication protocols are assimilated to languages.
Looking at the space and resource problematic of this work we don't want to rely on an intermediate language, mainly because of the complexity and resource cost of the multiple dictionaries that it would require (one dictionary for each industrial protocol we want to support). In fact using an intermediate language means that the security process only needs to understand one protocol and greatly simplifies it. But it would also mean that for each incoming different industrial protocol we have a dictionary which allows translation to our intermediate protocol and so multiply the resource cost for each new incoming protocol. Instead we introduce the use of an intermediate format which is less expensive for the embedded process.

Most of the industrial communication protocols have in common the type of information they use and require to fulfill their needs. These types are as following:

- Routing and access control data such as unique ID or Equipment Name, supplementary to the classic IP address, Port and MAC address used to route data in an

Ethernet/IP network. These routing information, either strings or numerical values, are needed by the protocol to ensure that the communication takes place between the right interlocutors.

- Operation related parts as described in the introduction to industrial networks, industrial protocol messages request for an operation, with more or less abstraction. Information related to that operation, such as Function Code or Operation Number, are basic because they account for the reason why the message was send.

- Parameters, complementary to the operation, they are additional information that allow the correct application of the operation. Generally under the form of Address, Value, Object Identifier or anything which allows to specify the target and the context of the operation.

Figure 4.1 shows on three industrial protocols (Modbus, S7, BacNet), which parts are Routing, Operation or Parameters. Knowing the types of information, what we do is creating a structure with a section for each one of these types. Fundamentally this is the intermediate format we introduce, the target is in fact to reorder the incoming information carried by the protocol, while removing unused or irrelevant, to allow the security engine to process one type at a time and always in the same order. Doing so, instead of dictionaries, we store and use structures, each incoming protocol matching one structure which indicates the reordering needing to be done, knowing that this reordering is easily done using hardware resources. In addition all the supplementary data transported by the industrial protocol with no connection to the context are classified as miscellaneous as they are not needed for security purpose. It allows to introduce a custom format that we call common representation model constructed as following $\{[Routing]\ [Operation]\ [Parameters]\ [Miscellaneous]\}$, incoming messages data will be dispatched in the corresponding types in reference with the corresponding reordering structure.

The use of a common representation for most of the industrial protocols is one simple way to achieve protocol agnosticism, even if it is not perfect, it is a relatively inexpensive solution, in terms of resources and speed. Doing so allows to reduce the complexity over the matching process and keep it as simple and efficient as possible. But pattern matching alone allows only to compare incoming network messages to a list of registered ones, it does not take into account the context of the message, the state of the network and other parameters. As industrial control systems handle events and operations that can be part of a sequence or a chain, and need synchronization. In this case the context and previous messages are additional conditions to the pattern matching.

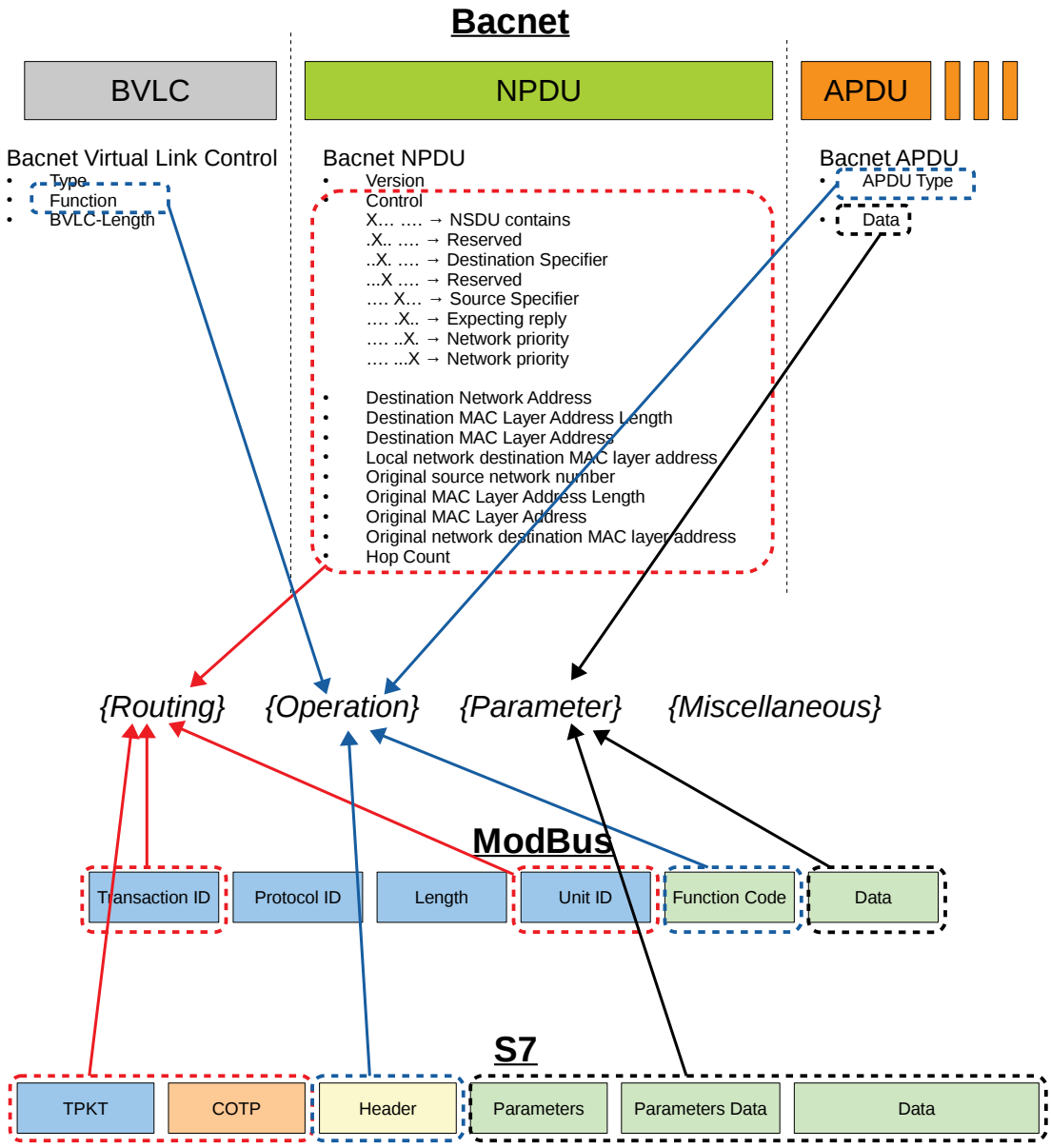


Figure 4.1: Information Sorting in Industrial Protocols

4.2 Operational-Process Aware Filtering

[28] has demonstrated that simple pattern matching is not enough, advanced application threats need the security analysis to be aware of the operational-process state to ensure correct filtering. In accordance with this work and in collaboration with industrials, it appears that to perform correct security analysis the pattern matching process needs to

be aware of the operational-process state but it needs time awareness too. It can be either measuring the precise flow of time over an interval or the awareness of the time and date over the day. Following are two examples of situations where time awareness is needed:

- Many devices have technical constraints that must be respected under penalty of failure. These constraints might be on the device itself such as operating temperature and supply voltage, as well as on other connected equipments that are driven by it. Inside industrial control systems this second condition is even more true because the operation of a device can impact all the others that operate below it. Taking the very common example of a circuit breaker, this kind of device has a breaking per second limit beyond which it may be damaged. And all the devices supplied by the power line, controlled by the circuit breaker, have a power breakage limit beyond which they are destroyed. As connected and remotely controlled circuit breakers are common, and the breakage command is a legitimate one which is needed in many cases, an attacker may exploit this command to voluntary overcome the maximum breakages per second limit and destroy some equipments. To avoid these kinds of situations, the security process must be able to monitor a time interval and to check the time between some specific commands.
- The second case is easier to demonstrate, if we consider a company with a production line working from 6am to 5pm, the start-up of one of the device of this line, outside of these operating schedules, may very well be an attack. For this reason it is necessary for the security process to be aware of the date and time.

To match these needs, the representation model previously presented is upgraded. Routing information are split between emitter and reveiver, operation data are divided between the operation itself (read, write, activate, update), parameters required by the operation (Address, Object Identifier) and data itself (values, files). The newly updated common representation model is : $\{[Source]\,[Destination]\,[Operation]\,[Parameters]\,[Data]\,[Time\ Condition]\,[Miscellaneous]\}$.

This model is used for the following work on the matching engine, as it allows protocol agnosticism. As long as both incoming data and stored patterns share the same format or protocol, the matching will work. For implementation purpose the common representation model boundaries are set as followed. Source, Destination and Operation are mandatory information, it represents the minimum data sent using an industrial protocol to formulate and order, remaining information (Parameter, Data, Time Condition, Miscellaneous) are optional. Technically speaking the proposed representation model does not hold boundary limitation for fields size, as it is mean to be a model and not a dictionary

neither a format. Each field matches the size of the corresponding data in the protocol message.
But implementation problematic requires to set up size boundaries, as it will impact resources consumption, device choice and feasibility. For the following work, we choose to set up the maximum size of a protocol message to 1024 bytes, meaning that we will work with patterns of this maximum size. IT is not randomly chosen, compact protocols such as ModBus and S7 have relatively small APDU maximum length, 256 bytes for ModBus, so 1024 bytes of maximum size might appear oversized. But as we target a versatile solution to handle more that simple protocols, more verbose ones need to be considered, and so taking i0ton account BacNet and OPC-UA. BacNet maximum APDU size is 1476 bytes and OPC-UA's one is more than 2 gigabytes, both overcome the maximum pattern size of 1024 bytes, but they are theoretical limits, in application protocol messages are smaller.
In conclusion, we have chosen this size of 1024 bytes because it offers a good compromise allowing to work on multiple kind of industrial protocols, and it is a good starting point to work on pattern matching problematic.

4.2.1 Network Dimensioning

PLCs and others remote controllable automated devices are made to be very versatile, they offer lots of features, many variables, I/Os, wide memory spaces. On the contrary industrial processes and architectures are made for maximum reliability. They are kept as simple as possible with no space for surplus or useless. Consequently the number of used functions/possibilities is very reduced in comparison of what the devices offer, a typical industrial network shows limited command sets for supervision and remote control. During the security analysis of these systems it is appropriate to consider only the used portions of the devices capacities and features, else the task would be huge and contains mostly unused functions. Taking into account access control, network routing and the industrial process, it is possible to reduce the number of interactions available and used inside the control system. A relatively complex industrial network uses a limited number of industrial commands (for remote control and supervision). From our experience on the field and based on research works carried out with industrials, it is estimated that if we list every single request and command of the control system, a hundred is enough to model the complete process.

4.3 Hardware Pattern matching

One of the purposes of this work is to increase the protection over the core process of the network security analysis, the pattern matching. The target is to ensure that this pattern

matching can not be bypassed or altered. And so one of the research trails is to perform pattern matching on the FPGA's matrix, only using the programmable logic, without resorting to the embedded processor. In fact hard wired pattern matching is harder to impact, as hardwired designs or IPs can not be stopped or bypassed as easily as cutting a process or a task on an operating system. It is also interesting to mention that hardware resources allow higher processing speed, which is notable when speaking of network related topics. For these reasons we target to rely only on the programmable logic for our pattern matching. This section addresses state of the art pattern matching using hardware resources, the benefits and constraints of the existing solutions with implementation demonstrations to select the most suitable starting point in our case.

The simplest form of pattern matching consists in a bit to bit comparison between incoming data and stored rules, with the constraint of reading the pattern in memory then processing to compare and starting again with the next pattern. It implies a significant delay problem when increasing the number of rules. But literature addresses pattern matching problematic through a variety of algorithms and methods [31], depending on the constraints imposed by the application.
From Boyer-moore [77], to locate a pattern in a data flow, to Bloom filters [29], which allow to compress patterns for space efficiency: each method provides benefits for a dedicated usage. But the industrial context imposes strong constraints, the first relying on the reliability of the security analysis. Security devices using pattern matching are responsible for scrutinizing the network traffic to ensure that the exchanges comply with the nominal industrial policy and are not likely to cause dangerous behaviors on the targeted equipments. The error margin of the matching engines must be as low as possible with a target at 0% of errors. Since this work is about hardware implementation of pattern matching engine, the targeted platform for experimentation is an FPGA (Intel Cyclone V).

Along the following subsections we propose a study and experimentation for three very popular pattern matching methods, Content Addressable Memories, Bloom filters and Finite State Automatons. These three are commonly used in network applications and offer space for research and optimizations as well as very different implementation methods.

4.3.1 Content Addressable Memory

A Content-Addressable Memory (CAM) is a special type of memory, which compares input data to a table of stored values, and returns the address of the match. The user supplies a data word and the CAM searches through its entire memory to see if this data word is stored anywhere. It is commonly used in very high speed searching applications

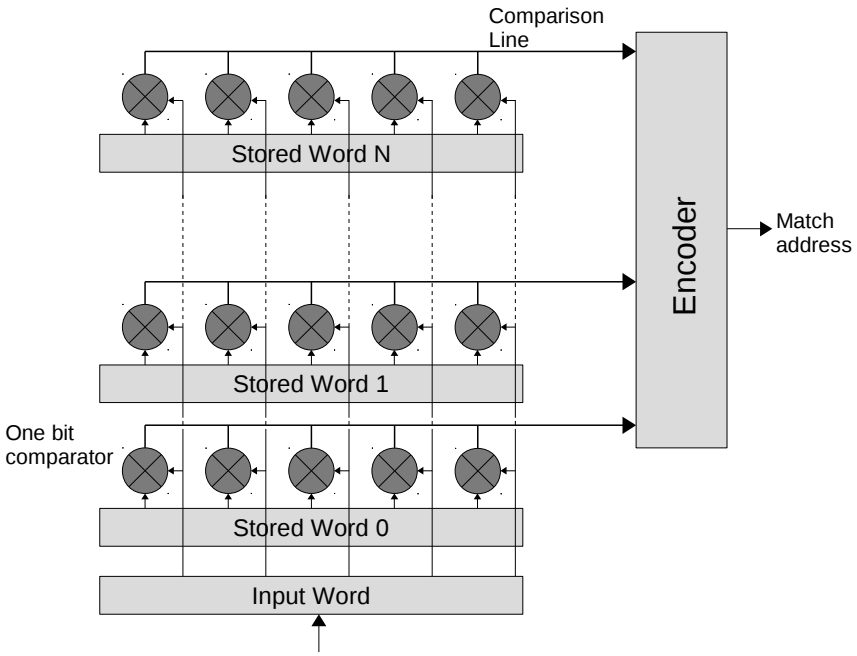


Figure 4.2: Typical Architecture of a Content Addressable Memory

due to the fact that it processes and returns the result of the comparisons in one clock cycle.
CAMs are privileged choices in our context because they cope with the availability and speed needed by industrial infrastructures and are well suited for hardware implementation. The drawback is the cost: each stored word needs its own associated comparator circuit to operate a bit to bit comparison with the input word (Figure 4.2). But it also means that the architecture is very reliable and leaves no room for uncertainty during the comparison. CAMs are proved very interesting and efficient when processing small patterns, experimentation (Figure 4.3) shows that increasing the size of the stored words compromises the scalability of the architecture. In a simple implementation, a word of size 1024 bytes requires 6953 Logic registers for storage.

Literature proposes to use RAM blocks, as this function is natively supported by all FPGAs, to ease the burden on the Logic resources.[76] suggested to use the address width of the RAM to store CAM words: the data corresponding to a RAM address indicates if the word is stored or not. The counterpart of this method is that it reserves the resources needed for storing every words fitting in the address width even if not used : 32 bits CAM words will need 32 bits RAM address width, which allows more than 4 millions of possibilities, even if we don't need this much. Furthermore this method still remains

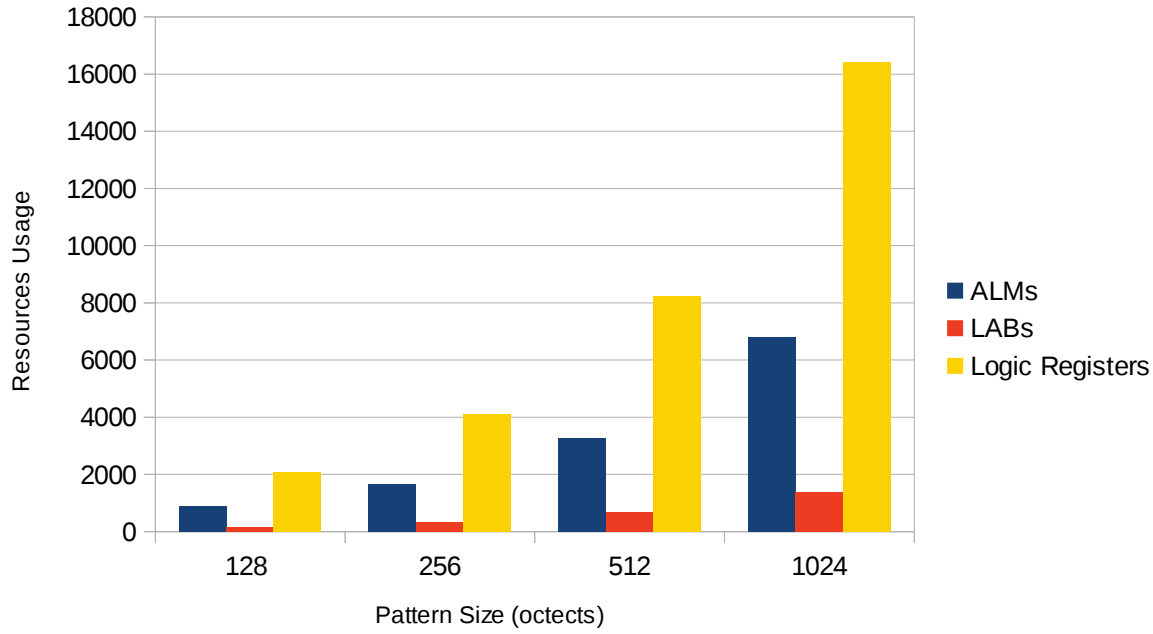


Figure 4.3: Logic Resources Usage for one pattern in accord to the CAM words size

not enough when handling large patterns, as the use of RAM blocks remains expensive (Figure 4.4). Considering, as previously, patterns of size 1024 bytes, it would requires 8192 bits of address. Divided between 16 bits memories, it represents a total of 512 memory blocks which is way more that the typical FPGA capacity.

CAMs have great compatibility with compression to reduce the resources overhead, as long as the compression algorithm used on the stored words is used as well on incoming data. The resources usage will change according with the compression ratio. On the other hand, using on the fly compression on incoming data might induce supplementary delays and the size of the output data will be variable depending on the size of the input. State of the art of compression methods ([25] MD5, [26] SHA-512) paired with FPGA quick processing allows to reduce this flaw (provided the likelihood of collision is low). Taking example of CAM words of size 1024 bytes, through an implementation of SHA-256, the newly hashed words of size 256 bits account for a compression ratio of 32, meaning that the needed resources for storing one word are divided by 32.

Regular expressions are commonly used for pattern matching as they allow to greatly reduce the patterns size and as well as patterns factorization. However due to its structure CAM doesn't support regular expression matching but it is still able to process ternary bits. It allows encoding the X value for one bit, each bit has 3 possible values : 0, 1, X. As shown in [27], it is generally achieved by adding a masking bit for each memory bit

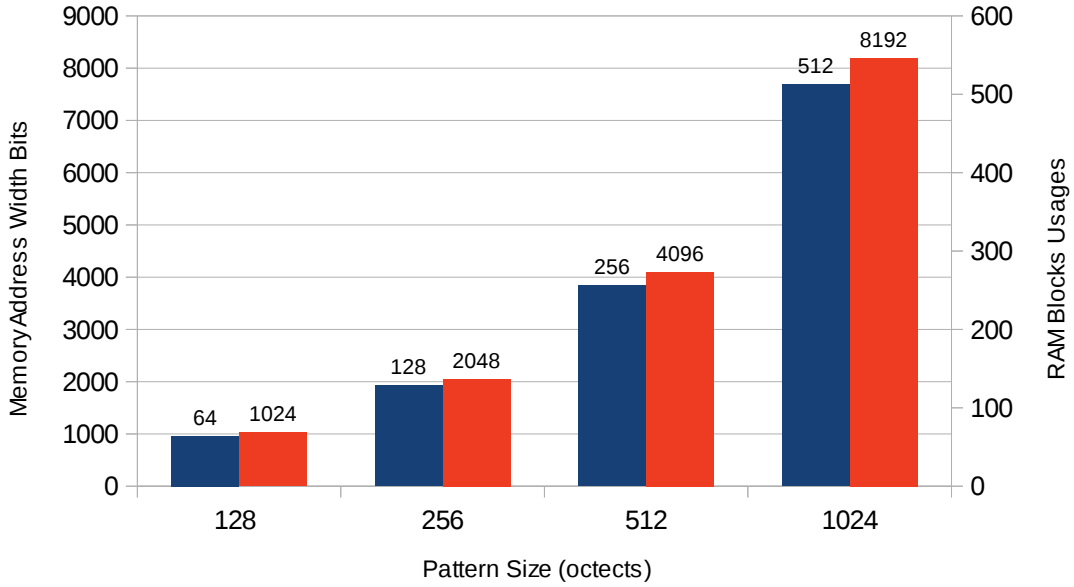


Figure 4.4: Memory Resources Usage according to the CAM words size

(care/don't care) or by encoding each memory bit on 2 bits or more (1 is 11, 0 is 00 and X is either 10 or 01). Sadly it doubles the memory space needed for words storage and is hardly compatible with compression.

CAM architecture offers high processing speed with low delays but is not very flexible and incompatible with common pattern matching optimization methods like patterns factorization or regular expressions. It is also interesting to note that commercial CAM chips exist, even if they are not so common, which may answer to a need for industrialization. Following this we will speak about Bloom filters, which are pattern matching structures relaying on hash functions to perform space efficient storage.

4.3.2 Bloom Filter

A Bloom filter is a space-efficient structure used to test whether an element is a member of a set. In our case it tests if the incoming data matches one of the authorized patterns. It is a bits array of size m, which uses k different hash functions. Each of them hashes and maps some set elements to one of the m array bits. To query for an element (test whether it is in the set), we feed it to each of the k hash functions to get k array positions. If any of the bits at these positions is 0, the element is definitely not in the set. A major issue of this structure is the possibility of false positive matches, in other words, a query returns that the data is in the set but in fact it is not. The filter is defined by two additional parameters: n is the number of elements in the set and p is the false positive rate. p and k^{opt} - the

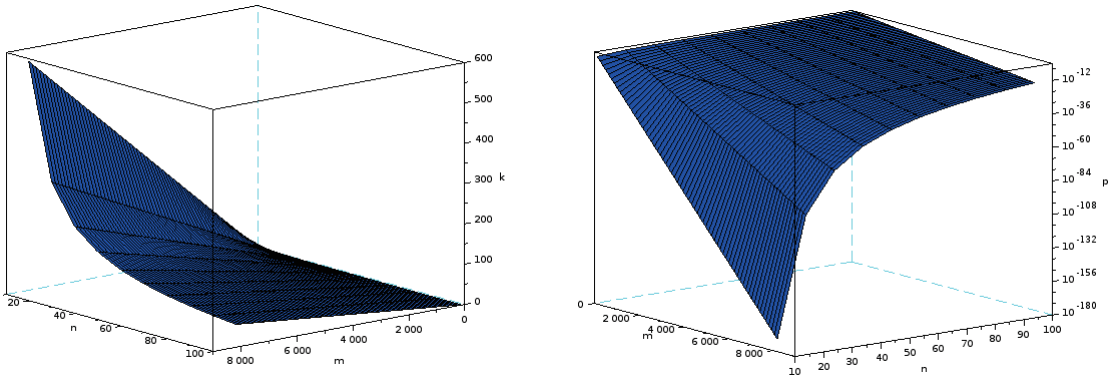


Figure 4.5: k and p function of m and n

optimal number of hash functions - can be expressed as functions of m and n.

$$p = e^{\frac{-m*\ln(2)^2}{n}} \tag{4.1}$$

$$k^{opt} = \frac{9m}{13n} \tag{4.2}$$

To solve the previous equations and determine the size of the bloom filter, two key constraints need to be considered: availability and security. Industrial network smooth operating and responsiveness is based on the fact that information is routed with minimal delay to its recipient, with transfer times of several milliseconds. The bloom filter has a fixed processing delay whose major part is due to the hashing of the incoming data, since the comparison is nearly instantaneous. This processing time can be minimized: [29] and [30] perform the hashing through combinatorial operations between the incoming data and random coefficients, allowing to hash the data, byte to byte, in one clock cycle.
The second point is the security: the objective is 0% error margin, which means that a match of the filter should be trusted and can't be a processing error: this is a parameter of the bloom filter known as false positive rate. Practically it is achievable if the bloom filter stores only one pattern and the hash functions have reliable collision rates, which is not interesting. To determine the size m of the bits array and the k number of hash functions needed to achieve a suitable false positive rate p, we choose to split the patterns into small sections up to 100 patterns, using multiple parallel filters. Figure 4.5 shows the evolution of k and p depending on the values of n and m.

Considering that a cryptographically acceptable false positive rate is $\frac{1}{2^{128}} = 2,94.10^{-39}$ (taking into account the state of the art of computing power available, the time and resources needed to find a collision on an algorithm with such false positive rate are consid-

ered not worthy of the cost) but it would require a bits vector of 18466 bits and 128 hash functions for 100 patterns. Those numbers are not realistic using standard hash functions implementations. [29] and [30] propose a solution to reduce the implementation cost of multiple hash functions: for each incoming byte, each hash operation matches to a combinatorial operation between the data and a random coefficient. The final hash vector is obtained by combining all the previous hashes. Doing so, the resources needed for implementing the hash functions are mainly memory blocks, required to store the random coefficients and the bloom vectors. These works are highly focused on the throughput of the system, so considering our problem, it is possible to reduce the estimated resource cost by reducing the lookup capability and performing sequential lookups. So according to these papers, for a false positive rate of $2,94.10^{-39}$ and with a bit vector of size $m = 8192$ bits, it would require 128 hash functions with 44 patterns per vector. If we use dual ports memories, it would need 64 M10Ks to perform the desired lookup in one clock cycle. For 100 patterns, it means approximately 2,5 mini blooms of 44 patterns so 160 M10K, and so forth, it increases linearly. For 1000 patterns it needs 10 times the cost of 100 patterns, so 23 mini blooms, and a cost of 1472 M10K cells. Otherwise it is possible to reduce the memory cost, if we consider doing the lookup in more than 1 clock cycle : for example a 4 clock cycles lookup needs 40 M10K cells instead of 160, for 100 patterns. The more patterns needed in the final bloom filter, the more it is possible to increase the lookup delay to keep the resource cost in acceptable values.

Bloom filters allow to trade false positive rate for storage efficiency which is an interesting compromise for network applications that required lots of patterns. But unfortunately this work requires a strict level of security where a pattern mismatch can be critical. Even through extensive resources consumption to reduce the false positive rate, bloom filters are hardly usable in our context. Now on we will look at Finite State Automatons, these mathematical models are commonly used in various applications and are privileged implementation choices for numerous pattern matching algorithms.

4.3.3 State Automaton

State Automatons are mathematical models of computation, widely used in software or hardware applications. This is composed of a finite number of states, connected by transitions, each one assigned with change conditions. The jump from one state to another through a transition is determined by a stimulus which triggers the associated condition. FSMs are common implementation vectors for lots of pattern matching algorithms [36], and there is variety of known algorithms [31] approached in the literature. Following are some of the most common and well used ones:

Brute force algorithm [32] could be considered as the most basic form of string match-

ing algorithm. It performs strict comparison between incoming and stored data, without need for pre-processing or supplementary operations. In case of a match or a mismatch, it shifts to the next position until it performs a complete match or reaches the end of the incoming data. It is used to search for the occurrence of a string inside a text but it may be used to perform strict matching between two strings of the same size, any mismatch would imply the end of the comparison. Using state automaton implementation, each character of the pattern is assigned a state and the matching of the current character allows to jump to the next state. This basic method is very resource-intensive and requires many optimizations to become effective.

Knuth-Morris (KMP) Algorithm [33] searches for occurrence of a chain P in a text S. It uses pre-processing to compute the position to restart the comparison when a mismatch occurs. It allows to avoid redundant comparisons, reducing the number of operations needed.

Boyer-Moore (BM) Algorithm [34] is the algorithm used by SNORT and one of the best for single pattern matching. Its particularity is that it starts the comparison with the last character of the string and then, in case of a match, continues from the first one. In case of a mismatch, it uses a table of comparison progression in addition to the last character's index to establish the next jumping position.

Aho-Corasick (AC) Algorithm [35] computes the dictionary of patterns before hand to construct a Finite-State-Automaton like tree, containing additional links, to allow transitions between stages sharing common prefix. Doing so, all the patterns are checked in the same time, this can lead to multiple matches in the case of patterns sharing inclusive relations. The efficiency of the algorithm depends of the pre-processing operation to construct the tree, meaning that any changes in the patterns dictionary requires to re-run this stage.

KMP, BM and AC target problematic is the occurrence of strings inside a stream of incoming data. It differs from the target of this work which is the perfect matching between patterns and incoming data. Consequently optimizations to reduce redundant comparisons or to determine shift positions before restarting the matching are useless and may even deserve the process. Brute force on contrary may appear to be missing optimizations but it offers more alternatives to add pre or post processing.

While speaking of state automatons and matching algorithms, it is worth mentioning Snort and Suricata, as they make extensive use of Boyer-Moore for pattern recognition, and hardware implementations based on finite state machine that are available in the literature [37]. But even if these two NIDS are very used in IT networks they lack the awareness of the system state and background.

Taking into account all these elements it appears that none of them match our needs.

Content addressable memories are very quick and efficient but lack scalability, while bloom filters, despite the scalability offered, pose a security problem due to the false positive rate. Most of the pattern matching algorithms presented are designed for string matching inside data flow, while we target a formal matching between two data of the same size. For these reasons we introduce a **custom pattern matching solution based on brute force matching using Finite State Automatons with operational-process and time awareness**.

4.3.4 Synthesis

It exists numerous solutions and methods to perform pattern matching, some rely on specific and dedicated architectures, while others come under the form of algorithms, without particular implementations. Hardware pattern matching is a popular topic, even more in the network context, as it allows high processing speed, as we can find implementation such as [37] in the literature. But the context and needs of the industrial environment are on the other hand badly addressed, and mainly through extended firewall approaches. The specificity of our contribution is that we intend to combine hardware pattern matching with a very uncommon concern: time and operational-process state awareness. This would lead to complex application analysis, commonly addressed through dedicated firewalls, which are software applications. We target to propose an innovative approach to pattern matching for industrial network security which resorts to brute for matching with several post processings.

4.4 Versatile Operational-Process Aware pattern matching

[28] shows that to protect an industrial network against advanced threats, the actual state, previous and future behaviors of the system need to be taken into consideration as well as time awareness. Stuxnet [1] is a good example, modifying the rotation speed of an equipment is a valid operation, allowed on the targeted device. But taking into account the actual rotation speed, it appears that brutal changes may deteriorate the device. Even if it was performed through local modifications of the equipment firmware, the same result would have occurred by reproducing the attack through network communications. Based on this work we introduce an operational-process aware solution which answers to our problematic.

4.4.1 Concept

The purpose of this work is to propose and explore an Operational-Process aware pattern matching engine with enough versatility to fit the specific needs of most industrial networks. Keeping in mind the security, it needs a strong pattern matching algorithm as a base, but customizable enough to add post-processing and perimeter functions. Based on the previous state of the art, we sticked to brute force matching, using finite state automaton implementation to ensure the core matching function. It allows perfect trust with no error margin but lacks optimizations in processing speed or efficiency.

Core matching engine

State automaton's implementation of Brute force matching using one state per character is very expensive. Taking as example a string of 1024 characters, with 8 bits characters, which is on purpose a very long string but remains in the specifications of most recent industrial protocols.
Two kinds of storage are available to construct the state machine, using logical resources or memories (RAM blocks). Figure 4.1 shows the cost in resources, for both implementation methods, for a 1024 bytes patterns. Logical resources are a privileged choice when processing small patterns, but when their size increases the cost became unbearable, 11% of the logic blocks for only one pattern is too much. When using RAMs, the states and transition conditions are stored in memory and the automaton simply reads it. But the problem is that even if the memory cells cost is under the capacity of one RAM block (10k bits), it still reserves a complete block, and as those blocks are hard wired in the FPGA architecture, they come in limited number. Even while storing multiple small patterns in one RAM block, if it goes beyond the lookup capacity of the device it induces delay problematic, which quickly becomes a problem in network applications. So the smaller the pattern is the more memory space it wastes. The test platform uses Cyclone V FPGA which came with 10K bits RAM blocks, so for a 1024 bytes pattern the wasted memory space in a block is relatively small (20% loss).

Using one automaton per pattern causes to multiply the resource cost according to the number of patterns, and is a serious threat to scalability. Merging some state machines to re-use shared states involves wider automatons with potentially high number of transitions and so the efficiency will worsen greatly as the patterns similarities decrease. When using RAM blocks for patterns storage, it is possible to perform space optimization by re-using wasted memory space and store multiple patterns.
Space optimization is made through two parameters, firstly the maximum lookup capacity of the RAM block, which is 2 in our current FPGA, allowing at maximum dual ports

Implementation	Resource	Usage	Cyclone V Resources Usage
Logic	LABs (Logic blocks)	152	11%
	Logic Registers	1403	5%
	M10K Memory (10K Bytes)	0	0%
RAM	LABs (Logic blocks)	1	>1%
	Logic Registers	32	1%
	M10K Memory (10K Bytes)	1	>1%

Table 4.1: Resource cost of FSM implementations for one 1024 bytes pattern

memories. Secondly the size of the patterns, taking as maximum size 1024 bytes, which is wide enough even for recent industrial protocols (we are aware that in some case industrial protocol messages may exceed this maximum but this is not common). The minimum size of patterns is set taking into account of relatively old protocols, 256 bytes is enough to handle even small patterns with minimal memory loss.
So, to optimize the memory consumption, each RAM block is virtually divided in eight smaller memory spaces. These may be used individually to store small patterns as well as combined to handle bigger patterns, allowing to store from 2 to 8 patterns in a RAM block (depending of the size).

When storing two patterns of 1024 bytes the memory allows to read both at the same time. To address the problematic of concurrence when handling more than 2 patterns, the memory words are cut in 4 and the patterns are stored vertically as seen in Figure 4.6. Storing 32 bits or 40 bits words has the same cost (one RAM block in both cases), so the 8 supplementary bits are reserved for future uses and the words are severed as 4 elements of 8 bits data and 2 bits reserved. The first 10 bits of a memory word belong to the first pattern, the next 10 bits belong to the second pattern and so on. For patterns between 257 and 512 bytes, it uses 2 small virtual memory spaces of 256 bytes and so on, for patterns superior to 512 bytes which use 3 virtual memory spaces. See Figure 4.6 for patterns representation in a RAM block, depending of their sizes.

Furthermore, as seen in [27], ternary bits are a good way to optimize patterns number by allowing factorization. So the same method is used, 4 of the previously reserved bits in the memory word are used as ternary bits which indicated that the associated 8 bits of data are processed as "don't care" (8 bits of data, 1 ternary bit, 1 reserved, and so on).

Still, with the perspective of reducing the patterns number to optimize the resources consumption, the concept of array/range is introduced in the pattern definition to allow factorization. The concept is very simple and widely used in all kind of topics, multiple elements, in our case patterns, which share lots in common are brought together, allowing to store only one time the shared part and keeping the differences. In our case we will

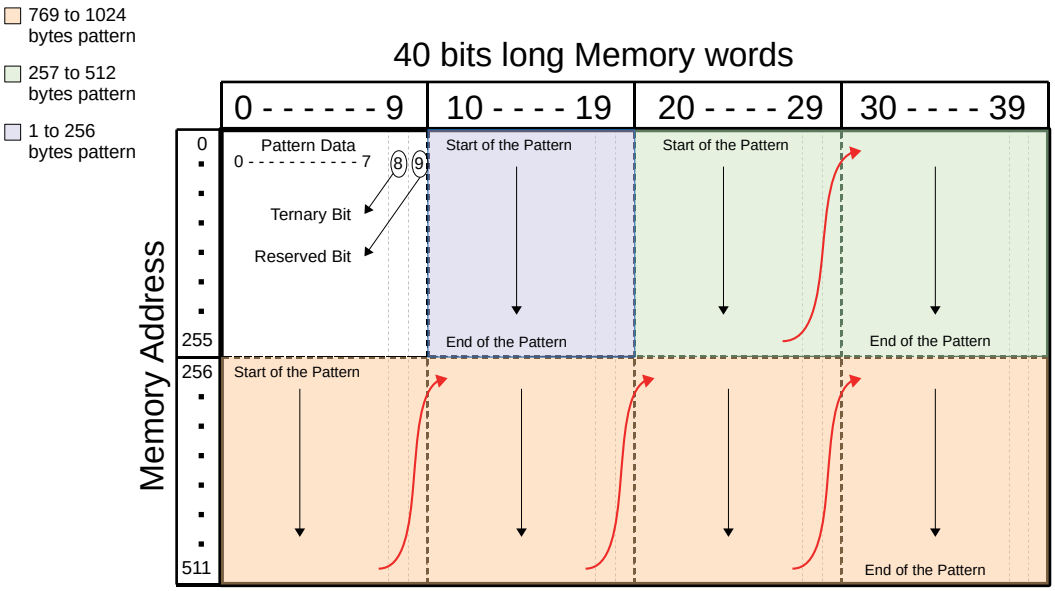


Figure 4.6: Patterns storage and representation in RAM

target patterns which present only one difference, and on the same part, furthermore we need these differences to be consecutive values.

For example, two similar patterns whose only difference is the IP address of the recipient, one is 192.168.20.34 and the other 192.168.20.35. We will factorize the two patterns in one with the recipient IP address expressed as a range from 192.168.20.34 to 192.168.20.35. It only works if the two values are consecutive, but three patterns can be factorized in the same way if the third is 192.168.20.36. This method of factorization can appear pretty reductive as it only addresses consecutive values, but in fact they are very common when speaking of industrial control systems. Whether in the network addressing plan, made for availability and efficiency it does not need to jump some addresses, or directly inside the PLCs programs, I/Os and memory cells are mostly successive. Technically speaking, specifying a range, with lower and upper bounds, in place of a single value allows a pattern to match multiple input values. Typically in industrial process, the parts which would make best use of ranges are values contained in parameter or data fields. So we are using the last reserved bits in the memory word to indicates the presence of a ranged value in the pattern. In some cases it may allow to divide by two the space needed for patterns storage in return for a little increase in the processing complexity of the engine.

As we have spoke about pattern matching, which is the core of this work, we will now discuss about operational and time awareness, achieved through additional post process-

ing modules.

Post Processing for operational and time awarness

Patterns are expressed using the previously presented common model for industrial protocols. Each part/element/field of the common model is assigned a state, taking into account the fact that parameters and data are optional. The main part of the engine is the core matching process using virtually split RAMs and state automaton, in addition, a post processing process will handle operational aware checking and time awareness.

While [28] requests the process state regularly, or when needed, to perform process aware filtering, this method induces supplementary data exchanges with all devices which are part of the control system. These supplementary communications can impact the availability of the devices, if not of the whole network. A less invasive solution consists in recording some prior-identified messages and determinate the system state in accordance. This requires, on the other hand, to have a knowledge matrix of flows, application values and operating ranges on which to base the analysis. Even if this study is necessary before any security analysis, to determine the system state in accordance requires it to be well detailed.
More precisely the whole system state is not taken into consideration, only the state of some devices of the network. The first step is to identify when a control device, such as PLC, RTU and IED, receives a order/command whose execution depends of the actual state of any other device in the network, as well as the device itself. We will call the command in question the "*Order*", while the state it depends is called "*Condition state*".

For example, taking a heater or burner, when it receives the command to start heating up, it should not operate if their is nothing to be heated. These kind of relations, when they occur between elements controlled by the same control device (PLC, RTU, IED), are not always checked locally inside the automated program, and even less when it appends between elements linked to separate control devices. Then it is necessary to highlight the list of events and commands which lead to the "*Condition state*", they are called "*Condition order*", knowing that both "*Condition order*" and "*Order*" or in fact patterns. A list of dependencies is constructed for each "*Condition state*", it takes into account the conditions of entry but also exit from the state. These dependencies might be only one command which leads to the "*Condition state*" as well as a multiplicity, and the same for the exit. They are converted into expressions which are function of "*Condition order*" using "*OR*" and "*AND* operators. Finally, knowing the "*Order*" and functions of "*Condition order*" we linked them together using a TAG. It is basically a unique ID which associated a "*Condition order*", or a function of them, to an "*Order*", the value of the TAG indicates if the "*Condition state*" is

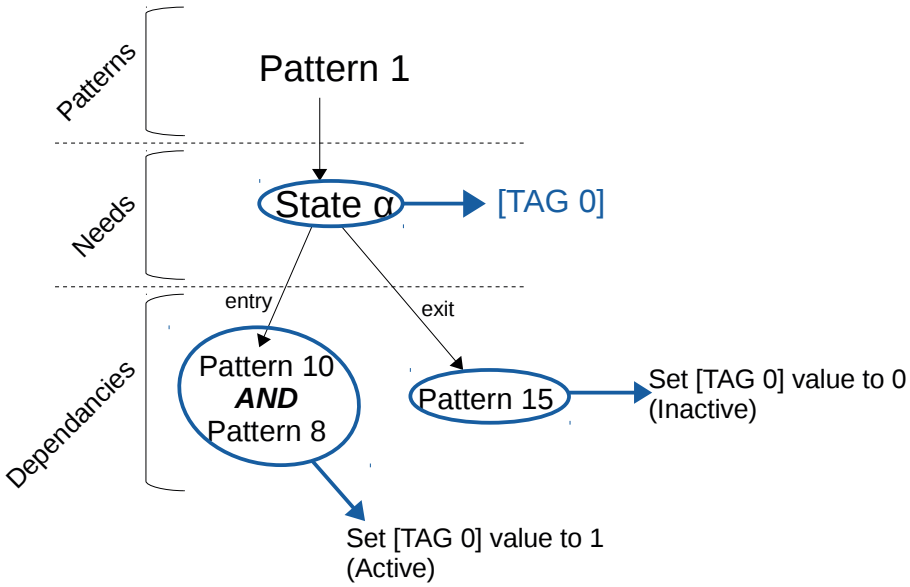


Figure 4.7: TAGs definition and usage

active or not. "*Condition orders*" are processed by a dedicated post process, while a centralized process stores and update the TAGs in a database. The patterns which depend of one or multiple TAGs for positive matching will cause interrogation of the database to conclude the matching.

For example, relying on Figure 4.7, taking the pattern number 1 ("*Order*") which matching ends positive only if the process is in the specific state α ("*Condition order*"). We determine that to enter in the wanted state α, the commands corresponding to patterns 10 and 8 ("*Condition order*") must have been sent to the process and the command corresponding to pattern 15 ("*Condition order*") makes it exit the state. From this point the state α is mapped to a new TAG, the number 0 for example, and two functions are made, one with *pattern10 AND pattern8* and one with only *pattern15*. While the TAG and its value are stored in a database, the functions, with their impacts on the TAG 0, are handled by an associated process. Finally, the pattern 1 is completed with the TAG it depends, as well as its value.

In most cases an "*Order*" depends on one or two "*Condition states*" which themselves only depend of a low number of "*Condition orders*". The previously described method works well in these instances, but it can occur time where numerous "*Condition state*" and "*Condition order*" are involved in the matching of a single "*Condition state*". To manage these situations two solutions are feasible, using factorization methods to perform backcrossing on "*Condition orders*" and "*Condition states*" to reduce their number, or to deliberately ignore some, to only consider the most critical "*Condition states*". Both methods will not

be discussed furthermore as they are more implementation problematic, but they are not needed in our experimentation context.

The last post processing operation concerns the time awareness, either through time measurement or time location. Patterns, which require it, are associated with a timestamp condition, containing either a short time interval with a maximum number of occurrences Structure 4.3, or a time window Structure 4.4 (in hours or even days).

- In case of time intervals, when defining the security requirements, a maximum number of occurrences of the corresponding command in a given amount of time is set. When we match the command an occurrence counter is increased by one, this counter is decreased by one each Δt interval, where Δt is *Time Interval* divided by the *Occurrences Limit*. If the occurrences counter goes beyond the occurrences limit value then the match is invalidated as the time awareness condition is broken, else it confirms the match.
- In case of time window condition, the security requirement is defined with a time window in which the corresponding operation is allowed, outside it is considered as a security violation. This time window is characterized by an upper and a lower bound, it can be counted in hours or days as well as just a couple of seconds or minutes.

When the corresponding command is matched, the time awareness process will check the timestamp, associated with the network message, and compare it with the upper and lower bound of the time condition to validate or invalidate the match.

$$\begin{aligned} &IntervalStructure\{ \\ &\quad Occurrences: \\ &\quad OccurrenceLimit: \\ &\quad TimeInterval: \\ &\} \end{aligned} \tag{4.3}$$

$$\begin{aligned} &WindowStructure\{ \\ &\quad UpperTimebound: \\ &\quad LowerTimebound: \\ &\} \end{aligned} \tag{4.4}$$

Distributed Security Analysis

We are resorting to a passive solution to know the current state of the process, as we extrapolate it from observing network exchanges. Doing so we need to be sure that we catch any network communications that may impact the operational-process aware analysis. The most effective way would be to have as many network security devices as network industrial devices, and so being aware of all the messages from and to any of these equipments. In practice it is redundant and unfeasible, as the cost and efforts to deploy all the security devices and to modify the network architecture would be enormous. A good compromise, that we propose, is to identify the most critical industrial devices of the network, either in term of security or process state, and to assign them a dedicated security device.
Then to virtually divide the network in smaller sub-networks, connected by nodes, and place security devices on these nodes. In Figure 4.8 we take as example a very simplified representation of the process layer of an industrial control system to show a proposition of distributed deployment for security devices. Multiple security devices as close as possible to targeted network equipments allow better control over network communications than only inspecting network interconnections.
It allows to follow inter-equipment communications inside local sub-networks, and so easily follow the process state evolutions, as well as it splits the attack surface, reducing the compromise risks. Furthermore in a context where the resource cost for the implementation on FPGA is important, dividing the analysis between multiple devices distributes the cost and may allow to alleviate this constraint.

Synthesis

To summarize we introduced multiple functions, the main is hardware pattern matching using parallel state automatons in ordered memories, then we spoke of patterns factorization through consecutive values. Finally we introduced post processing functions for time and operational-process awareness. But as we rely on Finite State Automatons, we have chosen to use a method inspired by RAM based FSMs to select the correct Post processing needed as well as the correct state transition. The first byte of a pattern is a code that defines what type of pattern it is, currently we process 3 types, simple, operation-aware and time aware. This code will impact the transition between states as well as the corresponding post process which will be called. In a way, it is similar to RAM based FSM state's that found their next state in the current one. This method is quite flexible as it allows to add or combine post processing stages simply by assigning new codes (pattern types). In practice the FSM starts by reading the type code and sets up the post processing transition

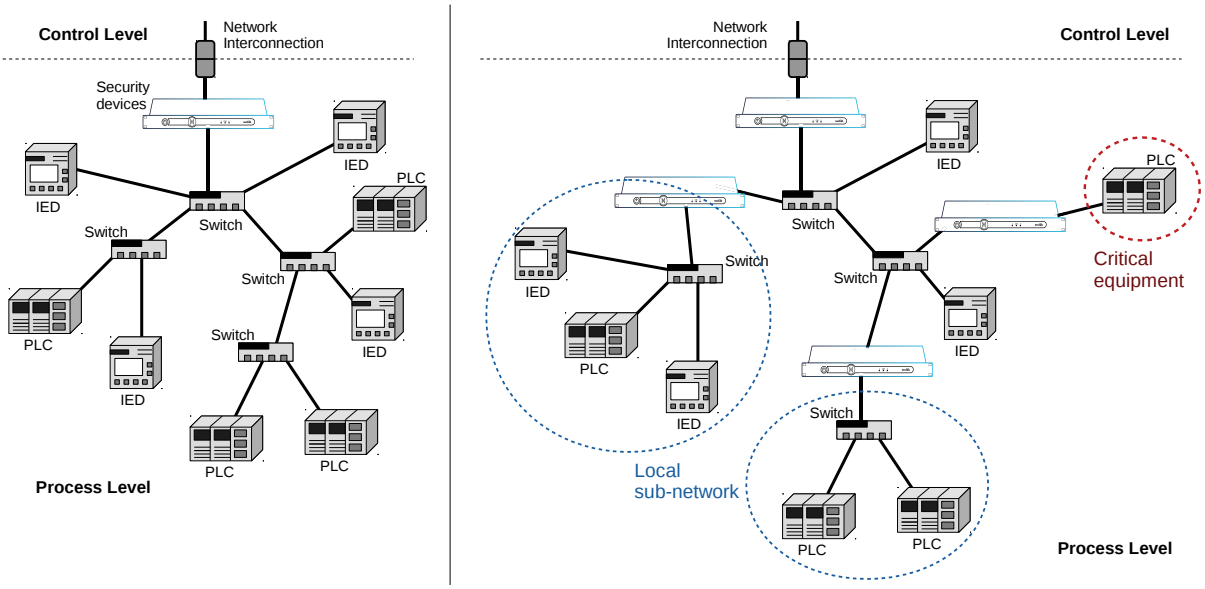


Figure 4.8: Network Architecture with distributed security devices

state if needed, see Figure 4.9 for graphical representation.

4.4.2 Architecture

Figure 4.10 shows how the contribution is integrated into the research platform previously described, there is a first stage of format translation where the application payload of the network message is translated into our custom format, and then multiple matching modules work in parallel to perform the application analysis. The whole architecture of the application analysis engine, composed of multiple matching modules, is described in Figure 4.11 and is made as following.

A pattern matching FSM, as described previously, is in charge of the comparison between incoming messages and patterns stored inside the dedicated memory. But as we target to increase the bandwidth of the system in terms of pattern comparisons, we rely on one of the fundamental techniques when working on network application, the parallelization, so we duplicate the matching FSM. As described in the Concept section, we optimize the sorting of the patterns in memory, so for dual port memory (which is standard on FPGA) we are able to output eight words of one byte at the time, for a total of eight pattern matching FSMs, each one processing one byte of its own pattern. In case of longer patterns, which overlap on multiple sub-memory spaces (see Figure 4.6), one of the matching FSM is simply deactivated and not used. Then we have one time awareness post processor for each FSM, they only process the time constraints. Finally Operational-

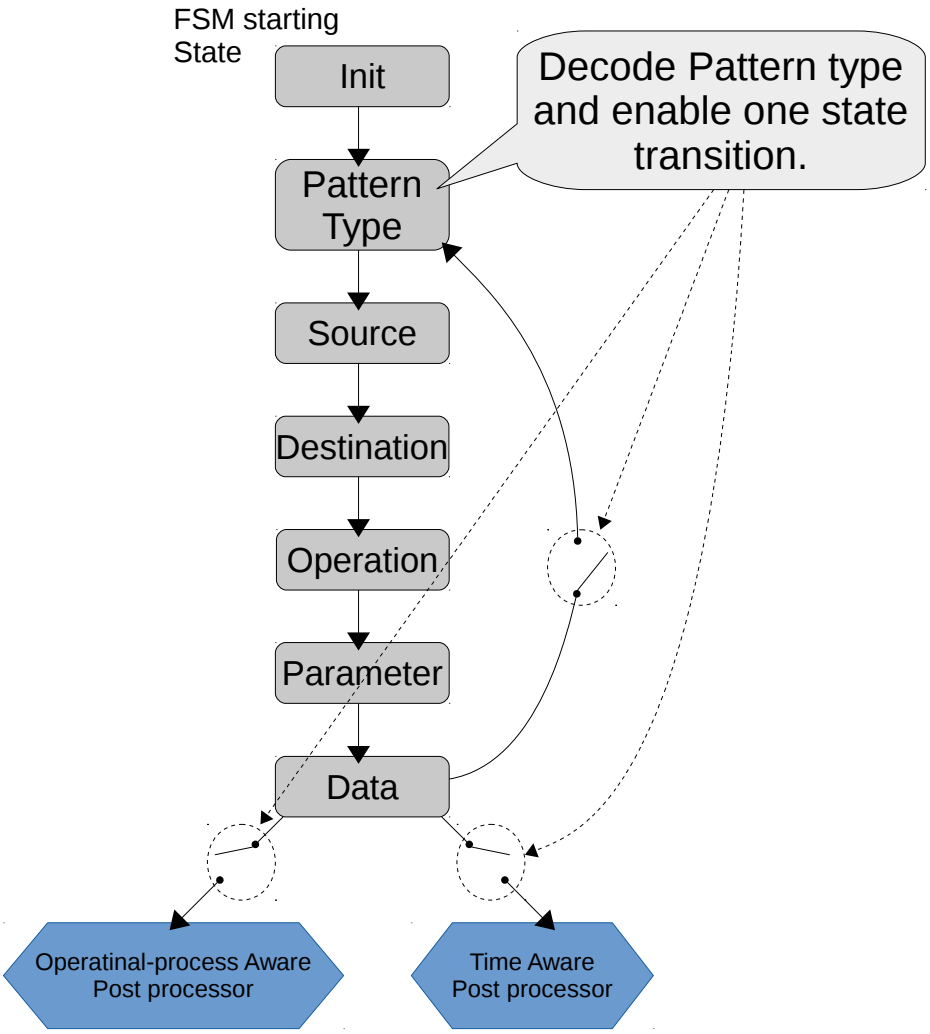


Figure 4.9: Post processor set-up in accord with pattern type

Process Awareness is handled by the TAG manager, it is in charge of the TAGs update and storage as well as answering the FSMs when requested. The communication between matching modules and the TAG manager is performed through a custom MDIO interface, matching FSM are divided in two groups of four (one group for each port of the pattern memory), both group share the same driver to the MDIO bus and each FSM accesses it through a dedicated buffer. We will now go into more details concerning each element of the architecture.

The core module of this architecture is the pattern matching module (Figure 4.12), which is duplicated eight times, and is architected around the pattern matching FSMs. Firstly a pre-processor multiplexer decodes the pattern type to set up the post-processor,

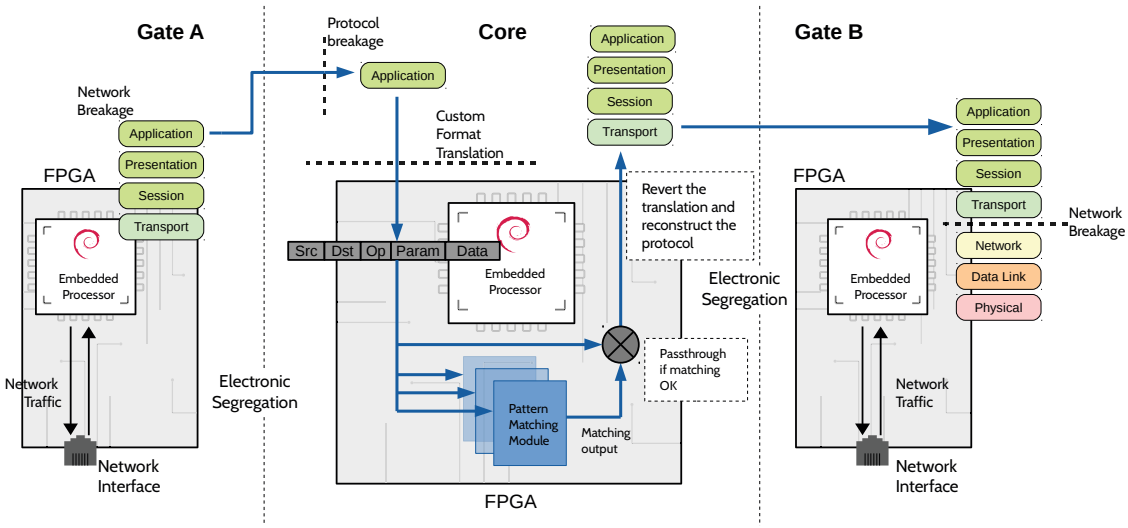
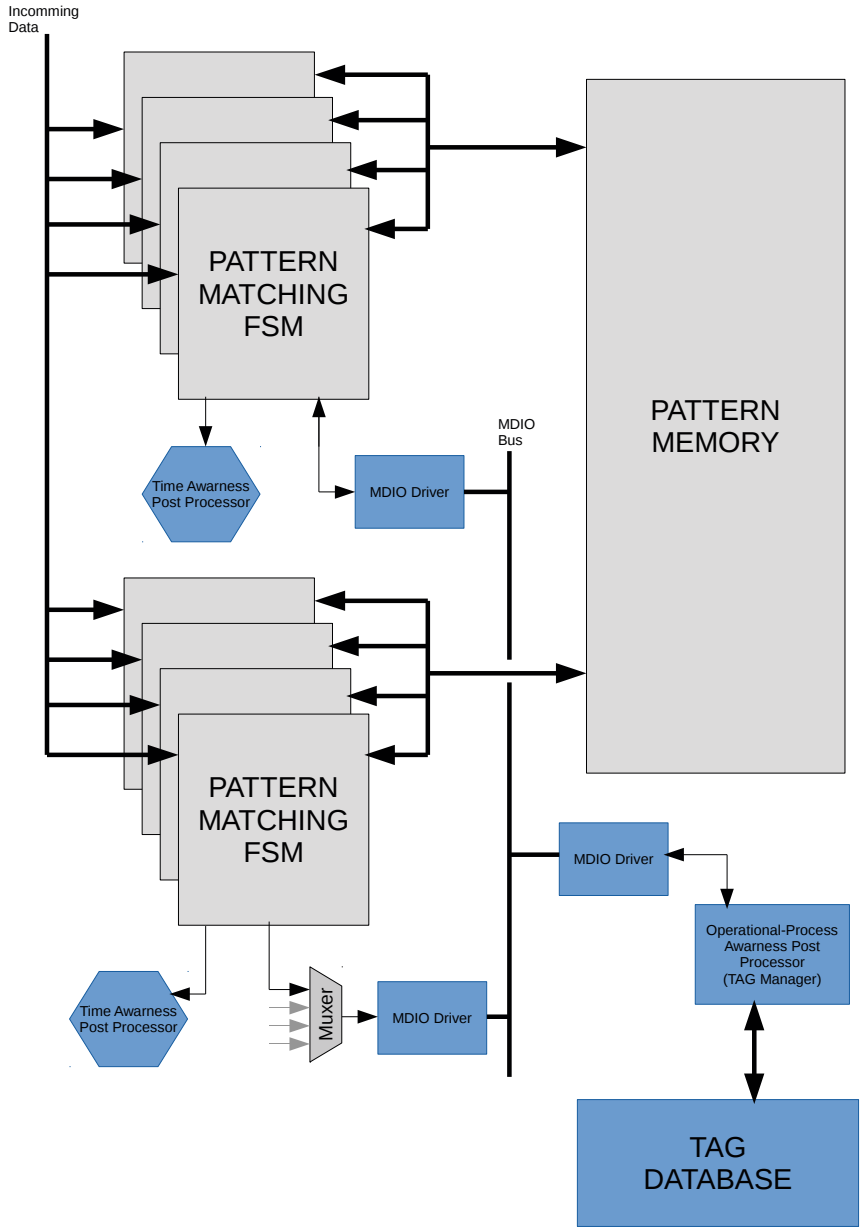


Figure 4.10: Overview of the research platform with the application analysis

as a reminder this type is the first byte of the pattern. As the pattern memory has 40 bits words, its outputs are divided into four sub-words of ten bits, see Figure 4.6. Then the FSMs process the pattern matching, byte by byte, between the incoming data and the stored words. At the end of the comparison each FSM calls the pre-selected post processor (if needed, in accordance with the pattern type).

The first post processor is the time aware process, it processes two types of constraints (see Structure 4.3 and 4.4) which are time interval and time window. In case of time window constraint, the bounds of the window are embedded in the pattern, so the process only needs to compare them with the current time and date.
In our case we rely on a I2C interface with a Real Time Clock, outside of the FPGA fabric, to provide the correct time measurement, but it is possible to use any other method. When processing time interval constraint, there too the Occurrences limit and Time interval are embedded in the pattern but the occurrence counter increment and decrement must be managed by the post processor. When processing the pattern for the first time, it reads the constraints and stores the Occurrences limit and Δt time to decrement the counter (which is the Time interval divided by the Occurrences limit). Then it creates the Occurrences counter and manages it, the resource needed to store these values is small as there is at maximum two of them (Δt time and Occurrences limit) by pattern.

The time aware post-processor is shared between all the eight FSMs as its function is simple, it allows to conserve resources. The second post-processor is dedicated to operational-



![h]

Figure 4.11: Operational-Process Aware Pattern Matching System

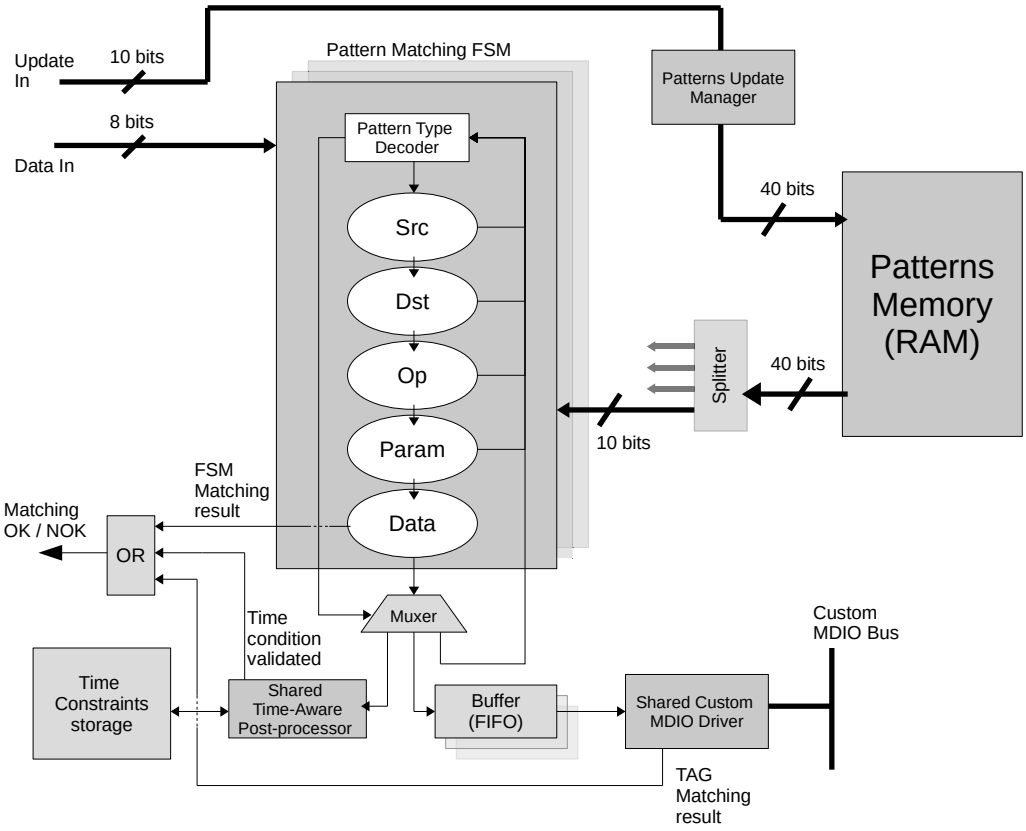


Figure 4.12: Architecture of a Operational-Process Aware Pattern Matching Module

process awareness, it is a two parts module, each FSM accesses to a shared custom MDIO interface, through a dedicated buffer. These buffers are necessary to allow multiple accesses to the same shared resource. We speak about a custom MDIO driver as we are not using the MDIO standard but a custom format heavily inspired by it, both MDIO packet and our custom payload are represented in Figure 4.13.

The second part of the operational-state aware process is made of the TAGs handling module. Its job is, when requested by a pattern matching module, to update or return the value of one or multiple TAGs. Each pattern has a unique identification number which is made of the unique ID of the pattern matching module itself and the number of the pattern (from 1 to 8). These IDs of used by the operational-aware process, their combination is associated to a TAG and a dictionary is constructed according, it is done offline, in the same time as the patterns definition. In a more explicit way, taking the pattern number 12-1, which is the pattern one of the pattern matching module twelve, conditional to state α (see upper for details about conditions and states). The industrial process enters this state when the pattern number 10-3 is matched and exits the state with the pattern 5-2.

	MDIO Packet														
Bit :	0	1	2	3	4	…	8	9	…	13	14	15	16	...	31
0	32 Bits Preamble														
32	Start		Operation		Physical Address			Register Address			Turn-around		Data		

Standart MDIO Packet Format

	Custom Packet														
Bit :	0	1	2	3	4	…	8	9	…	13	14	15	16	...	31
0	Start		Operation		Pattern Matching FSM Address			Pattern Number			Unused		TAG value		

Custom MDIO Packet Format

Figure 4.13: Comparison between the standard MDIO packet and the custom payload used

- A first entry is added to the blank dictionary with the TAG number 0, it associates the pattern 10-3 with the TAG 0, the operation is *update* and the value 1 (for state active).
- Then a second entry is added with the pattern 5-2, the TAG 0, the operation is *update* and the value 0 (state inactive).
- Finally a third entry is added with the pattern 12-1, the TAG 0 and the operation *return*.

When a matching FSM processes and matches the pattern 12-1, it knows from the pattern type number that it is supposed to ask for a TAG value and should compare this value with the one in the pattern to end the matching. The same goes for pattern 10-3 and 5-2, with the difference that the pattern type is different and the FSM does not expect answer, it simply notifies the match to the operational-aware process and concludes. When TAG's value change is associated to a function of patterns, the corresponding function is stored in a dictionary. TAGs value are stored in a dedicated database, and as they

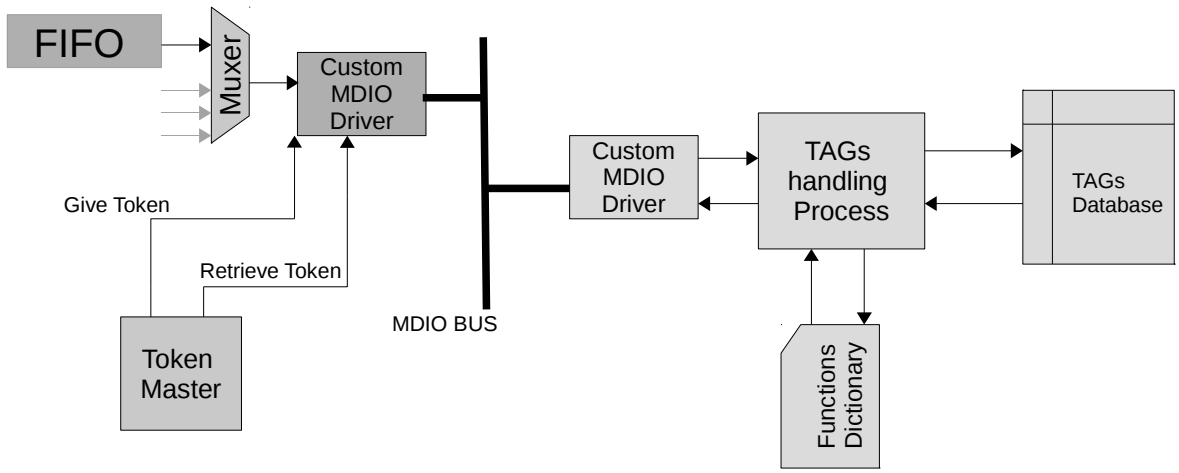


Figure 4.14: Architecture of the TAGs handling Process

are small words with a value of at maximum eight bits, if not binary, we are relying on a RAM-CAM approach to organize and search through it. It allows very quick search and update to not delay further the answer and the matching. This operational-aware process communicates through the custom MDIO interface with all the matching FSMs.
Each matching FSM has a dedicated FIFO then a multiplexer interfaces them to the custom MDIO bus, one at a time, bus access arbitration is handled by a master which resorts to token distribution. It gives access to the bus to one FSM through a token, then either the token is gave back when the exchange is complete or the master forcefully retrieves it. It is a relatively simple way of operating which ensures a sequential access to the resource. This architecture is represented in Figure 4.14. Being shared can bring up congestion problematic and delays, even using CAM, it is feasible to split or duplicate the database depending on the needs, as it reserves relatively low resources.

One of the topics, that was not addressed until yet, is about updating the patterns, and so on updating the security policy. In fact one of the key points that ensures network security is to be up to date. In a context where network security is the next battlefield, new threats and attacks are found every days, to keep abreast of them and their countermeasures is a must do. The efficiency of a security device relies on the fact that it is up to date in the state of the art of the network security. Knowing that it would be unthinkable to propose and design a not updatable pattern matching engine. As it is described in Figure 4.15, the whole pattern matching module has a high level supervisor which is able to put in standby the matching process and allows an update of the patterns, this operation is done between two matching of course. The pattern memory is writable only through this supervisor. Even if all the logic needed to correctly load new patterns into the mem-

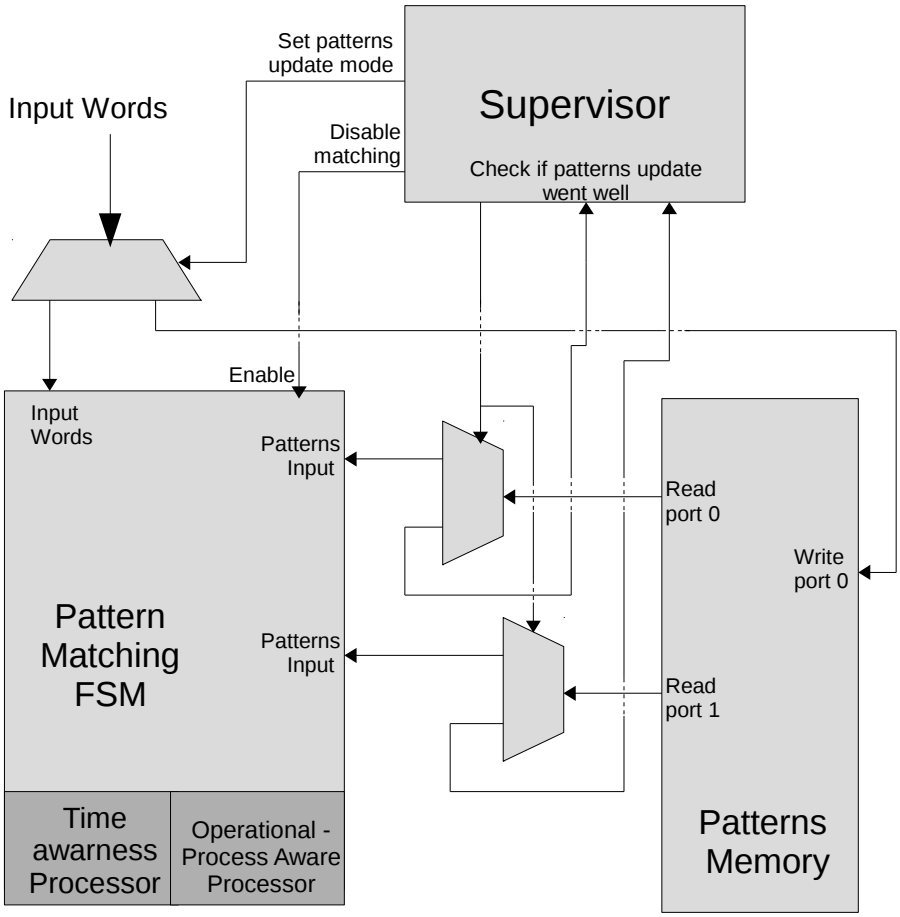


Figure 4.15: Patterns Update mechanism

ory and set up the matching FSM in accordance, is embedded into the supervisor, the patterns update is currently not studied in detail nor experimented, as it faces a complex and unaddressed problem in this work: the security of the updates. This problematic is vast, as it affects the encryption, authenticity and non-regression of security rules, it would require a mechanism to ensure secure exchanges between the FPGA and a third part. Whether through the network, mass storages or files, it addresses needs like shared secret, encryption mechanisms, keys storage and many others.

4.4.3 Experimentation

For experimentation purpose we use a Cyclone V Intel FPGA, which is a middle sized industrial one with 15,880 ALMs, 60,376 Logic Registers and 2,7M memory bits. The implementation of the proposed pattern matching architecture will be done in order to measure some chosen parameters. The resources consumption and area cost of each parts, as well

Entity	Sub-module	ALMs	Logic Registers	Memory Bits
Pattern Matching Module	Pattern Memory	0 0%	0 0%	20480 0,75%
	Finite States Automatons	1793 11,3%	1284 2,1%	0 0%
	Custom MDIO Driver	132 0,8%	347 0,6%	1024 0,04%
	TOTAL	1925 12,1%	1631 2,7%	21504 0,79%

Table 4.2: Resource cost of a versatile pattern matching module

as the whole module, is a basic characteristic which directly impacts the maximum number of patterns that it will be possible to process, as well as the feasibility of deployment on embedded architectures.
Another point, which is critical in network applications, is the latency or delay, we will measure the induced latency of the matching engine in the best case (simple matching) but also in the case of complex matching, which involves more operations thus more latency. The latency of the whole security device will be given as information, to evaluate the impact of such equipment inside an existing industrial control system.

Table 4.2 summarizes the resource cost for the implementation of the versatile pattern matching module, the Synthesis and Place and Route are done with not specific optimizations of the compiler.
As expected, the patterns memory is the primary and main consumption of memory bits, less than 0,1% of the total available are used by the FIFOs for the custom mdio access. With a memory consumption under 1% of the total memory bits available we succeed in reducing the load over memory and thus leaving as much as possible for the interactions with the network (which requires extensive buffering). On the other hand we resort extensively to using logic elements, standard FSMs (not RAM-based ones) have the drawback of using lots of logic resources as the FSM grows in size. The fact that we choose to perform time aware and operational-process aware analysis increase furthermore the cost in logic resources.

Then we consider the TAGs manager, its database and the token master, whose cost in resources are shown in Table 4.3. The resources consumption of the TAGs handler is

Entity	Sub-module	ALMs	Logic Registers	Memory Bits
TAGs manager	TAGs Memory	0 0%	0 0%	2048 0,076%
	TAGs manager	84 0,53%	262 0,41%	128 >0,01%
	TOTAL	84 0,53%	262 0,41%	2176 0,077%
Token master		36 0,23%	12 0,02%	0 0%

Table 4.3: Resource cost of the TAGs handling database and the custom MDIO bus arbitrator

relatively small, the database and functions dictionary, we used in the implementation, were small so the memory bits needed were low. While the FSM which processes the TAGs request, searches both the dictionary and the database, and answer is simple. In the same way the cost of the token manager can almost be ignored as it is small.

Considering the whole matching engine, the total cost is about 2045 ALMs, 1905 Logic Registers and 23680 Memory Bits (Table 4.4), it includes Pattern matching for 2 to 8 patterns, time and operational-process awareness with TAGs management. The addition of further patterns is made by using more versatile pattern matching modules, for 2 to 8 supplementary patterns (it depends of their size, see Figure 4.6) by module, the total additional cost is available in Table 4.2.
If we consider the FPGA used in this experiment, it can fit up to 7 versatile pattern matching modules for a total of 14 to 56 patterns, while having a very low impact on the available memory resources (less than 7%). The advantage of relying extensively on logic resources, instead of memories, lies on the fact that the multiplication factor of the logical elements, from one matrix size to the upper one, is larger than for memory resources.

For example, when going from our current FPGA (5CSEA4) to the upper one (5CSEA5) the available ALMs are multiplied by two (from 15,880 to 32,075) and the memory bits by 1,5 (from 2,7M to 3,97M). To reach the targeted number of 100 patterns, the Intel Cyclone V 5CSEA5 is enough, considering that we address small patterns (Modbus one for example), else for bigger ones it would be necessary to go up to the next model (5CSEA6).

One of the fundamental parameter to take into account when working on network applications/devices is the latency that it adds on network exchanges. In our case the

Entity	ALMs	Logic Registers	Memory Bits
TAGs manager	84 0,53%	262 0,41%	2176 0,077%
Token master	36 0,23%	12 0,02%	0 0%
Versatile pattern matching modules	1925 12,1%	1631 2,7%	21504 0,79%
TOTAL	2045 12,86%	1905 3,13%	23680 0,88%

Table 4.4: Total cost of the pattern matching engine with only one versatile pattern matching module

induced delay is calculated on two levels, the latency of the pattern matching and the latency of the research architecture, the sum of both gives the total latency of the device. Concerning the latency of the pattern matching, a chronogram of processing for a pattern is shown in Figure 4.16, it concerns the most basic matching with no post-processing. We need three clock cycles to perform the pattern matching, one required by the patterns memory (when asking for the data at the current address, the memory need one clock cycle to give the output), one needed to process the pattern type (the first octet of the pattern) and the last cycle for the FSM to end the comparison. For instance three clock cycles at a frequency of 125 MHz, which is the frequency we have been using for this design, account for a latency of 24 nanoseconds, which is irrelevant in a network with common latencies in milliseconds. This delay can be reduced by reading in advance the pattern type for example but it does not seem necessary. When all the FSMs have finished their tasks, the whole matching module goes from *BUSY* to *NOT BUSY* and the *SUCCESS* is lift if a pattern was matched.

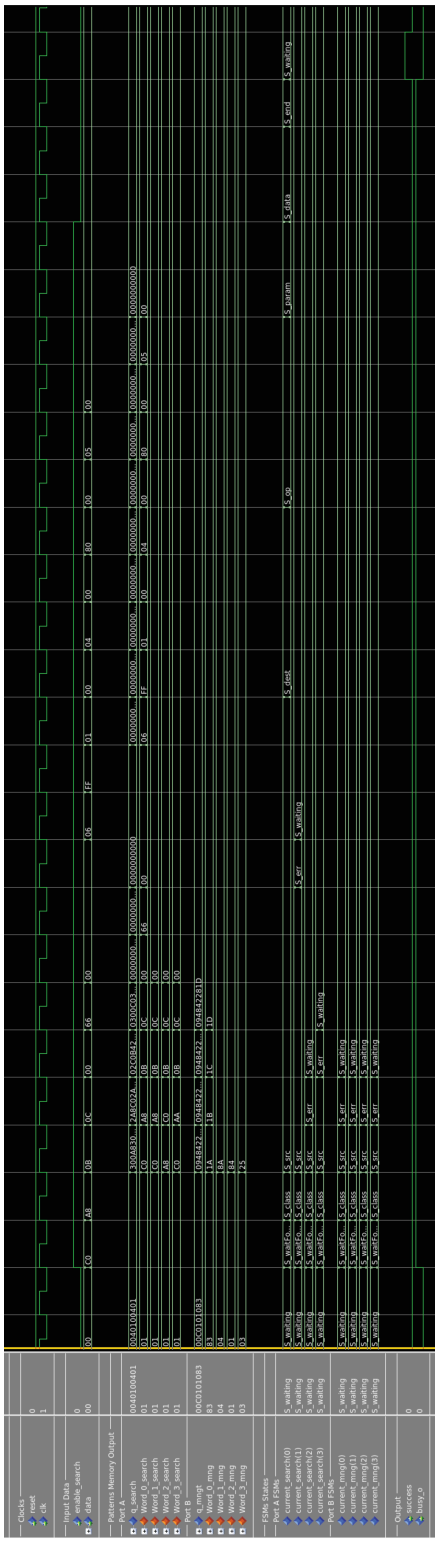


Figure 4.16: Chronogram of matching for simple patterns

Then, each post-processing and supplementary operation induces more delay, processing ranged values will increase the latency by two more clock cycles by array. The implementation we are currently using only supports two ranged values, one in the parameter field and one in the data field, for a total of four more clocks cycles in the worst case.
When speaking of the time aware processing, the latency comes with a variable value, the temporal constraint requires two clock cycles to be compared, this is a fixed delay, but monitoring and acquisition of the current time and date depends on the chosen method. If it is stored locally, inside the FPGA matrix, the delay can be just a couples of clock cycles, but if it resorts to external component such as RTC, the latency of the answer can grow up to several dozens of cycles. The way we handle operational-process awareness involves a question and answer mechanism between the matching FSM and the TAG manager, this exchanges is short and compressed in most cases but its duration can increase when multiple FSMs line up.

As it can be seen on Figure 4.17 the matching FSM asks for the TAG value and the custom MDIO driver stores and constructs the request, then the driver asks for permission to use the bus and sends the request. These operations, from the TAG request to the emission on the bus, are done in 13 cycles. The TAG manager is quick to perform the TAG search and to answer, as it is master on the bus it can use it right away and don't need to wait for the authorization to answer.
The complete question and answer process is made in 30 clock cycles, 240 ns in the best case, when multiple FSMs need to interrogate the TAGs master, the latency is multiplied by the number of FSMs. The update of a TAG value by the TAGs manager is not accounted in the latency calculation as this operation is made after the matching and has no impact on its processing time.

Table 4.5 recapitulates the induced latency of each part of the process and gives the total latency, in best and worst case, for the whole versatile pattern matching engine. For instance, the measured latency of the research platform without any filtering or pattern matching, in pure data transfer, is around 2 milliseconds. This is inherent to the platform and depends of the original architecture it is based on, so event if it is possible to optimize this latency, this is not the subject of this work and will be taken as accounted. The maximum latency of 1592 nanoseconds for our matching engine, compared to the 2 ms latency of the platform, it represents an increase of less than 0,1% of the induced delay.

Concerning the security and reliability of the solution, as said previously the patterns library works as white list, which means that only the stored patterns are allowed to pass through the engine. In one hand it puts more constraints on the patterns definition as it

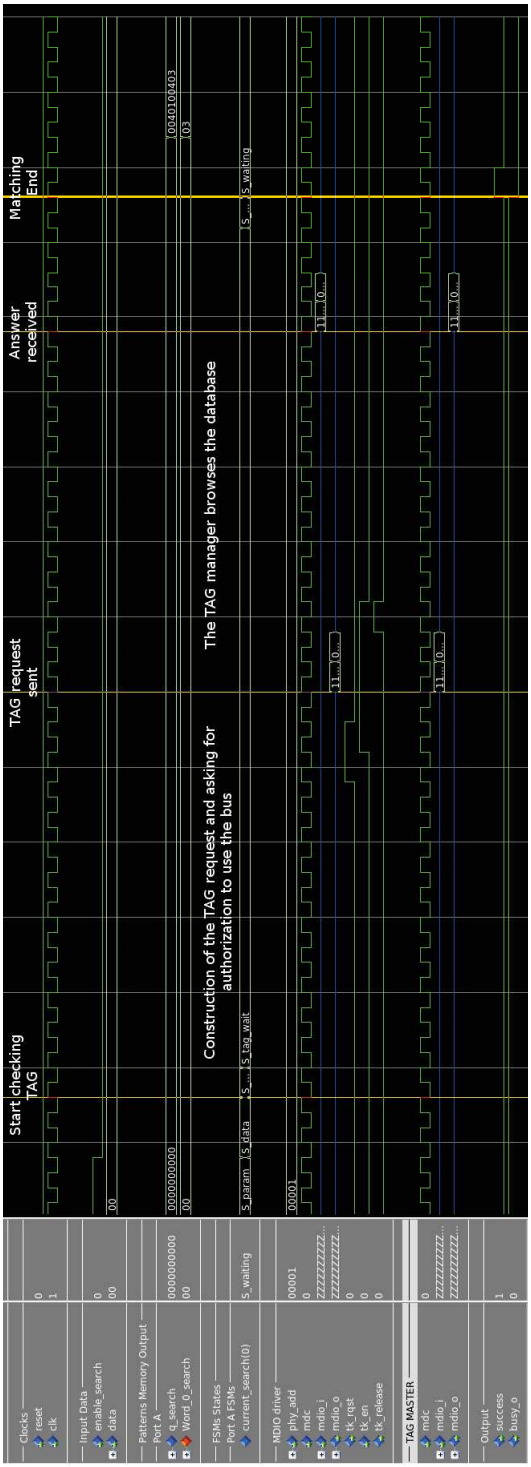


Figure 4.17: TAG request chronogram

	Latency	
	Best Case	Worst Case
Pattern Matching FSMs	3 clock cycles 24 ns	
Ranged Value (Array)	2 clock cycles 16 ns	4 clock cycles 32 ns
Time Aware Post-processor	6 clock cycles 48 ns	32 clock cycles 256 ns
Operational-process Aware Post-processor	30 clock cycles 240 ns	156 clock cycles 1248 ns
Total	43 clock cycles 344 ns	199 clock cycles 1592 ns

Table 4.5: Latencies summary

needs to be exhaustive, but it also gives more security as it prevents from many 0-days threats. In addition, as we resort to brute force matching, the theoretical error rate is null (in practice implementation errors or breaches may lead to degraded error rate). When incoming data is processed, if one matching FSM has a successful matching, it allows the data to pass through, but the default state of output is data dropped. One topic which was not addressed is about the patterns size compared to input data size. In fact both don't need to have the same size, a pattern shorter that the input data can gives a positive match but an input data shorter than the pattern will always return a negative match. Looking back at the common representation model introduced, only the [*Source*] [*Destination*] [*Operation*] fields are mandatory, so inputs can easily match a pattern of this form, but in case of a pattern which includes [*Parameter*] and [*Data*] fields, it can not match smaller inputs anymore. It was made this way to filter some potentially malformed requests that we did observe during the experiments.

4.4.4 Results

Using experimentation results, we can estimate the area cost by pattern in accord with the pattern size. So, for small patterns (under 128 bytes) we achieve a cost of 256 ALMs and 238 Logic Registers with a memory bits imprint of 2960 bits. In case of bigger patterns, the 1024 bytes one costs 1023 ALMs, 953 Logic Registers and 11840 Memory bits. As said earlier, the FPGA that we are using in this work can hold 7 occurrences of the versatile pattern matching module, for a total of 14 to 56 patterns depending of their size. For example, when working in Industrial control system using ModBus as communication protocol, we will be able to fit a maximum of 56 patterns on the FPGA, while for more complex industrial protocols, only 14 patterns will fit. On a Intel Cyclone V FPGA 5CSEA4

FPGA	Patterns Number	Cost		
		ALMs	Logic Registers	Memory Bits
5CSEA4	14 - 56	14315 90,02%	13335 21,91%	165760 6,16%
5CSEA5	28 - 112	28630 89,3%	26670 20,8%	331520 8,35%
5CSEA6	40 - 160	40900 98,53%	39100 23,54%	476600 8,56%

Table 4.6: Patterns scaling in function of the FPGA size and available resources

it is impossible to fit the targeted 100 patterns, but if we compare with bigger ones from the same products line, the 5CSEA5 has two times more ALMs and Logic Registers and 1,5 time more memory bits, while the 5CSEA6 has 2,75 times more ALMs and Logic Register, with two times more Memory bits. So on a 5CSEA5 we can fit up to 112 small patterns and 28 big ones, for a cost of 89,3% of the available ALMs, 20,8% of the registers and 8,35% of the memory bits. In the same way, a 5CSEA6 can handle 106 small patterns and 40 big ones for a cost of 98,53% of the ALMs, 23,54% of the registers and 8,56% of the memory bits. Table 4.6 summarizes these results.

But practically speaking, filling up the whole FPGA with pattern matching engines is meaningless, as we need to preserve resources for network interactions. For instance, the primary prototype we used in the earliest experiments required 6585 ALMs, 10331 Logic Registers and 975886 Memory Bits simply for network communications, and the current architecture of the research platform requires 5084 ALMs, 10442 Logic Registers and 1106432 Memory Bits alone. So, on our current FPGA (5CSEA4) we have to reserve approximately half the Memory bits and Logic registers, and 33% of the ALMs for network interconnections and data transfer, on bigger FPGA this amount is reduced. In Table 4.7 we recalculated and revalued the amount of patterns which is possible to embed on the three FPGA sizes, taking into account the resources needed by the research architecture. As expected the main bottleneck of resources is the ALMs, but in the other hand the research architecture mostly needs memory resources (for network interconnections and buffering) and we freed lots of them. The amount of patterns that we can scale on each FPGA has decreased, but as we take into account the whole architecture, it is as close to the reality as possible.

Furthermore, these numbers, on the maximum number of patterns that can be brought in the FPGA, support the distributed security architecture that we introduced earlier, as the target of 100 patterns is not reachable on our current FPGA (5CSEA4), but also on bigger. The Cyclone V 5CSEA4 can fit a maximum of 32 "small" patterns (128 bytes) which

		Cost		
		ALMs	Logic Registers	Memory Bits
5CSEA4	Research Platform	5084 33,7%	10442 17,3%	1106432 40,98%
	8 - 32 patterns	8180 54,19%	7620 12,62%	94270 3,49%
	Total	13264 87,88%	18062 29,92%	1200702 44,47%
5CSEA5	Research Platform	5084 15,85%	10442 8,14%	1106432 27,87%
	26 - 104 patterns	26585 82,88%	24765 19,30%	307840 7,75%
	Total	31669 98,73%	35207 27,44%	1414272 35,62%
5CSEA6	Research Platform	5084 12,25%	10442 6,29%	1106432 19,86%
	34 - 136 patterns	34765 83,75%	32385 19,50%	402560 7,23%
	Total	39849 96%	42827 25,79%	1508992 27,09%

Table 4.7: Realistic Patterns scaling in the research architecture

is not enough to cover the whole control system requirements in terms of security, not to mention the "big" patterns case (1024 bytes). But even the 5CSEA6 can not handle a sufficient amount of patterns if we consider more verbose industrial protocols, hence the need to divide the security analysis between multiple devices in the network. Figure 4.18 shows an example of the patterns distribution in accordance with the criticality of the equipments and the impact in terms of FPGA required by the security devices.

The highest level of security analysis, located at the networks interconnection, will need the most resources (the bigger FPGA), as it will process access control patterns, which are the most numerous. While devices which ensure security for local sub-network will need smaller FPGA as they will process less patterns, and finally for the most critical targets, with dedicated security, will accommodate with a low number of patterns and a small FPGA. It confirms that the available resources factor is one of the fundamental problematic of network security through pure hardware platform.

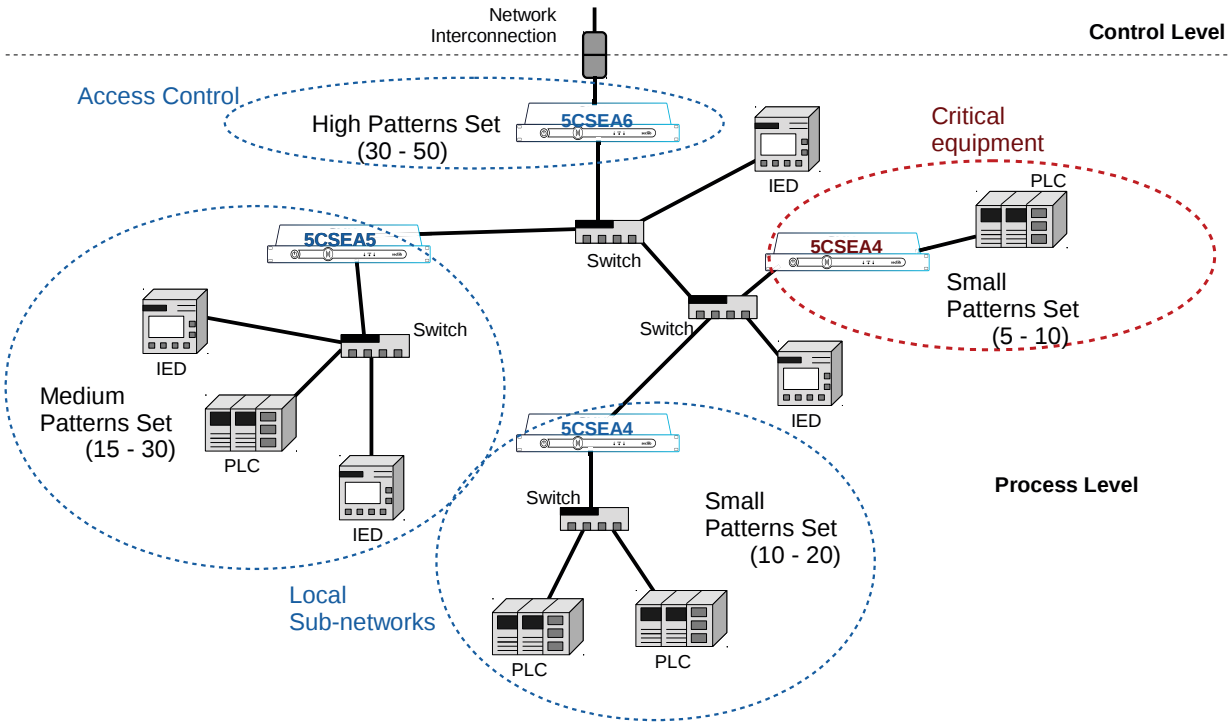


Figure 4.18: FPGA requirement of the distributed security analysis in accord with the patterns distribution

4.4.5 Discussion

This work intends to rely on brute force pattern matching to propose a hardware architecture for pattern matching than is more reliable that software implementations. It takes advantages of the flexibility offered by a FSMs based implementations to perform time and operational-process awareness through dedicated post-processors. Furthermore as we intend to address problematic from a very strict context, the FSM solution allows a perfect matching with no false positive rate. Patterns are stored using memory resources but the storage is optimized to reduce the memory imprint while processing small patterns.

While the current experiments of the suggested architecture show satisfying results, with low memory imprint and quick processing time, we intend to continue the researches through further optimizations. Even if the memory resources cost is quite low, conversely the cost in logic resources is a bit too expensive, so we intend to reduce this cost by pooling some operations or states between the multiple FSMs. An other point that we would like to improve concerns the operational-process awareness through the TAGs management. This solution is far from perfect, it would be more effective if coupled with a request system to interrogate the process on its state. Doing so we add some network traffic but

have a better precision on the process state.

Finally, the major research axis to improve this work is about regular expressions. Many firewalls and IPS/IDSs make use of regular expressions to optimize their pattern matching, SNORT and Suricata are both examples which heavily rely on them. The major advantage of this method concerns the patterns definition, as regular expressions allow to reduce the patterns size and number, a high number of characters can be represented through a short expression. Furthermore it simplifies the resort to ranged values and bits masking. So from this point it appears clearly that regular expressions are the next step of this work.

Chapter 5

Conclusion

The opening of industrial infrastructures to the Internet caused their exposure to new types of threats, whether they are network or application related. The most common counter-measures used by industrials to answer network threats are firewalls, IPSs, IDSs and other common IT solutions. But application threats show different problematic which are badly addressed by IT means, they require analysis of incoming messages based on a security policy, but in addition they require deep knowledge of the industrial context and current process. IT solutions are not designed to handle the knowledge of the industrial process, so dedicated solutions were made available, such as industrial firewalls, which take into account the application parts. But unfortunately they are quite uncommon and mainly lie on a software platform, which is very weak against variety of attacks, through the network or other means. The main problem is that, when the software platform is compromised it impacts the security process it hosts, and may allow to modify or bypass the security. In this context, the purpose of this work was to offer solutions to increase the security against application threats, either by protecting existing software solutions, or by offering hardware alternatives.

This work was organized around two major contributions, firstly to increase the security over existing software applications protection methods and then to propose a hardened security process by relying on a hardware platform. One of the research axis, for protecting the software platform of current security devices, addresses the protection against persistent threats. Ensuring the integrity and authentication of any software element during the boot process allows to be sure that after a reboot the system is in a healthy state, without modifications, and will be able to correctly perform its functions (no persistent threats). To do so, we suggested to embed software boot elements in the FPGA, allowing to make a verification chain with security inherited from the bitstream verification mechanism. Each element of the boot chain will verify the following one, and to ensure that

the first elements are not compromised, they are embedded inside the FPGA matrix (in the user logic). Doing so they benefit of the security mechanisms put in place by manufacturer to ensure the security of the bitstream. Then we used dynamic reconfiguration to reduce the footprint, of the stored boot elements, over the FPGA resources. We achieved nearly no resource consumption through the usage of two bitstreams, one with the software boot elements and the second with only the user features. Even the slight increase of the boot delay caused by the reconfiguration can be ignored compared to the boot time of the operating system.
This work is part of the current trend around roots of trust for embedded systems, even if our proposed architecture is more restrictive compared to other solutions. The fact that we depend on the FPGA for most of the security of the low level boot elements, makes this contribution unusable for CPU-only based architectures. Still, U-boot is a privileged tool, with kernel verification, widely used in the literature and easily portable on other platforms. Furthermore one of the major flaw, that we have willfully chosen not to address here, concerns the communications with the RAM. As many of the cryptographic operations, which the security is based on, are made in the RAM, any probing or attack may endanger the whole chain of trust.

Secondly, we suggested a versatile pattern matching engine relying on FPGA's hardware resources to offer increased security. The pattern matching is the core process of many security operations, as it allows to quickly compare incoming network data to a list of patterns, identifying harmful messages and threats. Using hardware resources to perform this security analysis has two benefits, the first is about performances, as hardware is faster than software, and the second lies on the increased security of the hardware architecture. Hardware is less subject to compromise because it is harder to modify and so it is less vulnerable to attacks. But to ensure correct application check, pattern matching is needed in addition to time and operational-process state awareness. We propose an architecture based on FSMs, for brute force pattern matching, and with a dedicated post processor for time awareness. The operational-process awareness is achieved through TAGs management, a post processor links orders that change the state of the process to unique TAGs, which are requested by the matching engine to check the system state in accordance with the security policy and current matched pattern. We achieved a suitable resources cost with a limited memory imprint for the patterns storage, and managed to fit up to 56 modbus patterns in our experiment platform with a Cyclone V 5CSEA4 FPGA. While the delay induced by the pattern matching is relatively low, we achieved 344 nanosecond of additional delay for a matching with time and operational-process state awareness, far from the milliseconds delay of the network.
Today's trends in terms of pattern matching tend to promote regular expressions, as it

excels in terms of flexibility and storage optimization. We have willfully chosen to work with brute force matching, without regular expressions, as we intended to define and test our architecture with the simplest form of matching, but we intend to switch to it in the future. We are obviously less optimized than widespread solutions which use regular expressions, such as Snort, as we require more patterns. Because we targeted to use hardware architecture and resources only, we benefit from faster processing speed and stronger security, where most of available solutions and research works are software based. The process state awareness we introduced in this contribution is far from being perfect, as we chose not to interrogate the process but instead to extrapolate its state from probed communications, but still, taking into account the process state is rarely done by common security applications. We are aware that the suggested architecture presented in this work is not the most efficient when addressing recent industrial protocol like OPC-UA. But currently the majority of industrial infrastructures make use of older protocols and were constructed to last at least for 20 to 50 years. This contribution is more suited to target these infrastructures, that, despite the evolution of the protocols, will require state of the art security solutions.

From this point there is multiple leads to further continue this research work. Concerning the boot security, in a short-term perspective we are currently working on the root Image verification. The current secure boot, introduced in this work, suspends the boot process to perform the verification of the whole file system image. Instead, we target an alternative where the memory segments of the flash memory that contains this image are individually signed, then they are verified during runtime when they are read. It should reduce the boot delay and the whole root file system is still verified, as its signature is the result of the signatures of all its segments. In a medium to long term perspective, there is two ways to improve this work, as both bootloaders are located in the bitstream, their update could either be done by taking them out or by updating the bitstream itself.

- In the first case, one suggestion would be to encrypt both bootloaders and store them in an external memory, the same EPCQ used for the bitstream for example. Then the driver used by the FPGA to access this memory will be located in the user logic and slightly modified to allow on the fly decryption of the memory content. Doing so, both bootloaders can be updated, they remain confidential and an encrypted signature can be added for integrity check.

- In the second case, keeping both bootloaders inside the bitstream and wanting to update them forces to put in place a remote update mechanism. This will need dedicated logic, a secured link between the FPGA and the update operator, shared

secrets.

Furthermore, as the boot of the operating system is based on the bitstream, if it is corrupted, or not correctly loaded, the whole SoC (System on Chip) will be unusable. It would require some form of backup to allow the FPGA to fall back on a controlled state and permit a recovery or an exterior intervention. Xilinx natively supports fallback on a user specified bitstream when the initial one fail (since the Spartan 6), while Altera gives tools to the user to perform the same. This contribution heavily relies on the fact that the hardware is trusted, physical components as well as memories and communication bus. So the compromise or attacks of any of them will endanger the boot process, it is vulnerable to fault injections, probing of memories might allow to retrieve critical information. In the future it would be interesting to use dedicated protecting methods such as memory encryption, fault resistant algorithms to cover these vulnerabilities.
This work was originally designed for industrial architectures, but the principles behind it stay true for many embedded architectures, with the rise of the IOT we can easily imagine using this secure boot for FPGA based IOT devices.

Concerning the pattern matching work, in a short-term we intend to adapt the matching engine to support regular expression matching. Relying on regular expressions would allow to reduce furthermore the memory usage for the patterns storage as well as it allows more freedom and adaptability in the patterns definition. Brute force matching is a proven solution, but with the increase in size and complexity of new industrial protocols, it will become necessary to use regular expressions. Actually the patterns definition and writing is done manually, through some bash and python scripts, a first improvement would be to propose a graphical interface for the administrator. But manual definition requires deep knowledge of the network and process, it can gather multiple trade associations. Among possibilities to pursue this contribution, the development of machine learning techniques and Artificial Intelligence researches, opens new areas of work. The patterns definition would be automated through these methods, a dedicated entity could scrutinize the network traffic, learn the patterns and even react to 0 day attacks by identifying suspect behaviors. Furthermore, machine learning may help the matching process, by learning which are the most used patterns, identifying critical paths and transitions in the state automatons, it could optimize the matching and increase the efficiency. The usages are numerous, but machine learning will be a good way to simplify and optimize complex and laborious learning processes for human administrators.
Among popular trends, the Blockchain is also one of the improvement methods that should be retained. We can imagine a network architecture where each control device (PLCs, RTUs, IEDs) is part of a Blockchain which will give its approval to each command entering the network. Doing so the security process is divided between each device of the

network and not centralized anymore.
Finally, security recommendations tend to advocate for encryption to secure network communications. The analyse and matching of encrypted data (without the decryption key) is a research topic that requires dedicated algorithms, and it is one of the long-term subjects that will need attention.

Chapter 6

Publications relative to the study

6.1 International Conferences

- Peter Rouget, Benoît Badrignans, Pascal Benoit, Lionel Torres, *SecBoot - Lightweight secure boot mechanism for Linux-based embedded systems on FPGAs*, ReCoSoC 2017.
- Peter Rouget, Benoît Badrignans, Pascal Benoit, Lionel Torres, *FPGA implementation of pattern matching for Industrial Control Systems*, RAW, IPDPS 2018.

Bibliography

[1] N. Falliere, L.O. Murchu, E. Chien, *W32.stuxnet dossier*, Symantec, Tech. Rep., 2011.

[2] B. Galloway, G.P. Hancke, *Introduction to Industrial Control Networks*, IEEE Communications Surveys & Tutorials, 2012.

[3] J. Libardo Sanchez Torres, *Vulnerability, Interdependencies and risk analysis of coupled infrastructures : power distribution network and ICT*, Electric power, Université de Grenoble, 2013.

[4] B. Zhu, A. Joseph, S. Sastry, *A taxonomy of cyber attacks on scada systems*, International Conference on Internet of Things and 4th International Conference on Cyber, Physical and Social Computing, 2011.

[5] P. Huitsing, R. Chandia, M. Papa, and S. Shenoi, *Attack taxonomies for the modbus protocols*, International Journal of Critical Infrastructure Protection, 2008.

[6] Y. Xu, Y. Yang, T. Nanjing, J. Ju, Q. Wang, *Review on Cyber Vulnerabilities of Communication Protocols in Industrial Control Systems*, 2017 IEEE Conference on Energy Internet and Energy System Integration, 2017.

[7] Z. Drias, A. Serhrouchni, O. Vogel, *Taxonomy of attacks on Industrial Control protocols*, International Conference on Protocol Engineering and International Conference on New Technologies of Distributed Systems, 2015.

[8] P. Kocher, R. Lee, G. McGraw, A. Raghunathan, S. Ravi, *Security as a new dimension in embedded system design*, Proceedings. 41st Design Automation Conference, 2004.

[9] Altera Corporation, *White Paper - Design Security in Stratix III Devices*, 2009.

[10] Microsemi Corporation, *Introduction to Implementing Design Security with Microsemi SmartFusion2 and IGLOO2 FPGAs*, 2013.

[11] K. Wilkinson, *Using Encryption to Secure a 7 Series FPGA Bitstream*, Application Note: 7 Series FPGAs, Xilinx, 2015.

[12] K. Chawla, T. Siddiqua, *Mutation Resistant Runtime Code using Kernel Attestation*, OS-SEC project, 2009.

[13] S. Drimer, M. Kuhn, *A Protocol for Secure Remote Updates of FPGA Configurations*, Reconfigurable Computing: Architectures, Tools and Applications, 2009.

[14] Lattice corporation, *LatticeXP2 Family Handbook*, 2012.

[15] B. Badrignans, *Using FPGAs for security-sensitive applications*, Université Montpellier II - Sciences et Techniques du Languedoc, 2009.

[16] F. Devic, *Securing embedded systems based on FPGA technologies*, Université Montpellier II - Sciences et Techniques du Languedoc, 2012.

[17] K. Kepa, F. Morgan, K. Kosciuszkiewicz, T. Surmacz, *SeReCon: a secure reconfiguration controller for self-reconfigurable systems*, International Journal of Critical Computer-Based Systems, 2010.

[18] Discretix corporation, *Discretix Secure Boot (DxSB)*.

[19] Atmel corporation, *Safe and Secure Bootloader Implementation*, 2006.

[20] J. de Cesare, *US 2009/0257595*, Apple corporation, 2009.

[21] ARM corporation, *Building a Secure System using TrustZone Technology*, 2009.

[22] Microsemi Corporation, *Microsemi secure boot reference design – White Paper*, 2014.

[23] T. Kai, X. Xin, C. Guo, *The Secure Boot of Embedded System Based on Mobile Trusted Module*, Intelligent System Design and Engineering Application, 2012.

[24] J.R. Moyne, D.M. Tilbury, *The Emergence of Industrial Control Networks for Manufacturing Control, Diagnostics, and Safety Data*, Proceedings of the IEEE, 2007.

[25] Z. Shi, C. Ma, J. Cote, B. Wang, *Hardware Implementation of Hash Functions*, Introduction to Hardware Security and Trust, 2011.

[26] H. Li, C. Miao, *Hardware Implementation of Hash Function SHA-512*, First International Conference on Innovative Computing, Information and Control - Volume I, 2006.

[27] K.M. Binu, A.I. Neethu, *Ternary Content Addressable Memory*, International Journal of Recent Trends in Engineering & Research, 2016.

[28] S. Hachana, F. Cuppens, N. Cuppens-Boulahia, *Towards a new generation of industrial firewalls: Operational-process aware filtering*, 14th Annual Conference on Privacy, Security and Trust (PST), 2016.

[29] S. Dharmapurikar, P. Krishnamurthy, T. Sproull, J. Lockwood, *Deep Packet Inspection using Parallel Bloom Filters*, IEEE Micro, 2004.

[30] J. Crenne, *Sécurité Haut-debit pour les Systèmes Embarqués à base de FPGAs*, 2012.

[31] K. Al-Khamaiseh, S. ALShagarin, *A Survey of String Matching Algorithms*, International Journal of Engineering Research and Applications, 2014.

[32] P. Michailidis, K. Margaritis, *On-line String Matching Algorithms: Survey and Experimental Results*, International Journal of Computer Mathematics, 2001.

[33] D. Knuth, J. Morris, V. Pratt, *Fast pattern matching in strings*, SIAM Journal on Computing, 1977.

[34] R. Boyer, J. Moore, *A fast string searching algorithm*, Communications of the ACM, 1977.

[35] A. Aho, M. Corasick, *Efficient string matching: An aid to bibliographic search*, Communications of the ACM, 1975.

[36] G.F. Ahmed, N. Khare, *Hardware based String Matching Algorithms: A Survey*, International Journal of Computer Applications, 2014.

[37] E. Azimi, M.B. Ghaznavi-Ghoushchi, A.M. Rahmani, *Implementation of Simple SNORT Processor for Efficient Intrusion Detection Systems*, IEEE International Conference on Intelligent Computing and Intelligent Systems, 2009.

[38] E-ISAC, *Analysis of the Cyber Attack on the Ukrainian Power Grid, Defense Use Case*, 2016.

[39] www.modbus.org

[40] www.profibus.fr

[41] www.bacnet.org/

[42] opcfoundation.org

[43] www.seclab-security.com/

[44] M.T. Rose, D.E. Cass, *RFC1006 - ISO Transport Service on top of the TCP*, Northrop Research and Technology Center, May 1987.

[45] A. McKenzie, *RFC905 - ISO Transport Protocol Specification ISO DP 8073*, April 1984.

[46] Y. Pouffary, A. Young, *RFC2126 - ISO Transport Service on top of TCP (ITOT)*, March 1997.

[47] Z. Feng, S. Qin, X. Huo, P. Pei, Y. Liang, L. Wang, *Snort improvement on Profinet RT for Industrial Control System Intrusion Detection*, 2nd IEEE International Conference on Computer and Communications (ICCC), 2016.

[48] *Courtesy: Control Engineering Mobility, Ethernet and Wireless Study*, November 2013.

[49] M. Wollschlaeger, T. Sauter, J. Jasperneite, *The Future of Industrial Communication: Automation Networks in the Era of the Internet of Things and Industry 4.0*, IEEE Industrial Electronics Magazine, 2017.

[50] J.D. Decotignie, *A perspective on Ethernet-TCP/IP as a fieldbus*, IFAC international conference on fieldbus systems and their application, 2001.

[51] Y. Lai, J. Tai, *Network Security Improvement with Isolation Implementation Based on ISO-17799 Standard*, International Conference on Network-Based Information Systems, 2007.

[52] B. Sharma, B. Bajaj, *Packet Filtering using IP Tables in Linux*, International Journal of Computer Science Issues, 2011.

[53] K. Joshi, T. Kashiparekh, *Implementing Firewall using IP Tables in Linux*, International Journal of Emerging Trends in Science and Technology, 2016.

[54] www.checkpoint.com

[55] www.fortinet.com

[56] www.cisco.com

[57] www.stormshield.com

[58] www.tofinosecurity.com

[59] A. Jones, *Netfilter and IPTables: A Structural Examination*, 2004.

[60] www.snort.org

[61] R. Ajami, A. Dinh, *Design a Hardware Network Firewall on FPGA*, 24th Canadian Conference on Electrical and Computer, 2011.

[62] A. Kumar, A. Mittal, A. Gupta, S. Ghosh, *Firewall Implementation*, CS425: Computer Networks, 2010.

[63] A. Zhiyuan, L. Haiyan, *Realization of Buffer Overflow*, International Forum on Information Technology and Applications, 2010.

[64] R. Johari, P. Sharma, *A Survey on Web Application Vulnerabilities (SQLIA, XSS) Exploitation and Security Engine for SQL Injection*, International Conference on Communication Systems and Network Technologies, 2012.

[65] M. Khari, P. Sangwan, Vaishali, *Web-application attacks: A survey*, 3rd International Conference on Computing for Sustainable Global Development, 2016.

[66] A.W. McMorran, *An Introduction to IEC 61970-301 & 61968-11: The Common Information Model*, 2007.

[67] V. Schiffer, *The Common Industrial Protocol (CIP^{TM}) and the Family of CIP Networks*, ODVA, 2016.

[68] R. Elbaz, L. Torres, G. Sassatelli, P. Guillemin, M. Bardouillet, A. Martinez, *A parallelized way to provide data encryption and integrity checking on a processor-memory bus*, Proceedings of the 43rd annual Design Automation Conference, New York, USA, 2006.

[69] J. Crenne, R. Vaslin, G. Gogniat, J. Diguet, R. Tessier, D. Unnikrishnan, *Configurable Memory Security In Embedded Systems*, ACM Transactions on Embedded Computing Systems, 2013.

[70] L. Bossuet, G. Gogniat, W. Burleson, *Dynamically Configurable Security for SRAM FPGA Bitstreams*, International Journal of Embedded Systems, 2016.

[71] M. Vaˇsut, *Secure and flexible boot with U-Boot bootloader*, 2014.

[72] Gentoo Fundation, *Initramfs/Guide*, 2016.

[73] *The GNU Privacy Guard*, 2015. https://www.gnupg.org/

[74] *Secure Boot with SecBus*. https://secbus.telecom-paristech.fr/wiki/SecureBoot

[75] J. Teki, *U-Boot: Verified RSA – Boot on ARM target*, U-boot Mini Summit, Edinburgh, 2013.

[76] C.P. Wedage, *Efficient Content Addressable Memory Design Using RAM*, International Journal of Electronics and Electrical Engineering, 2016.

[77] K. Al-Khamaiseh, S. ALShagarin, *A Survey of String Matching Algorithms*, Journal of Engineering Research and Applications, 2014.

[78] S. Mocanu, M. Kabir-Querrec, J. Thiriet, E. Savary, *Cybersécurité des sous-stations électriques IEC 61850*, 2015.

[79] www.netfilter.org

[80] www.pfsense.org

[81] T.J.Williams, *The Purdue enterprise reference architecture*, Comuters in Industry Vol 24, 1994.

[82] C. Shields, *What do we mean by Network Denial of Service ?*, 2002.

ABSTRACT: Study and design of mechanisms for rupture and filtering of industrial protocols

With the rise of Industry 4.0, many infrastructures were forced to open their networks to the Internet, mainly to meet the growing need for supervision and remote control. But where these infrastructures were previously isolated, spared from external threats, their opening has caused the emergence of new threats, particularly network ones, which were not addressed and present serious risks.

Network cybersecurity solutions, like Firewalls, Intrusion Detection Systems or Intrusion Protection Systems are commonly used to address the concern of industrial infrastructures cybersecurity. However the trend of relying on software-based systems to ensure network protection brought to light the vulnerabilities of these systems, due to their inherent software implementation. Furthermore, the industry is tied to its own specificities (low-latency, support of specific network protocols), which are rarely covered by common IT solutions.

The main goal of this thesis is to study the use of FPGA-based devices applied to cybersecurity for industrial networks. Either as support for software-based security applications, or to perform critical network analysis operations. First it presents the industrial context, with control systems, their architectures, needs, implementation rules, specific protocols and also gives two examples of control systems as they can be found in the industry. Then it highlights the security problematic, with a description of the most common threats, cases study about their applications and impact in a control system, and discussions on the state of the art countermeasures available on the market. Through the establishment of a security target, it points the vulnerability of software elements and operating systems as well as the lack of process state aware security analysis.

To address these issues, we propose, through a first contribution, to enforce the security of the software system by taking advantage of existing FPGA's protection mechanisms. Finally, to answer specific application threats, we introduce an implementation of a brute force matching architecture with time and operational-process awareness, on FPGA.

This thesis was conducted in collaboration between the Montpellier computer science, robotic and microelectronic laboratory (LIRMM) and the SECLAB company.

RÉSUMÉ: Étude et conception de mécanismes de rupture et de filtrage de protocoles industriels

Avec l'essor de l'Industrie 4.0, de nombreuses infrastructures ont été contraintes d'ouvrir leurs réseaux à Internet, principalement pour répondre au besoin croissant de supervision et de contrôle à distance. Mais là où ces infrastructures étaient auparavant isolées, épargnées par les menaces extérieures, leur ouverture a provoqué l'émergence de nouveaux risques, en particulier à travers le réseau, potentiellement sérieux et qui ne sont pas couverts.

Les solutions de cybersécurité, comme les pare-feux, les systèmes de détection d'intrusion ou les systèmes de protection contre les intrusions, sont couramment utilisés pour répondre aux préoccupations liées à la cybersécurité des infrastructures industrielles. Cependant, la tendance à se fier aux systèmes logiciels pour assurer la protection du réseau a mis en lumière les vulnérabilités de ces systèmes, en raison de leurs implémentations logicielles inhérentes. En outre, l'industrie est liée à ses propres spécificités (faible latence, support de protocoles réseaux spécifiques), qui sont rarement couvertes par les solutions informatiques communes.

L'objectif principal de cette thèse est d'étudier l'utilisation de dispositifs FPGA appliqués à la cybersécurité pour les réseaux industriels, soit comme support pour des applications de sécurité logicielle, soit pour effectuer des opérations critiques d'analyse réseau. Ce travail présente d'abord le contexte industriel, avec les systèmes de contrôle, leurs architectures, leurs besoins, les règles de mise en œuvre, les protocoles spécifiques et donne également deux exemples de systèmes de contrôle comme on peut en trouver dans l'industrie. Il met ensuite en lumière les problèmes de sécurité, avec une description des menaces les plus courantes, des études de cas sur leurs applications et leurs impacts dans un système de contrôle, et des discussions sur les contre-mesures de pointe disponibles sur le marché. Suite à l'établissement d'une cible de sécurité, nous mettrons en évidence la vulnérabilité des éléments logiciels et des systèmes d'exploitation. Nous verrons aussi comment l'absence d'analyse de sécurité tenant compte de l'état des processus peut mener à certaines vulnérabilités.

Pour pallier à ces problèmes, nous proposons, par une première contribution, de renforcer la sécurité des systèmes logiciels en tirant parti des mécanismes de protection existants du FPGA. Enfin, pour répondre à des menaces applicatives spécifiques, nous proposons la mise en œuvre d'une architecture de reconnaissance de motifs, sur FPGA, prenant en considération le cadre temporel et l'état du procédé industriel. Cette thèse a été réalisée en collaboration avec le Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier (LIRMM) et la société SECLAB.

MOTS-CLÉS: FPGA, Sécurité, Reconnaissance de motifs, Protocoles industriels