SPARTA

D6.1

Security-by-Design Framework for the Intelligent Infrastructure

Project number	830892
Project acronym	SPARTA
Project title	Strategic programs for advanced research and technology in Europe
Start date of the project	1st February, 2019
Duration	36 months
Programme	H2020-SU-ICT-2018-2020

Deliverable type	Report
Deliverable reference number	SU-ICT-03-830892 / D6.1/ V1.0
Work package contributing to the deliverable	WP6
Due date	January 2020 – M12
Actual submission date	4th February, 2020

Responsible organisation	CINI
Editor	Gabriele Costa
Dissemination level	PU
Revision	V1.0

Abstract	Intelligent infrastructures (II) are becoming a cornerstone of our modern society and their security is crucial. As many existing II were built without security in mind, they are now exposed to sever risks and require costly and inefficient re-engineering. This document investigates the technologies for making II *secure-by-design*. Such technologies converge in the definition of a secure orchestration framework and toolkit for the II.
Keywords	Intelligent infrastructure, secure orchestration, security-by-design

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 830892

Editor

Alessandro Armando, Gabriele Costa (CINI)

Contributors

Joaquin Garcia-Alfaro, Jean-Max Dutertre, Jean-Luc Danger (IMT)
Gianluca Roascio, Paolo Prinetto, Gabriele Costa, Alessandro Armando (CINI)
Michel Hurfin, Ludovic Mé (Inria)
Giorgio Bernardinetti, Francesco Mancini (CNIT)
Sergej Proskurin, Claudia Eckert (TUM)
Uwe Roth, Qiang Tang (LIST)
Lukas Malina, Petr Dzurenda (BUT)
Manon Knockaert, Jean-Marc Van Gyseghem (UNamur)
Raimundas Matulevicius, Abasi-Amefon O. Affia, Kaspar Kala (UTartu)
Branka Stojanović, Katharina Hofer-Schmitz (JR)
Marek Pawlicki (ITTI)
Tewodros Beyene (FTS)
Nerijus Morkevičius (KTU)

Reviewers

Algimantas Venčkauskas (KTU)
Romain Ferrari (THALES)

Disclaimer

Executive Summary

This document presents the roadmaps of the HAII-T Program. Roadmaps are defined both individually for each Task and collectively for the integration process. The document starts by discussing the state-of-the-art of the defense mechanisms for the legacy components. These components have a central role in most Intelligent Infrastructure (II) and there exist a rich literature of attacks targeting them. Then, the focus moves to security of the operating systems and software for the Internet of Things (IoT) and field devices. A secured OS is a crucial to establish security properties of a system at a higher, application level. The technologies for the orchestration of complex, secure-by-design II are also discussed. Secure orchestration ensures that the security requirements are respected allover the II lifecycle. Then, other two crucial aspects of the secure design of every II, i.e., *resilience-by-design* and *privacy-by-design*, are presented. A resilient infrastructure ensures the continuity and the quality of service over time, while the privacy guarantees protect the confidentiality of critical data against potential leakages. Finally, the integration roadmap is also presented and use cases, challenges and milestones of the Program are identified.

Table of Content

Chapter 1 Introduction . . . 1

Chapter 2 Hardening Legacy Components . . . 2

2.1 Analysis of Existing Defense Mechanisms . . . 2
2.1.1 Defenses against Memory Corruption Vulnerabilities . . . 2
2.1.2 Defenses against Fault Injections . . . 11
2.2 Devising General Defense Mechanisms . . . 19
2.2.1 Defenses against Data-oriented Attacks through Virtualization . . . 19
2.2.2 Defenses against Code-oriented Attacks through Hardware-based Monitoring . . . 24
2.2.3 Defenses against Fault Injection Attacks for Present and Future Devices . . . 26

Chapter 3 Securing Operating System Software . . . 27

3.1 Embedded OS Primitives for Future-proof Security . . . 27
3.1.1 IoT crypto primitives . . . 27
3.1.2 Formally Verified Crypto Primitives . . . 27
3.1.3 Embedded primitives for secure multi-tenant IoT software . . . 28
3.2 Low-power & Secure Network . . . 29
3.2.1 Zero-touch secure low-power network bootstrap . . . 29
3.2.2 Light-weight end-to-end security at transport layer and above . . . 30
3.3 Securing the Supply Chain of OS Software . . . 31
3.3.1 Automated security assessment & policies for IoT application update software binary bundles . . . 31
3.3.2 Integrated prototype of secure IoT software update for RIOT . . . 32

Chapter 4 Secure Orchestration of the Intelligent Infrastructure . . . 34

4.1 Security Orchestration Framework . . . 34
4.1.1 Intelligent Infrastructure Lifecycle . . . 34
4.1.2 Infrastructure Provisioning . . . 34
4.1.3 TOSCA . . . 36
4.2 Formal Verification of Protocols . . . 38
4.2.1 Formal Methods . . . 39
4.2.2 Smart Home Scenario . . . 40
4.3 Formal evidence language for the II . . . 42
4.3.1 Evidence-based security frameworks . . . 42
4.3.2 Formal evidence language . . . 42
4.3.3 State of the art for evidence languages . . . 42
4.3.4 Semantic modelling aspects . . . 43
4.3.5 Evidence and certificate formats . . . 44
4.4 Multi-layered security model for Fog computing . . . 44
4.4.1 The overview of Fog computing paradigm . . . 44
4.4.2 Main issues of the Fog computing . . . 45
4.4.3 Existing solutions to Fog computing related issues . . . 47
4.4.4 Fog orchestration solutions . . . 48
4.4.5 Security orchestration challenges in Fog computing . . . 50
4.5 Key Components of an Intelligent Infrastructure . . . 52
4.5.1 Components of an Intelligent Infrastructure . . . 52
4.5.2 Key Components Classification . . . 54

Chapter 5 Resilience-by-design of Intelligent Infrastructures (II) . . . 56

5.1 Intrusion detection . . . 56
5.1.1 Signature-based intrusion detection . . . 57
5.1.2 Anomaly-based intrusion detection . . . 58
5.1.3 Control-theoretic detection of cyber-physical attacks . . . 62
5.2 Resilience techniques based on fault and intrusion tolerance . . . 66

5.2.1 Intrusion tolerance concepts . . . 66
5.2.2 Intrusion Tolerance Strategies . . . 68
5.3 Final remarks . . . 69

Chapter 6 Privacy-by-Design . . . 70

6.1 Analysis of Privacy Threats and Attacks: Technical Attacks, Privacy-Based Leakages, and Social Aspects . . . 70
6.1.1 Introduction . . . 70
6.1.2 Asset Description . . . 70
6.1.3 Privacy Threats and Leakages . . . 70
6.1.4 Concluding Summary . . . 72
6.2 Privacy-Preserving Management and Regulations . . . 73
6.2.1 Challenges . . . 73
6.2.2 Privacy-Preserving Management: a GDPR Perspective . . . 73
6.2.3 Privacy by Design . . . 75
6.3 Privacy-Enhancing Technologies: Readiness Analysis for the Adoption and Evaluation of Privacy-Enhancing Technologies for Intelligent Infrastructures . . . 76
6.3.1 Privacy-Enhancing Digital Signatures . . . 76
6.3.2 Privacy-Enhancing User Authentication . . . 77
6.3.3 Privacy-Enhancing Communication Systems . . . 78
6.3.4 Privacy-Enhancing Encryption Technologies . . . 78
6.3.5 Privacy-Enhancing Computations and Data Storing . . . 79
6.3.6 General Anonymisation Techniques . . . 79
6.3.7 Concluding Summary . . . 80
6.4 Privacy Evaluation Techniques and Methods . . . 80
6.4.1 Introduction . . . 80
6.4.2 Tool-Assisted Methodology for Data Protection Impact Assessment . . . 81
6.4.3 Privacy Languages for Data Policies . . . 82
6.4.4 Privacy Policy Enforcement via Encryption Schemes . . . 83

Chapter 7 Integration Roadmap . . . 84

7.1 Integration challenges and review of the technologies . . . 84
7.2 Use cases . . . 85
7.2.1 Smart building . . . 85
7.2.2 xMP . . . 85
7.2.3 Privacy-Preserving Internet of Vehicles . . . 86
7.3 Integration milestones, risks and mitigation strategy . . . 86
7.3.1 Milestones . . . 86
7.3.2 Risks and mitigation . . . 87

Chapter 8 Conclusions . . . 88

Chapter 9 List of Abbreviations . . . 90

Chapter 10 Bibliography . . . 91

List of Figures

Figure 2.1 Return address corruption, start point of ROP attacks. . . . 7
Figure 2.2 A basic example of ROP attack. . . . 7
Figure 2.3 Test code for instruction skip analysis . . . 13
Figure 2.4 EM-induced single instruction skip: effect on execution time (a), ability to choose the skipped instruction (b) . . . 14
Figure 2.5 EM-induced instruction skip: ability to skip two consecutive instructions (a), effect of the voltage pulse duration (b) . . . 14
Figure 2.6 Laser-induced instruction skip at 0.4 W and 75 ns: effect on execution time (a), ability to choose the skipped instruction (b) . . . 15
Figure 2.7 laser duration from 50 ns to 410 ns, 0.4 W laser power . . . 15
Figure 2.8 Bypass of a verify PIN algorithm with four separate laser pulses . . . 16
Figure 2.9 Performance impact of xMP on Nginx with varying file sizes and number of connections (X-axis: [file size (KB)]-[# requests]) [288]. . . . 24

Figure 4.1 An abstract representation of the II lifecycle. . . . 35
Figure 4.2 Layered view of a virtualized II. . . . 36
Figure 4.3 A generic IaC specification for the example II. . . . 37
Figure 4.4 An excerpt from the (TOSCA-style) diagram for the virtual II. . . . 37
Figure 4.5 An excerpt of a TOSCA specification. . . . 38
Figure 4.6 Properties distribution in the Multi-layered Fog Computing architecture [401] . . . 45
Figure 4.7 Fog and Cloud service partitioning example . . . 48
Figure 4.8 Orchestration framework architecture . . . 49
Figure 4.9 Fog computing orchestration architecture with flat control layer [231]. . . . 50
Figure 4.10 Fog orchestrator architecture [379]. . . . 50
Figure 4.11 Fog orchestrator interaction with Fog node [69]. . . . 51
Figure 4.12 Security orchestration challenges in Fog computing. . . . 51
Figure 4.13 IoT technology architecture . . . 54

Figure 5.1 Signature-based Approach for Cyber-attack Detection . . . 57
Figure 5.2 Anomaly Detection Approach for Cybersecurity Detection . . . 59
Figure 5.3 Cyber-physical adversarial model . . . 63
Figure 5.4 Cyber-physical replay attack . . . 63
Figure 5.5 Cyber-physical stealthy attack using either bias injection techniques . . . 64
Figure 5.6 Cyber-physical covert disruption attack . . . 64
Figure 5.7 AVI composite fault model. . . . 67
Figure 5.8 Preventing security failure. . . . 67

Figure 6.1 Method for achieving regulation compliance, adapted from [143] [201]. . . . 76
Figure 6.2 Privacy-enhancing technologies in the intelligent infrastructure environment. . . . 80
Figure 6.3 An overview of our methodology. . . . 81

Figure 7.1 Smart building network scheme. . . . 86

List of Tables

Table 2.1 Registers r16 to r25 readback values, for a fault free execution (top) and for an instruction skip targeting r19 (bottom, highlighted in red). . . . 13
Table 2.2 Number of obtained instruction skips vs laser pulse duration . . . 16
Table 2.3 Redundancy-based software defense against fault injection - fault injection detection [32] 17
Table 2.4 Duplication-based software defense against the single instruction skip fault model [260] 18
Table 2.5 Performance overhead of protection domains for page tables, process credentials, and both, measured using LMbench v3.0 [288]. . . . 23
Table 2.6 Performance overhead of protection domains for page tables, process credentials, and both, measured using Phoronix v8.6.0 [288]. . . . 23

Table 4.1 Overview of Smart Home protocols . . . 41
Table 4.2 Software components of Intelligent Infrastructures . . . 53

Table 5.1 Sample list of attacks reported in the control-theoretic literature . . . 62

Table 6.1 Summary of technical threats . . . 71
Table 6.2 Categories of privacy-enhancing (PE) technologies. . . . 77

Table 7.1 Table of technologies investigated within HAII-T Program. . . . 85

Chapter 1 Introduction

The development of modern II promises to raise the bar for several aspects of our everyday life. This includes faster and improved services for the citizens as well as integrated solutions for industrial production and the supply chain. The pillars of this evolution consist of several technologies including, e.g., IoT, Fog and Cloud computing as well as telecommunication infrastructures.

As often happens, along with opportunities, these technologies also bring new risks. Perhaps the main concern is the possible presence of vulnerabilities that may enable attackers to steal or affect sensitive data or control devices remotely. Although security mechanisms exist, they are rarely portable to the II. A reason is that these infrastructures are extremely complex and heterogeneous, made of myriads of objects using very different hardware and software. Such an extreme diversity, together with the quick development of the modern infrastructures, require appropriate countermeasures to ensure the security of the next-generation II.

Security requirements cannot be put aside when designing and building a II. As a matter of fact, when new infrastructure is designed without security in mind, it can be hardly secured later on. For this reason, a *security-by-design* methodology is needed so that security requirements can be evaluated at each stage of the lifecycle of II. The goal is to obtain overall security guarantees in line with the state-of-the art security mechanisms which, often, are difficult to be applied to technologies such as the IoT. Providing practical tools to support the security-by-design approach is the objective of the HAII-T Program.

This document defines the roadmap of the five Tasks contributing to the HAII-T Program as well as the roadmap that will drive the integration process. We start in Chapter 2 by discussing the state-of-the-art of the defense mechanisms for the legacy components. Then, Chapter 3 will focus on security of the operating systems and software for the IoT and field devices. In Chapter 4 we will discuss the technologies for the orchestration of complex, secure-by-design infrastructures. Chapter 5 and Chapter 6 will deal with two crucial aspects of the secure design of II, i.e., *resilience-by-design* and *privacy-by-design*. The integration roadmap is presented in Chapter 7 where we will discuss the use cases of the Program as well as the integration challenges and milestones. Finally, for each of the aforementioned chapters, we will draw the concluding remarks in Chapter 8

Chapter 2 Hardening Legacy Components

Defense mechanisms on different abstraction levels became essential, especially in the era of mobile, IoT, Industrial Cyber-Physical System (CPS), and Industrial Control System (ICS) devices. There exist a wide range of attacks aiming at various vectors of such systems endangering both trust of delivered information and safety. As such, these systems are expected to withstand state-of-the-art attacks, while, at the same time, they are build on top of (potentially less powerful) legacy systems with limited architectural capabilities. Due to this limitation, as part of the task 6.2, we focus on hardening legacy devices, ranging from microcontrollers to legacy x86-based systems. Specifically, we focus on establishing defense mechanisms against different attacks including those resulting from memory vulnerabilities (in particular, code-oriented attacks and data-oriented attacks) and side channels.

2.1 Analysis of Existing Defense Mechanisms

In order to establish a strong defense, first, we analyze existing defense mechanisms targeting memory corruption (in particular *control data* and *non-control data* attacks) and fault injection attacks.

2.1.1 Defenses against Memory Corruption Vulnerabilities

Even if advanced programming languages are growing, most lines of software code are still written in C and C++, since these languages permit low-level control without losing the advantages of high-level statements. Although, the direct management of data structures and memory pointers opens the door to a wide range of vulnerabilities. The lack of memory safety capabilities (such as a strong typization, present in other modern languages) enables attackers to exploit these flaws by maliciously altering the program's behavior or even taking full control over the control-flow.

In C and C++, memory pointers are normal variables, that instead of containing data, contain addresses to memory. As normal variables, they can be freely manipulated by the program which declares them: initialized to a random value, incremented, decremented, reassigned, and so on. This may cause serious problems, as programming errors may lead a pointer to point literally everywhere in the addressing space, and to overwrite data outside the range it was initially thought for.

An interesting paper of Szekeres *et al.* [345] tried to give a systematization of memory corruption problems. In the authors' opinion, the corruption of a memory pointer always follows 2 main steps: first, a pointer is made *invalid* by going out the bounds of its pointed objects or by being deallocated. At this point, the pointer has to be *dereferenced* to trigger the error. When an out-of-bound pointer is dereferenced, a *spatial* memory error occurs; when a deallocated pointer is dereferenced, a *temporal* memory error occurs.

The probably most famous vulnerability of this kind is *buffer overflow*, which is caused by incrementing or decrementing a pointer without proper boundary checking on the data structure that is being accessed, with resulting out-of-bounds writes which corrupts adjacent data on stack, heap or other zones. Similar problems may rise when *indexing bugs* are present in the code, i.e., boundary checks over an index for a given data structure are missing or incomplete. Indexing bugs are often caused by integer-related errors like an *integer overflow*, truncation or signedness bugs, or incorrect pointer casting.

Temporal memory errors are instead often referred to as *use-after-free* vulnerabilities. Here, a pointer is dereferenced (used) after the memory area it points to has been returned (freed) to the memory management system. After the free, the pointer still points to the deallocated region, which in the meanwhile can have been written with other data. The consequence is that newly allocated data on the heap may be corrupted by accessing erroneously to the memory using these *dangling* pointers.

Code-Oriented attacks are best described by the ability to redirect the execution flow of a system to executable regions where the attacker can reuse individual instructions or whole functions to craft a malicious execution path. That is, given a memory corruption vulnerability, the attacker is able to change variables that dictate the execution path when triggered, such as return addresses or function pointers. Modifying the return address can be achieved via buffer overflows into the stack layout, or format string vulnerabilities, where the return address of the caller is stored, so that, the attacker is able to overwrite the benign return address with a malicious destination. Other sensitive control flow dictators are function pointers. Overwriting function pointers has attractive use cases for a malicious attacker in heap structures, where the User- or kernel space allocators intermix objects with different functionality in contiguous segments of the memory. Therefore, heap overflows or corruptions of free objects metadata allow an attacker to overwrite into different object adjacent to the vulnerable object, and control function pointers to craft their malicious execution chain, when invoked. Both

techniques require the attacker to trigger a memory address leak, which can be used to compute the addresses of the malicious gadgets. Recent years have offered plenty of research against Code-Oriented attacks, which make such attacks harder or impossible in certain states of the system. Techniques such as Data Execution Prevention (DEP), stack canaries, ASLR, Position Independent Executable (PIE), Control Flow Integrity (CFI), Return Address Guard, and others, have been proposed which mitigate Code-Oriented attacks with their own contributions and trade-offs. ALSR and PIE require the attacker to obtain a memory address leak, otherwise the address layout is random and hard to guess, which masks the vulnerability and cloaks the exploit payload.

However, less attention has been accorded to a new class of exploits called Data-Oriented attacks. Compared to Code-Oriented attacks, this novel strategy does not leverage redirecting the execution flow of the system, but rather corrupting selected memory segments that store sensitive data which an attacker can use for malicious purposes. Assuming a system equipped with various hardening techniques that mitigate Code-Oriented Attacks (such as the ones mentioned above), memory corruption vulnerabilities can arm an attacker with arbitrary read and write capabilities that must not necessarily be used for redirecting the execution flow to gain full control of the system. Instead, the attacker can leverage the capability to control arbitrary sensitive data structures which hold metadata/identity information about the executing process/task, such as their user permissions, privileges, capabilities or private information such as encryption keys or identity certificates. Even more, an attacker can consider altering the structures that describe the address space of the process, and inject their own address space with malicious functionality.

Causes of Arbitrary Reads and Writes

One of the most attractive sub-system hunted for vulnerabilities that can lead to arbitrary reads and writes is the heap memory allocator (aka the dynamic memory allocator), both in the User- and kernel space. Even though the two execution modes provide two different types of allocators with different memory layout for the chunk management, they share a common problematic design decision which makes the allocators prone to critical mistakes. The risk is introduced by (1) the fact that the allocators allocate objects in regions of memory adjacent to each other and (2) the memory resue optimization to keep metadata of freed objects within the previously allocated object. The management data of freed objects is represented by pointers to the next and/or previous freed objects, so that the allocator can server allocation requests in constant time and adjust the freed object chain rapidly. An attacker that manages to corrupt any of these pointers in their freelists, can trick the allocator into returning an address that points to a memory region of the attacker's desire. This technique represents a solid and popular way of obtaining arbitrary read and write capabilities that form the grounds for Data-Oriented attacks. Another method for obtaining arbitrary control of memory regions is through format string vulnerabilities, where the attacker is able to control the string passed to format-aware procedures, which implement esoteric features, such as writing at the address stored by one of its parameters. We will not dive deeper into these class of bugs, since they can be easily patched by compiler checks, annotations, and so on. We are focusing on the vulnerabilities induced by mistakes in handling heap objects. The bugs that lead to such vulnerabilities are explained in the following. They apply to both the User- and the kernel space.

Throughout this document, we consider a system that runs a Linux kernel distribution, and the vulnerabilities described are presented in the context of the user space glibc allocator and the kernel space SLAB allocator (the SLUB implementation).

Heap Overflows. Because of the layout of the memory allocators, where objects (either freed or allocated) are adjacent to each other, an overflow caused by an allocated object into the next object can lead to corrupting the freed object metadata. Moreover, in the case of the user space glibc allocator, overwriting one byte into the next object, be it freed or allocated, is enough to trigger a malicious sequence of allocations, modifications and frees to mislead the allocator into returning an arbitrary address to the attacker. In the first case, when the overflowed object is allocated, the attacker can overwrite its size field, which on further freeing of the victim object, results in merging the freed object with the previous one, which is in an allocated state. This leads to a Use-After-Free vulnerability. In the second case, when the overflowed object is freed, the attacker can control the next and previous pointers to whatever they desire, such that further allocations will return those addresses to the attacker. In the kernel space, a one-byte overflow suffices as well, but an attacker must overflow into a freed object, such that they alter the next free pointer field, which is the first field of the freed object. Once more, they can control the address of the next free pointer to whatever address they desire, so that the allocator returns, at a later point, the selected address. In both cases, for the User- and kernel space scenarios, a one byte overflow suffices in most cases, which means that arbitrary overflows are even easier for an attacker to exploit.

Use-After-Free (UAF). Both the User- and kernel space allocators store management metadata of the freed objects in the object itself, but with additional tags to mark the object as free. Therefore, what used to be user data, is now management data used by the allocator for objects. A critical mistake that lead to UAF attacks is the fact that one or more references of the allocated chunk are replicated by the user, and, after freeing, those

references are not destroyed and they still point to the location of the freed object in memory. Therefore, an attacker that still has references to those objects, can use the object as if it was allocated, and overwrite the free management pointers to obtain an arbitrary pointer to sensitive memory on further allocation requests.

Double-Free. The Double-Free bug leads to the Use-After-Free scenario described above, and it consists of wrongfully freeing an object twice. Both the User- and kernel space allocators use freelists to chain freed objects that allow for fast delivery of allocation requests. Therefore, if an object is freed twice, it is added in the freelist twice, so that the next allocation requests will return the same address twice. Since the attacker has two references to the same object, they can choose to call free on one reference, while keeping the second reference to alter the free management information that was just added because of the first free request. This leads to a UAF bug, allowing the attacker to control the next and/or prev freed objects pointers.

Invalid free. An attacker that is able to control the address that is passed on to the (k)free function has the ability to craft a fake chunk in a memory region that they control, and instruct the allocator to free that chunk as if it were previously allocated. When freeing, the allocator configures the pointers to the next and/or prev freed objects, which reduces this scenario to a UAF case, since the attacker still has the reference to the fake chunk, and is able to overwrite those free pointers.

Over/Underflowing Ref Counters. There exist data structures that are shared among entities, such as threads, processes, functions during the runtime of a system, which have associated with them a Reference Counter, in order to track how many entities have a reference to the objects. This is needed to determine when to completely destroy the object, so that no other entity still has a reference to the object. The reference counters are fields stored in the data structure increased on acquiring the reference to the object, and decreased when the user releases the object. When the reference object reaches the value of 0, the object is destroyed, which consists of possibly calling the (k)free function, since the object was highly likely a heap chunk. If the users can abuse the number of entities that are allowed to hold a reference to the object, they can keep requesting a reference to that object, so that the object manager keeps increasing the object's reference counter, until it overflows and it reaches the value of 0, and then 1, even though plethora of references are still held by user throughout the system. When the attacker requests to release the object, with a reference counter of 1, the value is decreased to 0, the object is therefore freed, and the attacker still has references to those object. Underflowing presents a similar strategy, but instead of requesting references to the object, the attacker repeatedly releases the object so that its reference counter underflows from 0, -1, -2, and eventually 1, which mirrors the previous case. Both lead to UAF scenarios, since the attacker is able to control the freed pointers of the reference counted object.

In the presence of a vulnerable user space process or a kernel space task, an attacker that has obtained arbitrary read and write capabilities can leverage the following strategies to corrupt the memory of the system, which can lead to various critical consequences:

Data Leakage

General Description: Once an attacker has obtained arbitrary read, they can inspect the whole address space of the vulnerable process or task, being able to read arbitrary memory locations. If a sensitive application that works with private user information, such as cryptographic keys or passwords, displays an arbitrary read vulnerability, the critical data that is kept in memory is at risk of being exfiltrated by the attacker. When met, these attacks can compromise the entire security provided by the protocol which the vulnerable application implements. Also, user accounts can be unlocked, therefore, breaking the privacy of the affected users. Moreover, in the presence of an arbitrary read vulnerability, an attacker is able to leak address space gadgets from the vulnerable process or task, which allows them to proceed with the attacks described below and successfully compromise the system. Without these leaked memory gadgets, the attacks described below would become impossible or tedious.

Practical Application: The notorious Heartbleed bug, found in the OpenSSL cryptographic software library, displays a memory corruption vulnerability that allows attackers to steal private secret keys kept in the library's memory space, from a remote machine located anywhere in the Internet. With private keys in possession, attackers were able to intercept securely-encrypted traffic, and eavesdrop on sensitive conversations, documents or other data, therefore, breaking the security of the whole Internet. The attack is possible due to an buffer over read that allows a remote attacker to send an arbitrary amount of crafted packets to the vulnerable system/server and retrieve blocks of 64KB of memory from the process' memory data segments.

For the next Data Oriented exploitation strategies we present a common Practical Application based on the `struct cred` that is described after the General Description of each strategy.

Data Modification

General Description: By employing a data modification strategy, an attacker is able to modify the content of a sensitive data structure located in the memory of the process or task, given a memory address leak that allows them to learn the exact location of the data structure in the memory. Modifying fields in sensitive data structures can enforce the attacker with privilege escalation, use-after-free bugs, address space injection or denial of service against the victim system.

Data Masquerading

General Description: Data Masquerading assumes the system contains sensitive data structures, allocated for privileged tasks (`struct cred init_cred`) or private to individual processes (structures that private encryption keys). An attacker with arbitrary write capabilities can overwrite the pointer to the their equivalent data structure with the pointer of the privileged/private data structure, given the address leaks of the victim data structure and the address leak of the container data structure which stores the actual pointer. Such strategies can lead to privilege escalation, address space injection, denial of service.

Data Forging

General Description: An attacker can craft its own data structure in the user space or in the kernel space where he can control the values of the sensitive fields. In the presence of the aforementioned capabilities, arbitrary write, the attacker can overwrite the pointer to its equivalent data structure with a pointer to the crafted data structure, given an address leak that reveals the base data structure which holds the pointer to the sensitive data structure. This strategy can lead to privilege escalation, address space injection, denial of service against the target system.

Practical Application: In the following, we exemplify the attack strategies presented above on the sensitive data structure *`struct cred`*, which contains fields that describe the permissions and capabilities of a user and its processes in a Linux kernel system. The data structure `struct cred` is intensively used in many permission-related security checks before accesses on various data structures, such as sockets, files, pipes, shared memory, other tasks and so on. For example, `struct cred` is used by the Linux kernel when a task wants to open a file on the filesystem to check weather the user that owns the task has the permissions to access the file. Therefore, being able to control the `struct cred` of a task, a malicious user can bypass the checks made by the kernel by impersonating privileged `struct cred`, which allows them to access restricted files that contain sensitive information, such as cryptographic private keys or passwords. `struct cred` contains identification fields that are used by the kernel to implement its Discretionary Access Control (Unix/Posix Access Control Lists) and Mandatory Access Control (extended security policies per oject) security checks when a task wants to access a resource. The most attractive fields are the UID (User ID), EUID (Effective User ID) and the SUID (Saved User ID) fields, which are critical for determining if the process is allowed to access the resource (file, socket, etc.) or not. A task that has an instance of `struct cred` with these fields set to 0 has the most elevated privileges and can access any resources at any time. We call such an instance of `struct cred` as the init_creds, which is configured by the kernel at boot time, and represents the credentials of the init process and tasks that belong to the root user, being the most privileged in the system. Therefore, the interest of the attacker is to transform its task's `struct cred` into instances of init_creds, such that, they can bypass any security checks and access any resource in the system. They can acheive this by employing the strategies presented above. Before diving into the attacks, it is worth mentioning that the current executing task is described by a kernel structure called 'struct `task_struct`', which also contains a pointer to the `struct cred` as credentials of the executing task. Therefore, in order to be able to manipulate its `struct cred`, the attacker must obtain (in most cases) an address leakage of the `struct task_struct`, so that they know where in memory the pointer to its `struct cred` is. In order leverage a Data Modification scenario, the attacker must learn, besides the address of its 'struct task_struct', the address of its `struct cred`. Nevertheless, under the assumption that the attacker has arbitrary read, leaking the address of the `struct cred` is not a concern, and can be acheived in the same way as 'struct task_struct' was leaked. With the memory address of the `struct cred` in hand, the attacker can target the UID, EUID and SUID fields in the structure to clear their values to 0, therefore, escalating their privileges. When manipulating their own `struct cred` is not possible, the attacker can opt for Data Masquerading. For this attack, the attacker must learn the address of the init_creds, which are the most elevated credentials in the system. Also, they need to learn the address of the `task_struct` that describes the malicious task. Once they have these two values, the attacker can overwrite the pointer to its credentials with the pointer to the privileged credentials, therefore, escalating their privileges by impersonating the root credentials. In case the address of the `init_creds` is protected from the attacker, but they still have the address of their `task_struct`, they can still escalate their privileges. They can achieve this by crafting a fake `struct cred` in a region of memory they control, with

all critical fields, such as `UID`, `EUID`, `SUID`, set to the most privileged values. With the address of their task struct in posession, they can overwrite the pointer to their original `struct cred` with the pointer to the fake privileged creds structure, therefore, elevating their privileges. This attack might become more difficult in case the attacker plans to craft the fake privileged credentials in the User Space memory given the presence of the Kernel Page Table Isolation enhancement, which forbids the kernel to access User Space memory, without specifically requesting it. They would have to find a region in the memory of the kernel that they control, or a chunk of memory that accidentally makes up a valid and privileged `struct cred`.

2.1.1.1 Defenses against Code-oriented Attacks

Memory vulnerabilities described above may enable attackers to maliciously take control over the program by forcing it to execute an unintended sequence of instructions. This exploit is generally called *Arbitrary Code Execution (ACE)*. ACE is achieved through tampering with the *instruction pointer*, in most of the architectures referred to as *Program Counter (PC)*, of a running program. The PC points to the next instruction to be executed, therefore if an attacker controls its value, he also can decide the instruction to be executed next.

In ACE, the PC register value is corrupted to point to the attacker's *payload*. Depending on the nature of this payload, 2 types of ACE attacks can be distinguished:

1. The payload is injected together with the corrupted instruction pointer in the memory of the program (*Code-Injection Attacks, CIA*);
2. The payload is composed of snippets of code already present in the memory of the program, but not intended to be executed in that order (*Code-Reuse Attacks, CRA*).

The instruction pointer is corrupted through tampering with the operand of an instruction that copies it from memory into the PC (*indirect control-flow transfer instructions*). The RETURN instruction, or indirect formats of CALL and JUMP, are example of such instructions, but also any other instruction that treats the PC register as a destination for a computing operation. For the purposes of an ACE attack, *direct control-flow transfer instructions* are instead useless, since their argument is an immediate value that is used as an offset to be added/subtracted to the current value of the PC, and therefore cannot be modified.

CIA exploits were made practically impossible after the wide adoption of main architectural countermeasures like:

- *Data Execution Prevention* (DEP) [343]: memory regions allocated for data are marked with a NX flag (no-execute) and cannot be loaded as instructions by the processor;
- *Write XOR Execute* [349]: no memory location can be both writable (W) and executable (X) at runtime.

On the contrary, CRA is a very strong threat still nowadays. In a paper of 2007 by Shacham *et al.* [328], the authors first theorized that "*in any sufficiently large body of executable code there will exist sufficiently many useful code sequences that an attacker who controls the stack will be able [...] to cause the exploited program to undertake arbitrary computation*". The control flow can be diverted to execute a series of small sequences of instructions, each ending with an indirect control-flow transfer instruction, known as *gadgets*. In a large enough codebase (such as libc, compiled for every application written in C), there is a massive selection of gadgets to choose from, and the attackers achieve the maximum of expressiveness. On the x86 platform, the attack is made stronger by the fact that, since there is no fixed instruction length, any sequence of raw bytes can be interpreted as an instruction, and the rogue return address can point even in the middle of an opcode, transforming it into another.

The most famous attack paradigm belonging to CRA is *Return-Oriented Programming (ROP)*. In ROP, the attackers write their malicious code using, instead of instructions, the gadgets found in the code of the system to be attacked as basic "bricks". These gadgets may perform any kind of general-purpose action, as copying values from registers to others, loading values from memory, or doing arithmetics and logics. The common property they must have is that they have to end with a RETURN instruction. Once individuated the set of gadgets, attackers fill the stack with a list of fake return addresses exploiting a memory vulnerability. Each of the injected addresses points to the beginning of each of the identified gadgets. Between addresses, attackers may want also to place immediate data for their execution.

The attack starts when the function that contains the vulnerability returns: by executing the RETURN, the processor copies the first corrupted value into the PC, and the program flow is redirected to the first of the gadgets. Once the first gadget is finished, another RETURN is executed, that carries the flow to the second gadget, then to the third and so on.

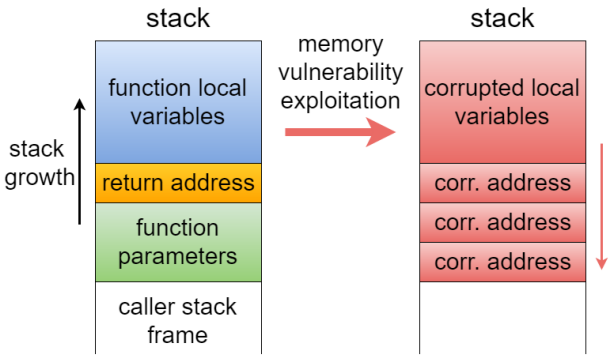


Figure 2.1: Return address corruption, start point of ROP attacks.

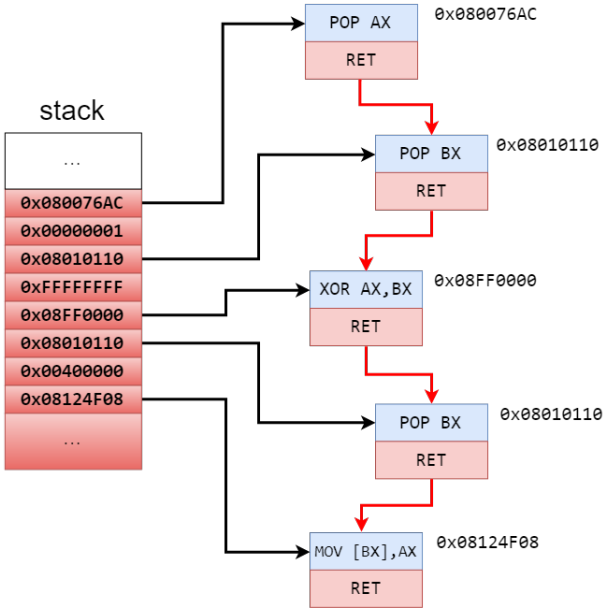


Figure 2.2: A basic example of ROP attack.

In the last years, research community and companies started elaborating and adopting different types of solution to counter CRAs.

Address Space Layout Randomisation (ASLR). It is a countermeasure taken at link-time which randomises the memory layout of the application, making it harder for an attacker to know the exact addresses of library code, where to retrieve gadgets [50]. Actually, in 32-bit architecture the introduced entropy is too low, and brute-force attacks can easily break the defense. Furthermore, since only the base address of each segment is randomised, it suffers of information disclosure: gaining the knowledge of a single address leads to compute the library segment base address in a straightforward manner.

Stack Canary (or Cookie). When a function is called, an additional word with a known value can be pushed on top of the stack, which is placed between the return address and the local variables. When the function returns, the value of the canary is checked, and, if it is found changed, the program is considered under attack and terminated [109]. The canary can have a random value difficult to guess or can be composed of terminator characters, making it difficult to manipulate using input function (such as gets()), since terminator character breaks the input streams when recognised. However, canaries have been shown to be circumventable with more targeted stack-smashing attacks [13].

Shadow Call Stack (SCS). Basically, at call-time, the return address is both saved on top of the normal stack and on top of an additional shadow stack, accessible only by the processor architecture and not visible to the programmer. At return time, the instruction pointer is popped from both stacks, and the values are compared. If they are not equal, a violation fault is triggered [360][68]. This solution blocks any kind of stack-smashing attacks, but it is not sufficient to fully protect an application, as it does not address memory vulnerabilities present in other segments (heap, bss, data, etc), enabling attackers to still rely on other techniques such as *Jump-Oriented Programming (JOP)* [57][91] or *Call-Oriented Programming (COP)* [314].

Heuristic-based Approaches. They claim to detect CRAs by typically monitoring the number of branches of the program and block it when suspicious behaviour is sensed. This defense are based on *assumptions*, e.g., gadgets for ROP and JOP attacks usually consist of no more than 5 instructions. DROP [90], kBouncer [274] and ROPecker [98] are examples of heuristic-based defenders. However, it has been demonstrated that the heuristic can be easily thwarted by executing, between malicious jumps, longer sequences of non-jumping instructions or branches considered as secure [163].

Control-Flow Integrity (CFI). Solutions presented so far can still be valid mitigation techniques, relatively simple to implement, but each of them addresses the problem of CRA with respect to vulnerabilities only, without an all-encompassing vision. A paper of 2005 by Abadi *et al.* [3] first tried to change perspective by introducing the concept of Control-Flow Integrity as basic defense against CRAs, regardless of the vulnerability that may cause them. The concept behind CFI is monitoring the program at runtime to detect abnormal diversion from what is stated in its *Control-Flow Graph*. Each node in the CFG represents a *basic block*, which is a group of non-jumping instructions executed sequentially. Edges represent *branches* in the control flow, caused by JUMP, CALL or RETURN instructions. The CFG is defined before the execution, through a static analysis of the source code/binary code, or by *execution profiling*, a test run which creates the possible execution paths. Then, at runtime, the possible control-flow transfers are restricted to the static CFG.

CFI policies are clustered into *coarse-grained* if the monitoring is not done by strictly enforcing the CFG, but based on simple program flow rules, such as ensuring that instructions targeted by a RETURN are preceded by a CALL, or that indirect CALL formats only target prologues of functions. *Fine-grained* policies, instead, check that the execution traverses valid edges of the pre-computed CFG *only*.

Coarse-grained solutions result in being not so different from heuristics, as both aim at distinguishing the rogue behaviours from those that *most probably* are benevolent. But ”most probably” does not mean ”certainly”, especially when we are dealing with clever attackers. Recent works [120][171] showed how it is possible to induce such security policies to believe that actions are within the rules when they are not. In [82], the authors showed that just 70 KB of binary code retrieved in 10 different executables within /usr/bin of Linux have been sufficient for mounting fully CALL-preceded ROP attacks.

The most common fine-grained CFI technique is the one originally introduced in Abadi’s paper, based on *label instrumentation*. It relies on inserting unique label IDs at the beginning of each basic block. Before each indirect branch, a the destination basic block’s label ID is compared to a label ID which is stored inside the program. Since unique label IDs are used, control flow tampering causes the check to fail, since the destination label ID will not match the label ID stored inside the program. The control flow checks are performed using code checks which are inserted at the end of each basic block containing an indirect branch. This technique guarantees very high protection, although it often incurs in performance overhead for checks, which in many cases (e.g., real-time systems) can be unaffordable.

Another defense based on code instrumentation is *Control-Flow Locking* (CFL) [56], which consists in inserting *lock* code before indirect transfer instructions and *unlock* code at each of their valid target. The lock code sets a lock variable to a value, while the unlock code, before proceeding with the execution, verifies whether the value is the lock one. The lock code also verifies if the just-executed code was unlocked and thus allowed to run, otherwise it notifies a violation. CFL suffers of lack of isolation with respect to the security variables, e.g., the lock variable, which can be corrupted with more advanced attacks that break even strong memory protection mechanisms.

Hardware-based CFI solutions try to overcome overhead and isolation limits typical of pure-software solutions. In fact, the program runs normally and the CFI checks are performed much faster and almost transparently. Furthermore, the data structures containing CFG critical information belong to the monitor exclusively, without the possibility of accessing by the main execution.

Literature has indulged in this theme. In [229], it is proposed the encryption at load-time of the indirect branch target instructions, and the addition of a processor module that decrypts on-the-fly these addresses before loading into the instruction register. The execution obviously crashes if a badly-decrypted instruction is run, and to mount a redirection attack the attacker must know the encryption key (extracted from a processor PUF, so generated everytime and never stored). In [292], the authors propose a similar method which involves encrypting with a lightweight version of AES the return addresses at call time before pushing them, and decrypting them at return time. Others propose to solve the question at a higher level of abstraction, e.g., marking code memory pointers as compile-time-generated or run-time-generated [94], encrypting and decrypting them on-the-fly [223], or using special instructions for their load and store and placing them in a different special stack to isolate them from buffer overflow vulnerabilities [155][302].

Others [10][67][123] have proposed the insertion of a call shadow stack into the architecture, implementing their solution on the RISC-V soft processor. In [123], the monitor is also equipped with a table of the destinations allowed for each indirect transfer, and gets the status of the monitored program through a parallel interface with the main core which basically carries out the instruction register on a bus.

Another technique relates calculating the allowed sequence of basic blocks before the execution and then verifying it at runtime by continuously computing the hash of the blocks. Authors in [377] propose as an hardware trusted module parallel to the main processor that does this by reading the program counter and instruction register. The same is proposed in [127] by inserting cheking modules directly connected to the pipeline stages. [119] and [88] propose to install validation modules between the processor and the instruction cache to sniff the instruction flow.

Possible modifications or security extensions have also been studied for the branch buffering and prediction modules present in most processors [331][380][222].

Others have offered solutions that go in the direction of modifying the ISA (Instruction Set Architecture) of the machine to directly give the programmer the possibility of inserting CFI-dedicated instructions in the program to be protected. The processor is therefore equipped with internal data structures as label stacks to protect backward edges and label registers to protect forward edges, and the instruction set is augmented with the opcodes necessary to manage them. Examples are the works presented in [121][102][341], where changes were made to the original design of some SPARC soft processors.

2.1.1.2 Defenses against Data-oriented Attacks

Contrary to the plethora of defense mechanisms targeting *control data* oriented attacks, defense mechanisms against *non-control data* (i.e., data-oriented attacks) are yet to become popular. Data-oriented attacks [186, 192, 342] avoid changing the control-flow. Instead, they focus on modifying non-control data to subvert the targeted application. In the following, we discuss four state-of-the art defenses against data-oriented attacks, namely *SeCage* [235], *MemSentry* [209], *PrivWatcher* [93], and *PT-Rand* [122].

SeCage

SeCage[235] focuses on reducing the data disclosure damage provided by memory corruption vulnerabilities, by compartmentalizing the sensitive software's critical code and data, which is separated from the trivial code and data that don't represent a danger for the secrets of the application. They isolate these components so that, secrets and critical data are only accessible to critical code. In order to achieve this isolation SeCage leverages virtualization techniques such as Extended Page Table (EPT) memory views and `VMFUNC` instructions. They define compartmentalization candidates by using static and dynamic analysis on the target application, together with programmer-expected flagging (labeling) of secrets, to split the application into a main component and a set of secret components. They guarantee that the secrets of one compartment are only accessible to code contained in that same compartment, by configuring at inception time an isolated memory view per compartment. Therefore, each compartment is described by a separate EPT. The transition from the main comportment into one of the secret compartments is done via trampoline segments of code, which jump via the `VMFUNC` instruction into a different memory view. SeCage allows two types of trampoline codes, either from the main compartment to a secret compartment, or the other way around. In the secret component they have all the data segments mapped, including secret and non-secret data, while they only map critical code, and keep the trivial code un-mapped. This is why, the Trusted Computed Base of SeCage is relatively small, and they can claim no vulnerabilities in the secret compartments. Also, they keep interrupts disabled while the CPU runs in the secret compartment, so that, a potential infected kernel, can't leverage interrupts to interfere with the secret views. At the rug of the gates of a secret compartment, the underlying hypervisor checks the defined policies to determine if the piece of code trying to take the trampoline is authorized to do so or not. Specifically, the hypervisor checks if the function requesting the ticket to the secret compartment is present in the set of functions curated by the static analysis. They enforce this check by mapping the equivalent data pages of the secret data to non-present, which triggers a VMEXIT just before the trampoline gets executed, because the software tries to access an EPT-unmapped data page. This redirects the execution flow to the hypervisor which checks the policies. SeCage assumes a strong adversary model that could control both the user-space and the kernel-space. Secret data such as cryptographic keys or password, stored by user-space applications (such as OpenSSL, CryptoLoop, and so on) are robust against an attacker that obtained full control of the software, while the application is protected by SeCage. This is due to the fact that any malicious attempt to bypass their security mechanisms (such as a counterfeit `VMFUNC` instructions) is checked via the hypervisor intervention. However, this is one major drawback of SeCage, which acts as a monitor at load and runtime of applications, that constantly need to jump the trampoline. Since the hypervisor must check a list of statically analyzed functions before each secret code call, doesn't scale to large systems that present a large code base with many functions. Moreover, static analysis can overestimate the number of secret compartments, which increases the Trusted Computed Base of the system. which increases the chances to encounter vulnerabilities. Also, static analysis can be imprecise to miss functions that define SeCage's policies in the hypervisor.

MemSentry

Since recent research developments have motivated CPU vendors to develop hardware extensions for isolating a system into multiple security domains, MemSentry [209] is proposing a framework that combines such novel features to implement memory isolation on x86-64 systems. The most notorious hardware extensions that contribute to the framework are Intel's Memory Protection Extensions (MPX), Memory Protection Keys (MPK) and `VMFUNC` instructions. They differentiate between two memory isolation techniques into deterministic and probabilistic protection, and classify their approach as deterministic memory protection. Deterministic memory protection tends to be more robust in the context on domain-based isolation, since the isolation method guarantees that the isolated part cannot access anything outside the isolation border, and vice-versa. On the other hand, probabilistic isolation emulates the deterministic isolation by information hiding of sensitive segments in memory, but which is vulnerable to memory disclosures. To further modularize deterministic isolation, MemSentry defines two types of memory compartmentalization: address-based and domain-based isolation. Address-based isolation splits the address space into partitions and memory accesses are configured to only access the allowed partitions. Domain-based isolation defines areas in the address space that are completely inaccessible from the outside, but they can be toggled on or off, allowing temporary access to authorized entities. First, they present a domain-based isolation implementation option using Intel's EPT and the capability to switch between them via `VMFUNC` instructions. Specifically, they define a nonsensitive domain and a set of sensitive domains, which are mapped one-to-one on a set of EPT memory views. By default, the nonsensitive domain is active and only special authorized instrumentation points are allowed to switch to the sensitive domains via the `VMFUNC` instruction, which doesn't require hypervisor intervention. Second, MemSentry presents Intel's MPK as an alternative for domain-based memory isolation. They configure the virtual addresses of sensitive domain with Protection Keys that represent restrictive privileged access in the special registers used to store access permissions per protection key. However, at the time of the writing, processors were not equipped with MPK functionality, so MemSentry emulates the feature. Moreover, MPK does not provide configuring kernel virtual addresses (since all of the higher bits of the kernel VAs are '1'), and the feature is limited to only 16 isolation domains. MemSentry also implements the second isolation alternative, that comprises into splitting the address space into memory partitions which forces a sub-set of all loads and stores to access only certain partitions. They achieve this using the Intel MPX feature allows the application developer to specify and enforce bounds for a range bounded by two 64-bit addresses, which are checked against a certain value by specialized instructions. Their approach is to map isolation domains into memory ranges, by specifying a set of upper and lower bounds to delimit the memory isolation domain. However, the hardware feature only supports 4 bound-values register, while additional bounds are kept in memory. This limits the method to only two isolation domains, since a larger number would require their bounds to be stored into memory, making the vulnerable to memory corruption vulnerabilities.

PrivWatcher

PrivWatcher [93] is a data-oriented attack mitigation framework which focuses on isolating process credentials in the Linux kernel. Their framework provides both integrity verification and protection of process credentials in the presence of memory corruption vulnerabilities, which blocks the attacker's attempt to escalate their privileges that potentially leverage one of the malicious strategies presented in the previous sections. One of the pre-requisites of PrivWatcher is that they can execute in an execution domain which is more privileged than the kernel's, for example, at the hypervisor level. PrivWatcher's approach to achieve protection of process credentials consists of three components. First, they propose separating the allocation of process credentials, so that sensitive non-control fields are not intermixed with unprotected data fields. Second, they implement a dual reference monitor that satisfies the Time of Check to Time of Use (TOCTOU) policy, which guarantees that the checks made when a modification of the process credentials occurs, are performed in the same execution context as when the sensitive fields are used. Third, they define a set of heuristics to provide data integrity verification, which aims to detect malicious kernel modifications of the process credentials. At a high level, they define a Read Only Safe Region, non-writable to the kernel, where credentials are isolated so that none of the attacks described above work. In order to update or modify the process credentials, the kernel needs to make an explicit request to the PrivWatcher mediator, which processes the request based on the kernel access control policies. In order to implement the first component, PrivWatcher doesn't interfere with the entity in charge of the process credentials memory chunks, namely the kernel dynamic allocator, but rather intervenes when a reference to the cred structure is assigned to a pointer, by creating a duplicate in the safe region and freeing the original un-protected structure. This component bypasses the memory overwrite scenario. To tackle the other attacking cases, PrivWatcher provides techniques to enforce the TOCTOU policy. Namely, they provide a one-to-one association between the `cred` and the data structure that defines the context of currently executing process, i.e. the task struct. They do so, by extending the `cred` with two additional pointers: a forward reference to the protected copy from the safe region and a backward reference to the `task_struct`.

Whenever the privileges of a process are set, or when the kernel performs ssecurity checks for accessing sensiutive resources, PrivWatcher make sure that the following conditions are met: the process credentials belong to the safe region and that the back reference and the CR3 address space value match the PGD address in the `task_struct` match. The third component of PrivWatcher is satisfied by not non-root processes that execute non-suid binaries to escalate their privileges. The threat model that PrivWatcher targets, is, unfortunately, solely focused on the process credentials referenced in the `task_struct`. Their solution is not generic to arbitrary memory regions, which leaves a system equipped with PrivWatcher, vulnerable to other attacks vectors, such as modifications in other objects from the kernel heap or modifications in the address space (paging tables) of the malicious process.

PT-Rand

To tackle the limitations of PrivWatcher, which is failing at protecting the paging tables of a process for modifications, PT-Rand [122] comes with a promising technique that provides address space robustness against malicious overwrites in the paging structures, which enables an attacker to perform address space injection attacks in the presence of a memory corruption vulnerability, Their approach relies on a hypervisor-less system that protects paging structure with minimally induced performance overhead. Address space injection is a critical attack that an attacker could use in a vulnerable kernel, since it allows them to twist the memory management functionality to replace benign code pages with malicious code pages. Thus, even in the presence of fine-grained CFI mechanisms, the attacker is able to obtain malicious code execution from the user space. The high level design of PT-Rand consists of two techniques: randomizing the location of the page tables in memory, based on a randomization secret, and obfuscating the virtual addresses of pointers that reference the paging structures, by replacing them with their equivalent physical addresses, which would reduce the damage of a data disclosure vulnerability. They assume an attacker that employs arbitrary read and write capabilities, while code injection is mitigated by CFI mechanisms. In order to achieve their security goals, PT-Rand allocates the kernel paging tables at boot time, based on a randomization secret generated from hardware entropy generators, and map these paging tables at an unpredictable virtual-to-physical offset computed based on the random token. They patch the paging table structures allocator wrappers to return the address of the physical page whenever a paging structure is requested, so that, in the presence of a leak, without the randomization secret, the attacker is not able to inherit the virtual address of the page. To isolate the randomization secret, PT-Rand uses a privileged register on the CPU, which is neither accessed by the kernel in its original codebase nor accessible to user space processes. Therefore, under the assumption that the attacker doesn't have arbitrary code execution capabilities, the privileged register is only accessible to dedicated kernel regions. While their approach proves high robustness against guessing attacks, in an attempt from the attacker to brute-force the location of the randomized paging tables, the PT-Rand mechanism is critically dependent on the randomization token stored in the privileged register. They rely on the fact that they completely eliminate the possibility of register spilling, by patching the compiler and checking the codebase of the Linux kernel to never access the register in cause. Moreover, similar to PrivWatcher, their approach is subjective to a particular data structure, the paging tables, without further extensions on arbitrary memory isolation.

2.1.2 Defenses against Fault Injections

An evaluation of the already existing defenses mechanisms against fault injection attacks requires an accurate analysis of the threat. This threat analysis, focused on the Instruction Skip fault model, is provided in subsection 2.1.2.1. Then, the effectiveness of the main defense mechanisms against Fault Attacks (FA) is discussed in subsection 2.1.2.2 according to the previous threat analysis.

2.1.2.1 Threat analysis - Fault injection attacks against microcontrollers

This subsection provides a state-of-the-art of fault injection attacks against microcontrollers, mainly focused on EM perturbations and laser as injection medium. This task further gathers and analyzes experimental data related to attacks that leverage EM and laser fault injections in microcontrollers. The reported experiments allow to derive a very strong fault model, which shows that an attacker may be able to erase an arbitrary number of instructions from a microcontroller program at runtime.

2.1.2.1.1 Study of the Instruction Skip Fault Model

A Fault Model (FM) generally describes the main properties of a FA scheme, often expressed in terms of requirements of synchronization (requirement to fault a particular step of an algorithm or program) and of extension (requirement to limit the fault extension, e.g. to a single bit or a single byte). In this work, we

consider the FM related to EM and laser fault injection. It may be defined at different levels of abstraction from transistor or gate level to the assembler or algorithm level. We studied the FM of microcontrollers experiencing instruction skips.

An instruction skip is a fault that results in skipping, meaning not executing, one instruction of a microcontroller program at runtime (as if the program flow had skipped over the faulted instruction).

Several works studied the EM-induced instruction skip FM. Most of them assessed single instruction skips (in the same 8-bit microcontroller we used as target for [42], and on a 32-bit microcontroller for [261]). To the best of our knowledge, the only works reporting several successive instruction skips are [301] and [403]. [301] assessed four successive skips of instructions stored in the target instruction cache while [403] succeeded in faulting instructions stored in the target's pipeline. Their ability to induce successive skips was linked to the micro-architecture of their targets.

Several works also deal with laser-induced instruction skips. [66] obtained single instruction skips with high accuracy and high success rate (on the same microcontroller we studied) and used it to perform a successful differential fault attack on AES. Still on the same target, [214] reports instruction skips based on resetting one or two bits of the targeted instruction opcode. The authors of [359] induced instruction skips on a more complex 32-bit cortex-M3 microcontroller. They were able to inject two single instruction skips distant from 58 ms to defeat a protected CRT-RSA algorithm. In terms of target complexity, [363] reports injection of single instruction skips into a quad core ARM cortex A9 microprocessor running at 1.4 GHz clock frequency. Hence, the state-of-the-art in laser-induced instruction skip was limited to single instruction skips (with a repetition rate in the range of tens of ms).

There is to date very few explanations of how an instruction skip is induced at the gate level, with the notable exception of [27]. It describes how increasing progressively the stress applied by a clock glitch to a microcontroller induces an increasing number of bit-reset faults into an instruction opcode. It results in (1) instruction modification at low stress or (2) in turning the instruction into an actual `nop` at high stress (i.e. the `no operation` instruction). Instruction modification achieves an instruction skip if the modified instruction has no effect on the code operations (instruction skips are often actual code modification); the same is true for turning an instruction into a `nop`. An analysis of single instruction skips due to instruction modification induced by laser is reported in [105]. It relates how faulting one bit of an instruction opcode led to two successful FAs.

Our research objective was to reproduce EM and laser-induced instruction skips on a microcontroller and to study the main characteristics of its FM: accuracy, extent, success rate, time between successful skips, etc. Our aim was also to assess whether the single instruction skip fault model could be extended further to multiple instruction skips. This latter aim was of interest because defenses are based on the known FMs. Hence, a defense based on a too narrow FM may reveal vulnerabilities at test time. From previous experiments and taking into account the results of [27, 65, 66], we focused our experiments toward achieving instruction skips by turning the target instructions into `nop` instructions. Our experiments were carried out with the same 8-bit microcontroller studied by [27, 42, 65, 66, 214] for comparison purposes and also to ease the analysis of the obtained results.

Test setup: We chose a simple target for the purpose of being able to analyse easily its answers to fault injection: an 8-bit non-secure ATmega328P microcontroller designed in the old CMOS 0.35 μm technology. It has 2 kB RAM, 3 kB Flash and 1 kB EEPROM memories; a Harvard architecture with a 2-stage fetch-execute pipeline. It runs at 16 MHz and has 32 general purpose registers. Registers r16 to r25 were used during our experiments.

We studied the effect of EM and laser-induced faults on dedicated test codes mostly written in assembly langage. Our intent was to induce and analyze instruction skips by examining their effect on the assembly instructions of the test codes. For each test series, we used two trigger signals for synchronization purposes (two outputs of the test chip):

- a *synchronization* trigger signal to accommodate for the latency of the laser source,
- a *core* trigger signal to synchronize fault injection with the part of the assembly code of interest.

The listing in Fig. 2.3 provides a description of the test code we used to tune our settings in order to induce instruction skips. The core part of the test code (encompassed by the core trigger) is a series of ten `ld rX, Z+` instructions, each one corresponding to a load in a destination register `rX` of a byte value stored in RAM memory at address `Z` with a post increment of `Z`. Prior to that, the ten destination registers, `r16` to `r25`, are initialized at `0x55` and an array of ten byte values `0x39` to `0x30` are stored in RAM with `Z` storing the address of its first element. Registers `r16` to `r25` are read back after the synchronization trigger is reseted (the two top wavefroms in Fig. 2.4.a and 2.5.a are the synchronization and core triggers). The top part of Table 2.1 displays the values read back from `r16` to `r25` for a fault-free execution

As an example, the right column of the test code core part in Fig. 2.3 displays the effect of a laser shot turning the `ld` instruction of line 9 into a `nop` instruction. The effect of such a laser-induced instruction skip

```
# Store 0x39 to 0x30 in RAM at address Z
# Initialize r16 to r25 at 0x55
# Set synchronization trigger
nop # 400 ns
# Set core trigger
ld r16,Z+      ld r16,Z+
ld r17,Z+      ld r17,Z+
ld r18,Z+      ld r18,Z+
ld r19,Z+      nop
ld r20,Z+      ld r20,Z+
ld r21,Z+      ld r21,Z+
ld r22,Z+      ld r22,Z+
ld r23,Z+      ld r23,Z+
ld r24,Z+      ld r24,Z+
ld r25,Z+      ld r25,Z+
# Clear core trigger
nop # 700 ns
# Clear synchronization trigger
# read back r16 to r25
```

Figure 2.3: Test code for instruction skip analysis

is highlighted in the bottom part of Table 2.1: the initialization value `0x55` is read back from `r19` (in red), and because an increment of address `Z` is missing, all the values read back from `r20` to `r25` are shifted (in gray).

Table 2.1: Registers r16 to r25 readback values, for a fault free execution (top) and for an instruction skip targeting r19 (bottom, highlighted in red).

Register	16	17	18	19	20	21	22	23	24	25
Fault free	0x39	0x38	0x37	0x36	0x35	0x34	0x33	0x32	0x31	0x30
Faulted	0x39	0x38	0x37	0x55	0x36	0x35	0x34	0x33	0x32	0x31

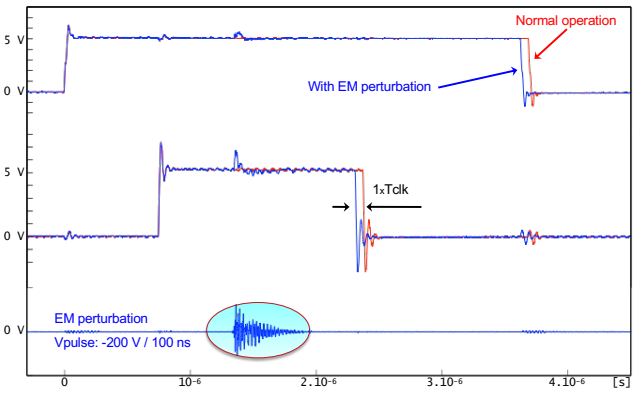
2.1.2.1.2 EM-induced Instruction Skips

Injection Setup: the EM pulse injection setup we used is described in [251]. The EM disturbance that induces a fault is generated thank to a voltage pulse generator: it delivers a square voltage pulse with a transition time of 2 ns, a maximum amplitude of $\pm$400 V in absolute value, and a minimum width of 6 ns. The voltage pulse edges are converted into current variations in a coil found at the tip of a handcrafted injection probe. The probe is made of three turns of copper wire around a ferrite core about 500 μm in diameter. The swift current variation induces an EM perturbation at the root cause of the injected faults. A trigger signal generated by the device under test synchronizes the voltage pulse with the operation of the microcontroller. The voltage pulse was set to -200 V and 100 ns.

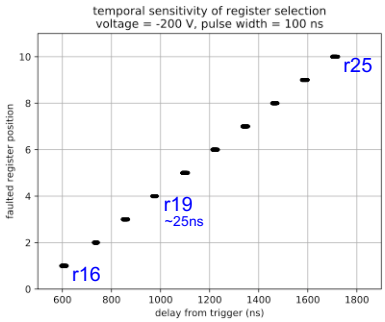
The left part of Fig. 2.4 displays the trigger signals of the test code (two top waveforms) and an image of the EM perturbation (third waveform). The trigger signals are both drawn for a fault-free execution (in red, denoted normal operation) and for an EM-injection inducing an instruction skip of the `ld` instruction into register `r19` (in blue). Because execution of a `ld` instruction takes two clock periods contrary to a `nop` instruction which takes one clock period, each consecutive instruction skip shall correspond to a reduction of the test code of one clock period. This phenomenon is displayed in Fig. 2.4.a for a single instruction skip. The test code execution time is shortened as well as the duration of the triggers signals: the fault free execution triggers in red last one clock period more than the faulted execution that is drawn in blue.

In terms of accuracy, we also tested whether this single instruction skip fault model was still valid while targeting the `ld` instruction of the other test registers. Our aim was to assess an attacker ability to target arbitrarily a single instruction of a program. To do so, we varied the time delay between the EM-perturbation and the synchronization trigger signal to span the whole test code. Figure 2.4.b reports the obtained results. It displays the skipped registers as a function of the delay. It reveals that an attacker is able to inject EM-induced single instruction skips into a running microcontroller with high timing accuracy. For each instruction, we were able to find an injection timing leading to 100 % success rate.

Some experimental results (not reported here) suggested that several consecutive instructions may be skipped simultaneously. For the purpose of verifying this assumption, we again carried out our experiments with a voltage pulse amplitude increased to -250 V. The corresponding results are displayed in Fig. 2.5.a. It shows



(a) Triggers waveforms modification due to EM injection

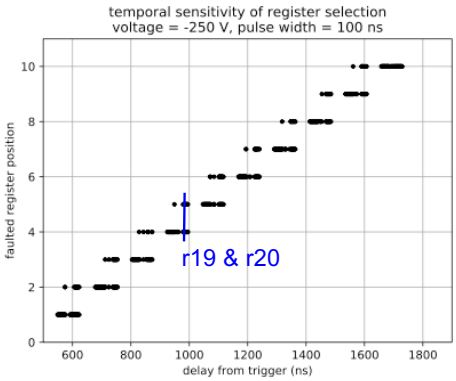


(b) Faulted registers as a function of EM injection time

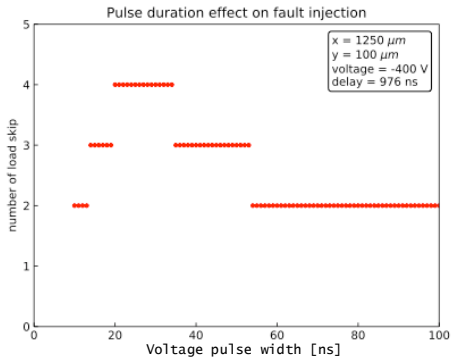
Figure 2.4**:** EM-induced single instruction skip: effect on execution time (a), ability to choose the skipped instruction (b)

that this increase makes it possible to skip two consecutive instructions with a still high timing accuracy (i.e. the ability to choose the two skipped instructions).

We also studied the tuning of an another parameter of the voltage pulse: its duration. And indeed, different voltage pulse durations leaded to different number of instruction skips as reported in Fig. 2.5.b. Up to four successive instruction skips were obtained (the maximal number we were able to induce).



(a) Faulted registers as a function of EM injection time



(b) Number of EM-induced instruction Skips as a function of the voltage pulse duration

Figure 2.5**:** EM-induced instruction skip: ability to skip two consecutive instructions (a), effect of the voltage pulse duration (b)

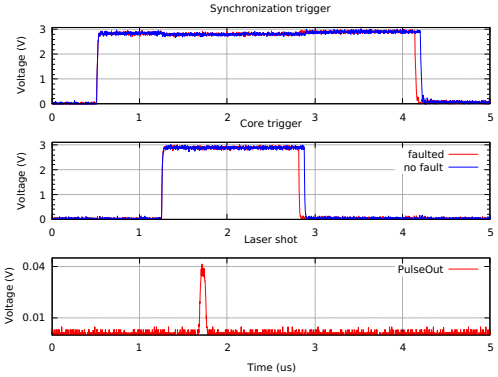
EM-induced Instruction Skips Fault Model: these experimental results demonstrate that a very high accuracy is achievable: we were able to choose and skip a single instruction into a test sequence with a 100 % success rate. Moreover, we were able to increase its extent to skipping four successive instruction skips which adds to its strengh (unlike [301, 403], the ability to skip consecutive instructions was not linked to the micro-architecture of the target).

2.1.2.1.3 Laser-induced Instruction Skips

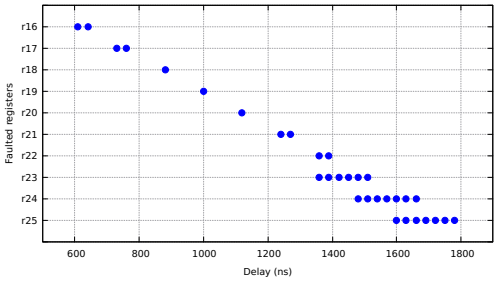
Injection Setup: our laser injection setup (it is described in [138]) has a nanosecond range laser source able to output a laser pulse with a 50 ns to 1 s tunable duration. It has a latency of less than 300 ns (i.e. the time interval between the trigger signal and the moment an actual laser pulse hits the target). Its wavelength is 1,064 nm (or near infrared, NIR). The laser source max power is 3 W (measured at the fiber optic output) which is enough to inject faults into the target. We performed our experiments with a $\times 20$ objective lens: it outputs a laser spot diameter of $5\,\mu m$. The laser pulse was set to 0.4 W and 75 ns for the first experiments.

Similarly to EM-injection, we were able to find settings that makes it possible to induce a single instruction skip with a 100 % success rate. Figure 2.6.a displays the trigger signals of the test code (two top waveforms) and an image of the actual laser shot (third waveform). The trigger signals are both drawn for a fault-free execution (in blue) and for a laser-induced instruction skip of the `ld` instruction into register `r19` (in red).

We also tested the ability the skip any instruction of the test code by varying the time delay between the laser shot and the synchronization trigger signal. Figure 2.6.b reports the obtained results. It displays the skipped registers as a function of the delay. It reveals that an attacker is able to inject laser-induced single instruction skips into a running microcontroller with high timing accuracy. For each instruction, we were able to find an injection timing leading to 100 % success rate.



(a) Synchronisation trigger signals with and without a laser shot (depicted in red and blue resp.), and image of the laser shot



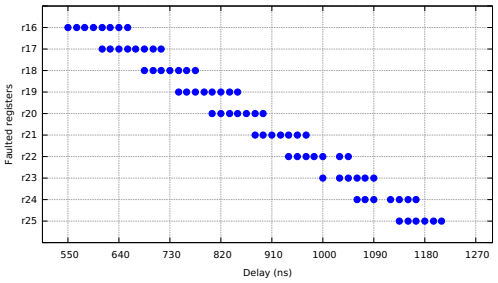
(b) Faulted registers as a function of laser injection time

Figure 2.6**:** Laser-induced instruction skip at 0.4 W and 75 ns: effect on execution time (a), ability to choose the skipped instruction (b)

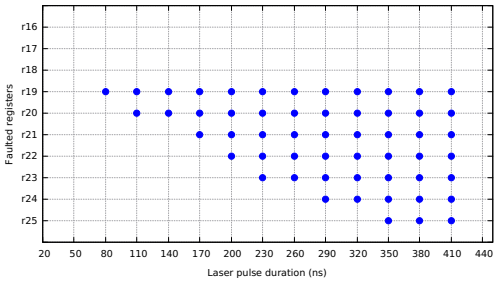
At some timings, two consecutive instructions were skipped (e.g. at 1,390 ns or 1,625 ns in Fig. 2.6.b), suggesting that several consecutive instructions may be skipped simultaneously. For the purpose of verifying this suggestion, we again carried out our experiments with a laser duration increased to 125 ns (two clock cycles) and a delay step set to 20 ns. The corresponding results are displayed in figure 2.7.a. It shows that increasing the laser duration to 125 ns makes it possible to skip two consecutive instructions with a still high timing accuracy (i.e. the ability to choose the two skipped instructions).

We also tested whether increasing the laser pulse duration would make it possible to skip an arbitrary number of consecutive instructions. The laser power was kept constant at 0.4 W, and the delay was set to target the `ld` instruction of register `r19`. The test series were carried out for a pulse duration ranging from 50 ns to 410 ns with an increment step of 30 ns. Figure 2.7.b reports the obtained results.

A first instruction skip was obtained for a laser pulse duration of 80 ns. Then, the number of instruction skips increased progressively with the pulse duration up to 7 consecutive skips at 350 ns. On average an additional instruction skip was obtained for every 60 ns increment of the laser pulse duration. For each number of instruction skips between 1 and 7, we were able to find settings leading to a 100 % success rate.



(a) Faulted registers as a function of laser injection time: laser power of 0.4 W and 125 ns



(b) Faulted registers as a function of the laser pulse duration

Figure 2.7**:** laser duration from 50 ns to 410 ns, 0.4 W laser power

Another test code was used to test further the ability to skip successive instructions with an increased duration of the laser pulse (see [138]). The laser power and delay were set to 0.5 W and 800 ns. As we increased progressively the laser pulse duration, an increased number of instructions were skipped, for which the shortening of the trigger signals was in accordance with the number of skips. Table 2.2 gives the number of obtained successive instruction skips for a selection of laser pulse durations.

It took a 20,400 ns long laser pulse to skip 300 instructions. We did not test the number of skipped instructions

Table 2.2: Number of obtained instruction skips vs laser pulse duration

Laser pulse duration (ns)	1,000	2,000	5,000	10,000	20,400
Number of instr. skips	17	33	82	143	300

beyond 300. There is a maximal number of instruction skips set by the endurance to laser illumination of the target circuit. Indeed, our device was destroyed when accidentally exposed to a continuous laser pulse at the same 0.5 W power. However, the device we used for these experiments showed no sign of fatigue after several tests at 20,400 ns laser pulse duration.

Our laser source has a repetition rate of 50 ns. It makes it possible to induce several instruction skips in the course of a program's subroutine. In order to assess the feasibility of this fault injection technique, we targeted a 4-digit PIN verification algorithm (described in [137]). It is protected against side channel timing analysis by a constant-time implementation: every of the four digits entered by the user are compared with those of a reference PIN.

We targeted successfully the four corresponding comparison loops as illustrated in Fig. 2.8: four carefully synchronized 60 ns long laser pulses were used to gain identification with a false user PIN.

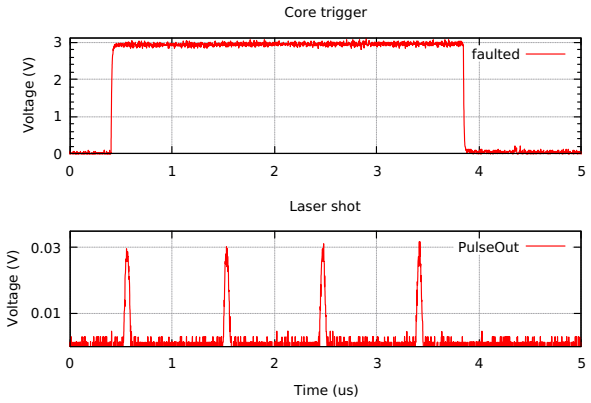


Figure 2.8: Bypass of a verify PIN algorithm with four separate laser pulses

Instruction Skip Fault Model

This research work assesses on experimental basis an extended fault model for EM and laser-induced instruction skips. The main characteristics of this fault model are:

- its accuracy, or ability to choose a single skipped instruction with a 100 % success rate provided a precise synchronization is obtained (EM and laser),
- its extension, or ability to skip an arbitrary number of successive instructions for laser injection (in case of EM injection the number of instruction skips is limited to four),
- its flexibility, or ability to skip several sections of the targeted firmware (for laser injection).

Simply put, laser fault injection may offer an attacker the ability to erase chosen parts of a microcontroller firmware at runtime. This experimental assessment speaks in favor of considering this fault model when designing defenses. A task that may prove difficult to complete given the assumption that any part of a software defense might be skipped as well.

2.1.2.2 Analysis of Existing Defenses against Fault Injection Attacks of Microcontrollers

Microelectronic devices are prone to errors due to non-intentional causes (radiations, EM coupling, etc.) that leads to the design of fault-tolerant systems [210]. The related countermeasures, either hardware or software, provided the basis of the first defense mechanisms against fault injection attacks [30]. Though, fault attacks are intentional attacks carried out by an attacker for the purpose of extracting information (e.g. a cryptographic key) or gaining an unauthorized access (e.g. bypass a verify PIN algorithm).

Since then, many defense mechanisms have been introduced [33]. In this work we only consider those that may apply to the *legacy component* microcontrollers that can be found in connected devices or industrial CPS. As a result, we do not consider hardware countermeasures such as detectors (e.g. light sensors to detect laser illumination, voltage glitch monitoring, etc.) that would require a redesign of the microcontrollers: we focus on

software countermeasures. We also address, on a lesser extent, hardware-assisted software defenses that apply to open core architectures (i.e. RISC-V ISA).

2.1.2.2.1 Existing software countermeasures against fault attacks

This subsection describes the main usual software defenses used to thwart fault attacks.

Randomization

The instruction skip FM described in subsection 2.1.2.1 assesses the ability for an attacker to skip a chosen instruction (or group of instructions) with a 100 % success rate provided an accurate synchronization of the injection process w.r.t. the target's activity is achieved. This is a strong requirement that suggests to use desynchronization as a countermeasure. Indeed, introducing a certain level of randomization into a program execution [30] will deny an attacker the ability to target specifically any of its instructions. Randomization can be built from random numbers of dummy instructions inserted in the course of a program, or by jumping randomly between different of its subparts (when compatible with the considered algorithm). As an illustration, inserting randomness in the PIN verification algorithm whose attack was reported in the previous section (illustrated in Fig. 2.8) would have prevented it from succeeding.

However, randomization is not an absolute defense:

- it only delays an attack success if the attacker is allowed to carry out several attack attempts in a row (thus degrading the attack success rate),
- some attack schemes do not require an accurate timing of fault injection to succeed (e.g. the attack of a RSA algorithm reported in [359]).

Redundancy

Redundancy is a long-established defense against faults [30, 33, 210]. It consists in executing several times the same subprogram and to compare the outputs. It is based on the assumption that an attacker is not able to induce an identical fault into each execution. It exists several variants of redundancy, the more common are:

- code duplication with comparison that makes it possible to detect fault injection,
- code triplication with majority vote that provides the ability to correct faults injected in one execution,
- duplication with inverse computations (e.g. for an encryption algorithm: perform the encryption and then decrypt the obtained cipher, any difference from the initial input indicates that a fault was injected) that further reduces the ability of an attacker to induce several times the same fault because the target's computations are different.

However, redundancy-based countermeasures suffer from a high overhead in execution time which is at least doubled or tripled (and possibly in code size as for duplication with inverse computations). Moreover, the previous state-of-the-art in fault injection reported an ability to skip two instructions sufficiently away in time (several ms for laser injection [359], and possibly less for EM injection given the 50 μs repetition rate of the used voltage pulse generators).

As a result, duplication techniques applied at instruction level were introduced for the purpose of choosing carefully the protected instructions. In addition, it may save code overhead (all instructions are not equally sensitive), allow quasi-immediate detection or mitigation of the injected fault. These defenses were built on the assumption that a fault (an instruction skip in our case) is restricted to a single instruction.

Initial code	Redundancy-based defense

`ldr r1, [r0]`	`ldr r1, [r0]`
	`ldr r2, [r0]`
	`cmp r1, r2`
	`bne <error>`

Table 2.3: Redundancy-based software defense against fault injection - fault injection detection [32]

Barenghi et al. introduced in [32] a framework based countermeasures solution applied at the instruction level of a program. Three techniques within the framework are considered: instruction duplication, instruction triplication, and a parity check method. The main principle of this defense is based on duplication and comparison. It is illustrated in Table 2.3 for a load instruction of a value stored in RAM into a register: `ldr r1, [r0]` .

A second register `r2` is used in the duplicated instruction. Then two additional instructions are used to compare the values stored in the two registers and branch into an error handler if the content of the two registers differs (hence revealing a store instruction was faulted). This defense is designed to provide data and code integrity. The authors detailed these countermeasures with tests on a AES implementation and suggested that the framework is employable on ARM processor family, starting ARM7 that use ARMv3 architecture. From the overhead point of view, execution time and code cost might be high.

Moro et al. proposed in [260] a variation on this countermeasure based on duplication without detection to cope with the single instruction skip fault model. It is exemplified in the top part of Table 2.4 for an instruction adding 1 to the content of a register and storing the result in a second register. A simple duplication of this instruction is sufficient to assure a mitigation against a single instruction skip (i.e. assuring that the correct result is stored in the destination register even if one of the two instruction is skipped). This defense was suitable for this specific instruction because its duplication did not change the result, a property called idempotence.

Initial code	**Idempotent instructions**	**Duplication-based defense**
`add r1, r0, #1`		`add r1, r0, #1` `add r1, r0, #1`

`add r1,r1,r2`	`add r3, r1, r2`
	`mov r1, r3`
		`add r3, r1, r2`
		`add r3, r1, r2`
		`mov r1, r3`
		`mov r1, r3`

Table 2.4**:** Duplication-based software defense against the single instruction skip fault model [260]

Duplication of the addition instruction in the lower part of Table 2.4 was not feasible without changing the result because the same register `r1` is used both as a source and a destination register. However, using an additional register `r3` in a move instruction turns into an idempotent series of two instruction that can be safely duplicated (see resp. the mid and last columns of Table 2.4). The authors formally verified the efficiency of this countermeasure on a 32-bit ARM Cortex-M3 microcontroller, based on an ARMv7-M architecture that runs the Thumb-2 instruction set; the application tests were applied to the AES and SHA-0 algorithms. The overhead, when applying this countermeasure, is important as reported by the authors: it may be higher than 100 % in clock cycles number and 200 % in code size. Hence, they suggest as a solution that it can be applied in selected sensitive parts of the target program to reduce the overhead.

This countermeasure was applied by Barry et al. in [35] on a modified Low Level Virtual Machine (LLVM) compilation tool as a generic mechanism to protect a code. The authors introduced a new approach that generate for all instructions an equivalent idempotent instructions and then proceed with the related duplication process. They also proposed an instruction scheduling mechanism that proceed to rearrange the execution order of the modified instructions to ensure a better execution time and an increased resistance to instruction skips. With this approach, it was possible to reduce the execution speed overhead and code cost by approximatively the half from the results obtained by [260].

Control Flow Integrity

The instruction skip fault model provides an attacker with the ability to tamper with the *Control Flow Graph* of a program at runtime (e.g. by skipping a jump or branch instructions). A usual defense against such CFG modifications is the use of *Control Flow Integrity* techniques. Following the work of Abadi et al. [3], CFI makes it possible to detect violation of a program CFG. The various variations of this technique are described in the CFI paragraph of subsection 2.1.1.1 dedicated to defenses against code-oriented attacks. They can be harnessed to mitigate CFG modifications induced by fault injection (e.g. as suggested in [289]).

A counter-based CFI technique introduced by [8] and later improved by [218, 289] is an inspiration for the defense mechanism we are developing. It is based on counters initialized at the beginning of any basic block (BB) of the CFG. Increment instructions are inserted between the BB instructions to count them. The counter value is then checked (at the end of the BB) against a fault free reference and a conditional branch into an error handler is taken if a difference is found [8]. [218] moved the counter check to the beginning of the destination BBs and interleaved it with initialization of the target BB instruction counter to extend this defense beyond BBs to the edges of the CFG.

Hardware-assisted software countermeasures

Danger et al. provide in [119] a solution based on extra hardware implemented block named Code and Control-Flow Integrity (CCFI). The method is presented as a generic countermeasure and is not intrusive since it does not need any modification of the core. This extra block is composed of two parts: a first part is used to store metadata related to the code and control flow information, and the second module is needed for the integrity check of both the code and the control flow. This method was tested on an implementation of PicoRV32 based on RISC-V ISA (three-stage pipeline). Their evaluation showed that it provided protection against simulated fault attacks. Yuce et al. demonstrate in [404] the efficiency of a new proposed countermeasure. The evaluation was proceeded by comparison of the protected code to both its unprotected version and its protected version using the instruction duplication method. The presented countermeasure is based on a hardware detector combined with a software block that handle the fault flag and run the application specific fault response.

2.1.2.2.2 Theoretical analysis of software countermeasures against fault attacks

The efficiency of the software countermeasures reported in the previous subsection is still to be assessed experimentally against the instruction skip fault model we introduced in subsection 2.1.2.1. However, we propose a first theoretical assessment in the following.

Randomization: In essence, the extended instruction skip fault model we introduced seems not to endanger the randomization countermeasure. It is true for EM-based fault injection that makes it possible to skip only a few successive instructions with a low repetition rate. On the contrary, laser-induced instruction skips may target as high as hundreds of successive instructions. Depending on how randomization is implemented, its relevance can be diminished or offset.

Redundancy: Full software redundancy as described in [35] can be defeated by both EM and laser injection. The 100 % success rate we highlighted (provided an accurate synchronization is obtained) suggests that an attacker would be able to induce the same fault into the redundant parts of the target. Instruction-level redundancy, as in [260], was designed to mitigate EM single instruction skip attacks on the assumption that the EM fault model was restricted to single instruction skips. Our experiments demonstrate that this assumption does not hold in practice: either EM or laser attacks are able to skip duplicated instructions with a 100 % success rate (as displayed in figures 2.5.a and 2.7.a).

CFI: CFI introduced a further level of useful complexity against fault attacks (especially if it also uses redundancy [289]). However, this defense is not absolute against an attacker who is able to erase several sections of arbitrary length of the targeted code. In a white box approach, the attacker would be able to select accurately the instructions to be skipped in order both to perform his attack and to deactivate the protections (e.g. the branch instructions into error handlers). This theoretical attack scenario is based on strong assumptions, though it is not beyond feasibility with an iterative attack that would defeat one software countermeasure after another.

2.2 Devising General Defense Mechanisms

Based on the thorough analysis, we device defenses for present and future devices against data-oriented and code-oriented attacks due to memory corruption vulnerabilities and against fault injection attacks.

2.2.1 Defenses against Data-oriented Attacks through Virtualization

We propose a hardened system that leverages virtualization features to reduce the damages that memory corruption vulnerabilities (in particular data-oriented attacks) can introduce. We suggest to unify the techniques of the state-of-the-art research presented above and bridging their limitations to induce higher security. Specifically, we design and publish *selective memory protection (xMP)* [288], a technique that can be applied to isolate arbitrary memory regions in disjoint protection domains while, at the same time, providing integrity of the pointers that reference data in these isolated protection domains. We implement primitives that developers can use to protect sensitive data in both kernel and the user space. We exemplify our primitives on the highly critical data structures, namely paging tables and process credentials (`struct cred`), in the Linux kernel space. In the user space, we isolate and protect data structures used by security-sensitive libraries (e.g., OpenSSL), that maintain cryptographic material in memory. Further, we complement the data structure isolation features by additionally establishing a pointer authentication feature that grants access to the isolated data structures only in the right context. We implement the pointer authentication feature by using the Linux kernel implementation of SipHash, a Keyed-Hash Message Authentication Code implementation. In pointer authentication of xMP base its Keyed-Hash Message Authentication Code (HMAC) on a secret, that itself is isolated in an exclusive memory protection domain, so that only authorized code can access it.

2.2.1.1 Selective Memory Protection using EPT-based Isolation

Virtualization introduces a software layer occupied by the so called hypervisor that runs at a higher privilege level than the kernel and the user space tasks. Therefore, a hypervisor represents a good candidate for enhancing the security of the lower privilege layers. Modern CPU architectures extend their ISA with instructions that allow hypervisors to support the governance of virtual machines (or simply guest's). For this, such architectures introduce additional hardware-enforced data structures that help hypervisors in managing the shared hardware resources used by the virtual machines. These data structures (among others) enforce isolation between the guests of the virtualized system, which is needed because virtual machines can not be allowed to access each other's address space, and they must be given the impression that they have full control over the system (much like processes in the user space of a kernel host).

The data structures used to isolate the guests' memory address spaces additionally comprise a set of Second Layer Address Translation (SLAT) tables, named EPT on Intel. EPT represent an additional layer of paging structures used to translate the guest physical into machine (or host) physical addresses. Generally, a translation of a guest virtual address into a guest physical address (with help of the in-guest page table) maps a Guest Virtual Address (GVA) into a Guest Physical Address (GPA). Consequently, after walking the EPT, the GPA results in the Host Physical Address (HPA). This allows hypervisors to segment the physical address space into isolation domains associated with every VM guest running in the system. Nevertheless, the system does not restrict a hypervisor from configuring more than one EPT for a VM. For example, on Intel CPUs, the virtualization extensions allow to configure up to 512 EPT pointers in the Virtual Machine Control Structure (VMCS) (the central data structure that configures virtual machines). In other words, instead of using only one, global view on the guest's physical memory, a hypervisor is able to configure different views on the guest's physical memory (e.g., by marking a GPA as present in one EPT, while in another EPT it is marked as non-present).

This technique can be applied to guard critical in-guest data structures. Assuming a critical data structure resides on a particular guest physical page, one of the EPTs can be configured to restrict access permissions. Thus, it becomes possible to establish protection domains that guard the particular data structure. Such a protection domain can reside in two states: *(i)* the protection domain is in the relaxed state with a relaxed access setting for the GPA in the respective EPT (e.g. rwx), while *(ii)* the protection domain in the restrictricted state imposes a more restrictive configuration in a different EPT (e.g. r–). That is, to define one protection domain, we have to employ two EPTs (to implement the relaxed and the restricted state, respectively).

Moreover, Intel CPUs provide a virtualization extension instruction called `VMFUNC` that allows kernel and user space to switch to a particular EPT, without any hypervisor intervention. Therefore, when attempting to access a memory region that is isolated in domain X, the guest can use the `VMFUNC` instruction to switch the current EPT. That is, the `VMFUNC` instruction allows guests to define custom policies that define when to enter (or leave) the defined protection domains.

To implement this feature, in xMP [288], we leverage the Xen altp2m subsystem for configuring multiple EPT for a virtual machine and make use of its interface to the Linux kernel to maintain guest physical addresses in individual memory protection domains (represented through different EPT). To place a particular GFN inside a protection domain, we relax the GFN's memory access permissions in the associated EPT, and restrict its memory access permissions in every other EPT. To ensure that the attacker does not have access to the isolated memory regions, we also define one EPT that restricts access to all other domains. This memory view is enabled by default, so that potential data-oriented attacks cannot illegally modify the contents of the protected data-structures without first having to enter one of the protection domains. Yet, since the attackers do not have the necessary means (including access to the isolated secret key that is used to authenticate the entered protection domain) to switch the protection domains, xMP has the power to prohibit such attacks.

2.2.1.2 Pointer Authentication

While the isolation of sensitive data structures is a necessary step, the pointers can still be adjusted by adversaries with arbitrary write primitives. As such, to ensure that pointers cannot be (illegally) redirected, we can authenticate them before they are de-referenced.

Pointer authentication represent an effective security feature implemented by the ARM architecture to ensure memory pointer integrity. The ARM architecture has already implemented the **PAC!** (**PAC!**) extension, however, x86 lacks a similar functionality. In order to complete the protection of arbitrary memory, with xMP [288], we propose a lightweight, yet reliable pointer authentication implementation in software, which complements the memory isolation protection. If we take the highly sensitive data structure that holds the process credentials (struct cred) as an example, assuming the content of these objects is protected via the EPT management technique described above, the pointers that reference `struct cred` are still vulnerable to overwriting, since we don't isolate every parent structure that stores them.

A strong attacker equipped with arbitrary read and write capabilities, can learn the pointer of the privileged (and through xMP isolated) `struct cred`, and use it to overwrite the pointer to the `struct cred` residing in the associated thread control block (represented through the data structure `struct task_struct` on Linux) to elevate their privileges and compromise the system, even though the `struct cred` object are isolated. Therefore, we need to make sure that pointers to sensitive objects can not be reused in a context different from the one they were originally configured in. Pointers to `struct cred` instances are kept in the `task_struct` of a task. Therefore, we need to bind a `struct cred` to its corresponding `task_struct`, which prevents the `struct cred` to be used in another `task_struct`. Generally, a pointer to a sensitive object must be bound to the parent object that stores the pointer. Recursively, the parent object must be bound to the grandparent object, and so on, until the last ancestor is found in the composite chain, that is not reference by any other object in the hierarchy. In Linux, the process credentials maintained by `struct cred` is referenced by a pointer stored in the `task_struct` representing the thread, which in turn is referenced only by CPU registers, and, therefore, we treat it as the top of the hierarchy that doesn't need to be authenticated (as it is immutable for every thread).

In order to implement pointer authentication, we leverage the lightweight SipHash functionality present in the Linux kernel, which is designed as a secure cryptographic hashing function, but optimized to work efficiently on small data, such as 8-byte pointers. SipHash works with 128-bits keys to compute HMACs. We then, we equip the first (unused) 15 bits of pointers to sensitive data structures with the resulting (trimmed) HMAC. Therefore, we use the most-significant 15 bits of a pointer to store its immutable HMAC. Whenever the pointer is used, we require the context (parent data structure which hosts it) to be given as well, so that we can authenticate the HMAC and authorize the use of the pointer. In case of an inconsistency (resulted, e.g., through a malicious corruption of the pointer), the kernel crashes. The strength of a cryptographic has function relies in the secrecy of its private keys. Therefore, we assign a secret key for each protected data structure, and we isolate it in the memory domain used to isolate the data structure. Only authorized regions of the kernel are able to access the key, and we eliminate pointer references to the key, by statically injecting at compile time its memory address in the operands of the instructions that use it. The 15-bit HMAC does not represent a danger for a brute force attack, since the attacker has only one shot into guessing the HMAC, after which the kernel crashes.

2.2.1.3 Kernel Space

In the following, we describe practical applications of the virtualization- and pointer authentication based techniques described above on sensitive data structures used by the Linux kernel to protect them in the presence of memory corruption vulnerabilities.

2.2.1.3.1 Page Tables

Attackers with arbitrary read and write capabilities can run data-oriented attacks against page tables. This can be done by *(i)* revealing the location of the page tables of a specific process in memory, and *(ii)* overwriting page table entries to introduce new, malicious, mappings. This capability enables many attack vectors that grant illegal execution. For instance, by clearing the Supervisor Mode Execution Prevention (SMEP) bit in the page table entries, the attacker could allow user space pages to be executable in kernel space, and therefore, redirect the kernel's control flow to injected malicious payload in user space. This can be addressed by placing the paging structures in dedicated protection domains (that are protected by EPT).

As such, we extended xMP [288] to protect pointers to page table structures, which we isolate in dedicated protection domains. For that, we extended the Linux kernel buddy allocator, in order to establish an interface between the Linux kernel and the Xen altp2m subsystem. This allows us to isolate guest physical pages. By further adjusting the Linux kernel page table management system, we manage to isolate all pages that belong to the paging structures of every process in dedicated protection domains. Whenever the kernel needs to adjust the paging structures and (legally) modify their entries, we switch to the specific protection domain that grants write-permissions to the paging structures.

2.2.1.3.2 Process Credentials

An attacker with an arbitrary write primitive could subvert the system by illegally modifying the process credentials (`struct cred`). Instances of `struct cred` are allocated from the kernel heap, using the kernel slab allocator. With few exceptions (kmalloc), the slab allocator groups objects of certain type across one or multiple pages. Such pages are thus referred to as slab caches. For instance, the data structure `struct cred` is allocated from the cache that is called `cred_jar`. To be able to protect all data structures of a certain type, we adjusted the slab allocator to communicate with the modified buddy allocator, in order to isolate entire slab caches in dedicated protection domains. For instance, to protect illegal write-accesses to instances of `struct`

`cred`, we define the protection domain such that its contents can be read, yet, not written to. Whenever the kernel needs to modify the fields of a `struct cred` instance, it switches to the specific protection domain, satisfies the write, and finally switches back to the default view (to restrict the domain).

In addition to protect the integrity of pointers to process credentials, we apply the pointer authentication feature (described above). Also, to avoid a potential redirection to high-privileged process credentials, we bind the `struct cred` pointers to the associated `task_struct` (representing the thread) so that they can not be reused in a different context.

2.2.1.3.3 Linux Namespaces

The Linux namespace subsystem isolate the visibility to selected global resources. In this way, namespaces represent the building blocks of OS-level virtualization. We foresee a high degree of compatibility between Linux namespaces and xMP. Therefore, we introduced a new Linux namespace that aims at improving container isolation by registering the paging tables of processes that belong to different namespaces into different disjoint memory protection domains. Therefore, a compromised container that is able to alter its own paging structures, is not able to manipulate the paging structures of other containers or the paging structures of the kernel itself. We introduced a new namespace, that causes the process to move its paging structures into the foreseen protection domain. Children of this process join the parent's paging table namespace and, therefore, maintain their paging structures in the same memory protection domain.

2.2.1.4 User Space

Similar to protecting critical data structures in kernel space, the memory isolation and pointer authentication features of xMP can be equally applied to sensitive data in user space. For instance, cryptographic libraries maintain cryptographic material in memory and thus can become subject to attacks. Recent vulnerabilities (Heartbleed) have revealed that sensitive tools with memory corruption vulnerabilities highlight the need for memory protection primitives in user space. Therefore, we applied our hardening techniques on popular open-source cryptographic libraries, such as OpenSSL, ssh-agent, mbed TLS and libsodium. We implemented system calls on the kernel side, that allow user space applications to create disjoint memory isolation domains, in which sensitive data can remain safe. Whenever the process needs to read or write a sensitive data structure, it must explicitly switch to the allocated isolation domain, execute the access, and switch back to the default domain.

2.2.1.5 Evaluation

As taken from [288], we measured the performance impact of xMP protecting the kernel's page tables (PT) and process credentials (Cred). We used a set of micro (LMbench v3.0) and macro (Phoronix v8.6.0) benchmarks to stress different system components We measured the overhead of protecting (i) each data structure individually, and (ii) both data structures at the same time (which requires two disjoint protection domains).

Table 2.5 illustrates the LMbench latency and bandwidth overhead results, to clarify the performance cost at the system software level. In most cases the overhead is low for both protected page tables and process credentials. When using xMP to protect page tables, we notice that the performance impact is related to functionality that requires access to page tables [288]. When protecting the process credentials (`struct cred`), we observe that although the kernel accesses the `struct cred` protection domain, the overhead for creating new processes is low. Yet, the same protection domain is heavily used during file operations, which require access to `struct cred` for access control. Besides, the performance impact of the two protection domains is additive in the setup protecting both page tables and process credentials (PT+Cred) [288].

Table 2.5: Performance overhead of protection domains for page tables, process credentials, and both, measured using LMbench v3.0 [288].

	Benchmark	PT	Cred	PT+Cred
Latency	`syscall()`	0.42%	0.64%	0.64%
	`open()/close()`	1.52%	75.74%	78.93%
	`read()/write()`	0.52%	150.84%	149.27%
	`select()` (10 fds)	2.94%	3.83%	3.83%
	`select()` (100 fds)	0.01%	0.31%	0.30%
	`stat()`	-1.22%	52.10%	53.33%
	`fstat()`	0.00%	107.69%	107.69%
	`fork()+execve()`	250.04%	9.36%	259.59%
	`fork()+exit()`	461.20%	7.78%	437.31%
	`fork()+/bin/sh`	236.75%	8.49%	240.64%
	`sigaction()`	10.00%	3.30%	10.00%
	Signal delivery	0.00%	2.12%	2.12%
	Protection fault	1.33%	-4.53%	-1.15%
	Page fault	216.21%	-2.58%	216.56%
	Pipe I/O	17.50%	32.87%	73.47%
	UNIX socket I/O	1.16%	1.45%	2.25%
	TCP socket I/O	10.23%	20.71%	37.13%
	UDP socket I/O	13.42%	21.98%	41.48%
Bandwidth	Pipe I/O	7.39%	7.09%	17.49%
	UNIX socket I/O	0.10%	6.61%	13.40%
	TCP socket I/O	6.89%	5.83%	14.53%
	`mmap()` I/O	1.22%	-0.53%	0.83%
	File I/O	0.00%	2.78%	2.78%

Table 2.6: Performance overhead of protection domains for page tables, process credentials, and both, measured using Phoronix v8.6.0 [288].

	Benchmark	PT	Cred	PT+Cred
Stress Tests	AIO-Stress	0.15%	5.87%	5.99%
	Dbench	0.43%	4.74%	3.45%
	IOzone (R)	-4.64%	26.9%	24.2%
	IOzone (W)	0.82%	4.43%	7.71%
	PostMark	0.00%	7.52%	7.52%
	Thr. I/O (Rand. R)	2.92%	7.58%	10.13%
	Thr. I/O (Rand. W)	-5.35%	3.01%	-1.29%
	Thr. I/O (R)	-1.06%	19.54%	20.08%
	Thr. I/O (W)	1.34%	-1.61%	-0.27%
Applications	Apache	6.59%	9.33%	11.14%
	FFmpeg	0.14%	0.43%	0.00%
	GnuPG	-0.66%	-1.31%	-2.13%
	Kernel build	11.54%	1.84%	12.71%
	Kernel extract	2.89%	3.65%	5.91%
	OpenSSL	-0.33%	-0.66%	0.99%
	PostgreSQL	4.12%	0.32%	4.43%
	SQLite	1.10%	-0.93%	-0.57%
	7-Zip	-0.30%	0.26%	0.08%

Table 2.6 presents the results for the set of Phoronix macro-benchmarks [288], split into stress tests, targeting one specific system component, and real-world applications. Overall, with only a few exceptions, the results show that xMP incurs low performance overhead, especially for page table protection.

Further, as discussed in [288], we evaluated the overhead of in-process memory isolation using our xMP-protected versions of the Nginx and mbed TLS servers (Figure 2.9). We used the server benchmarking tool `ab` to simulate 20 clients, each sending 500 and 1,000 requests [288]. To compare our results with related work, we run the Nginx benchmarks with the same configuration used by SeCage [235].

In most cases, the results show that the throughput and latency overhead is small. In contrast to to SeCage,

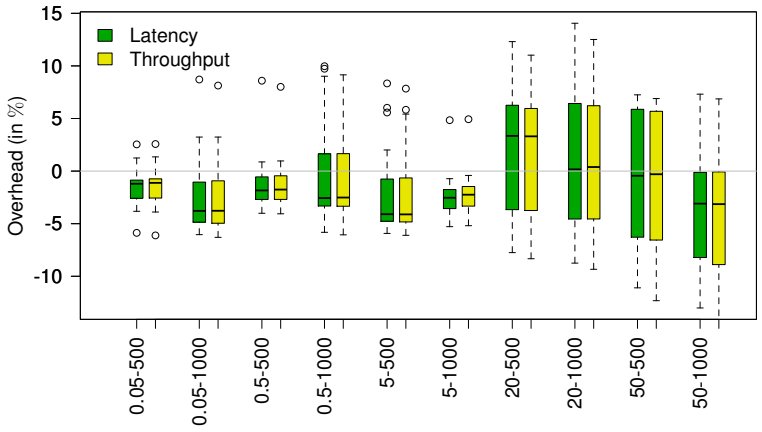


Figure 2.9: Performance impact of xMP on Nginx with varying file sizes and number of connections (X-axis: [file size (KB)]-[# requests]) [288].

with up to 40% overhead for connections without KeepAlive headers and additional TLS establishment, xMP does not have similar issues in such challenging configurations[288]. The average overhead for latency and throughput is 0.5%. For mbed TLS, we used the `ssl_server` example to execute an SSL server hosting a 50-byte file. On average, the overhead is 0.42% for latency and 1.14% for throughput.

2.2.2 Defenses against Code-oriented Attacks through Hardware-based Monitoring

As previously shown in 2.1.1.1, Control-Flow Integrity (CFI) seems to be the most promising defense technique to counter Code-Oriented Attacks. CFI is different from the other techniques first because of its being at a different level of abstraction: instead of trying to hide or eliminate memory vulnerabilities to prevent corruption of code pointers, the worry is about making these corruptions ineffective. CFI by itself does not obviate the problems of memory corruption, and yet this is one of its strongest points. Despite the wide range of tools today available, making the memory invulnerable is still an open problem, which spans all stages of production and activity of an application, from proper training of the programmers to the most advanced runtime protection techniques. Given a program, it is practically impossible to ensure with certainty that it is not vulnerable, at least from a formal point of view. The protections are often limited to only one part of the memory (e.g., stack, heap), despite of the others, or they are made in opposition to a particular *theat model*, while all the others are forgotten. CFI is strong because it succeeds in abstracting from these complications and look at the problem from another perspective.

Ensuring that the control flow is always consistent with the intended one, means involving, somewhere, a component that can verify that the transfers that the program is making at runtime are valid. This component is referred to as *CFI monitor*. Different kinds of CFI monitors can be implemented, first in software or in hardware. Comparing these two kinds of monitor, drawbacks and benefits can be highlighted, and the choice of one rather than of the other depends also on the overhead introduced, and on the impacted resources. There is no doubt that software techniques are more modular, more adaptable to different platforms and more easily modifiable. However, they certainly involve an extra occupation of memory, data or code, as well as the need to keep sensitive data on CFG isolated from the rest of the execution environment, which is not always feasible. On the other hand, hardware techniques will be more difficult to port, but they provide:

- higher efficiency, since validity checks are made *faster* and *in parallel* with respect to the execution;
- greater isolation from the execution environment;
- sometimes, total transparency with respect to the program performed.

All these features make the hardware-based CFI a very important sector in which to invest for the defense of information systems. Despite the great variety of solutions proposed as we have seen in 2.1.1.1, the research is not to be considered limited to finding more efficient forms of hardware CFG monitoring, but must also go in the direction of alleviating the main intrinsic drawback of these solutions: *the fact that they are not soft*.

In fact, for example, none of the presented hardware-based solutions can be applied to a device that is already operating in the field, since each of these requires an even minimal hardware patch, only possible when a new version of the device is released. Some authors boast of not modifying the internal structure of the processors, although they know very well they are using a *linguistic stratagem*: installing a CFI verifier between the instruction cache and the core is not that much different from inserting an additional module in the pipeline,

because an intrusion is still required in the original design, which means *redesign*, even if the processor in itself is the same as before.

Therefore, it is advisable to look for solutions that combine the advantages of having a hardware-based defense, for the reasons explained above, and the portability advantages of soft solutions. In this regard, FPGAs should be taken into great consideration. These reconfigurable components are the main candidates for the role of hardware-based CFI monitor, also due to their increasing use in computer architectures. According to latest Gartner[1] research about the future of Infrastructure and Operations [2], FPGA will be part of the top 10 technologies to drive innovation through 2024.

The most recent strategies depict a primary interest of using FPGA in server-side hybrid chips. Nevertheless, the rise of 5G technology, the consequently spread of IoT and OT infrastructures and the need for real time insights and localised actions, are forcing to deploy edge-computing solutions to process data closer to the source of generation. It is expected that, over the next few years, hardware vendors will focus on delivering computing hardware to execute complex, compute-intensive functions at the edge. In this context, hybrid chips based on CPU and FPGA components, will be the easiest and most power/cost-effective way to meet the new edge computing hardware requirements. Although there are still a few examples on the market, mostly provided by FPGA vendors who embed ARM or NIOS cores in their devices, hybrid CPU+FPGA chips are expected to become increasingly popular in the next years.

FPGA and CPU devices are already employed in many projects as separate components interconnected through a parallel bus and mounted on the same electronic board. In most of the cases, the FPGA is mapped as a memory device whereas the CPU acts as a master of the system. However, the mobile terminals market is driving a new trend, which aims to replacing the parallel bus with serial *differential lines* in order to reduce the final device size and, at the same time, to increase the data transfer rate. In terms of architectural access, we are talking of a migration from *memory-mapped* devices to *port-mapped* devices. In any case, since the new serial buses affect the memory components, it is expected that the CPU will adapt the instruction set to atomically manage the memory access with a *single-instruction paradigm*, either mapping the LOAD/STORE instructions to the new serial buses or introducing IN/OUT instructions to manage the serial memory access.

CINI is working on these CPU-FPGA cooperation aspects for devising a generic and lightweight defense technique, which is based on monitoring the software run by a CPU by exploiting a standard parallel communication interface with an FPGA, external or internal to the chip, which hosts a CFI monitor as a core synthesized onto it. The technique uses a minimal binary instrumentation based on single STORE machine instructions to communicate to the monitor the information about the status of the CFG. Basically, the program informs the monitor, before branching, about its position within the code, and communicates it again immediately after the branch. These two positions are coupled inside the FPGA monitor, which checks whether the pair is contained in a *table* that contains all the valid edges. If the pair is not found, or the second position is never sent to the monitor, it means that an attack has been launched, and the attacker has jumped elsewhere, to a code section which is different from the one originally established by that branch instruction, so the execution is interrupted. The solution is suitable for any type of platform, future but also already in the field (*legacy*). The technique does not require the modification of the internal structure of the microcontrollers, but just requires the availability of an FPGA, without the need to fabricate new silicon to start adopting the defense. Using a mixture of binary instrumentation and hardware-based supervision, the technique entrusts the binary enforcement with the sole task of informing the monitor about the status of the CFG through simple additional STORE instructions at critical points, and the hardware monitor with the conservation of the information about the CFG and the part of computation for the validation, thus obtaining advantages in terms of both isolation and performance.

In addition to the benefits introduced by the use of the FPGA, the technique is also innovative for the fact that it deals with a problem often forgotten in the literature on the topic. The problem is the protection of the execution context, put at the same level of importance as the protection of the branches, as in the opinion of the authors, if the former is absent, the latter loses meaning.

The problem was first presented in [246]: the execution sometimes undergoes deviations that cannot be calculated *a priori* with respect to the statically-defined CFG. These deviations are caused by the execution of Interrupt Service Routines (ISRs) in response to an interrupt hardware signal forwarded to the processor. In the abovementionend work, it is shown the impossibility to include these “phantom edges” in the CFG, because neither the source nor the target is known, and therefore it is not possible to protect them. However, these edges connect to code that, like the other, may contain vulnerabilities and may open the door to attacks. The result is that edge protection, to be effective, must be accompanied by context protection during the execution of the ISRs. The monitor must know the status of the program context as soon as the control flow enters an ISR, and must ensure that it has not changed when the canonical execution is restored.

[1] https://www.gartner.com/

2.2.3 Defenses against Fault Injection Attacks for Present and Future Devices

The software defenses against fault injection attacks (described in subsection 2.1.2) are mostly based on code redundancy and control mechanism of the integrity of the control flows of microcontrollers. A first theoretical analysis of their efficiency, against the extended instruction skip fault model we reported, revealed potential weaknesses. Indeed, the ability of an attacker to arbitrary skip instructions into the targeted code offers him the ability to skip also the instructions used to implement the countermeasures themselves. As a consequence, this problem does not have a straightforward solution in the software domain. It requires to resort to hardware assistance despite our objective to propose a software solution that is applicable to *legacy* microcontrollers (we do not want to introduce a defense that would require a hardware redesign of these microcontrollers). To that end, we propose a hardware-assisted solution that can be applied to legacy component: it derived from [8, 218] but uses hardware counters in spite of software ones.

Our approach takes advantage of embedded hardware timers or performance counters to measure the duration of a given section of a microcode (i.e. how many clock cycles it lasts). The measured duration is then compared to a reference value corresponding to a fault free execution (computed at compilation time). Any difference between the measured and reference durations reveals an ongoing instruction skip attack. As a matter of example, the instruction skip attack exemplified in Fig. 2.6 that results in a modification of the target execution time (as shown by the shortening of the trigger signals) would have been detected. This mechanism is based on the assumption that instruction skipping has an effect on the execution time of the running microcode and no effect on the hardware counters (that are not incremented by vulnerable software instructions). This technique makes it possible to implement a Control Flow Integrity (CFI) algorithm that ensures the integrity of the Control Flow Graph (CFG) and that of the instructions inside the CFG basic blocks. A first experimental validation of this approach on a verify PIN algorithm is in progress. Applied in conjunction with other mechanism it mitigates the threat of a PIN bypass also when using the full possibilities of the instruction skip fault model.

Chapter 3 Securing Operating System Software

3.1 Embedded OS Primitives for Future-proof Security

3.1.1 IoT crypto primitives

Strong cryptography is essential to protect communications and to authenticate online entities, including devices and the software that they run. For the conventional internet, the community has converged on standard protocol suites like TLS, which are composed of lower-level cryptographic primitives: algorithms such as hash functions, authenticated encryption, key exchange, and digital signatures. Securing communications always involves a trade-off between system security and the computational resources consumed to provide that security: even the most efficient cryptographic algorithms still cost power, memory, and CPU cycles. On low-end IoT devices, the challenge of optimising this trade-off is exacerbated by extreme resource constraints: memory in *kB*, power in *mW* or less, CPU in *MHz*, with no memory protection/management unit (MPU/MMU) or floating-point unit (FPU) in hardware.

Moving towards IoT, therefore, specific approaches must be developed to provide tiny crypto primitives without sacrificing speed and security. Cryptographers divide primitives into two broad clases: symmetric and asymmetric (public-key). Symmetric primitives, which include message authenticators and symmetric encryption tend to be highly efficient; still, the computational footprint of standardized symmetric algorithms like AES may be too big for some IoT applications. Some algorithms, such as Keccak, have been designed with variants using less internal state memory, which can reduce their impact on lower-end IoT devices. Going further in this direction, NIST is currently operating a competition and standardization process for so-called *lightweight* primitives, which have a drastically reduced computational footprint at the cost of lower security levels. We thus have a good range of maturing symmetric primtiives for IoT applications.

The main challenge is to push the limits of public-key systems towards *much* lower-end IoT hardware, providing algorithms and implementations for strong security that are versatile, portable, and energy-efficient. Public-key primitives, which include key establishment, identification, and digital signature protocols, typically involve intensive calculations in mathematical structures, and thus have a much larger computational footprint than their symmetric cousins Compared with the progress already made for symmetric primitives, IoT-focused public-key systems are still in their infancy.

Our goal here is to identify and further develop efficient signature and key establishment algorithms that are capable of being run on the widest possible range of low-end IoT devices.

A further and even greater challenge is posed by the global movement towards *post-quantum* cryptographic primitives: that is, algorithms which can be run on conventional computers, but which are expected to resist attacks from adversaries equipped with quantum computers. Such adversaries would easily overcome all currently-deployed public-key cryptography on the internet (for example, RSA signatures, which are the most widely-deployed signatures on the internet, could be efficiently forged using Shor's quantum factorization algorithm). Securing symmetric cryptosystems for a post-quantum world mostly implies increasing key sizes, which entails a moderate increase in computational footprint that might be challenging, yet far from prohibitive, for IoT systems. But securing public-key systems means completely replacing the underlying primitives, and most of the candidate algorithms under consideration by NIST have far heavier resource requirements than conventional public-key systems. Our goal here is to find, evaluate, and integrate the few public-key candidates capable of being run on low-end IoT devices; this represents a major challenge not just for the future security of IoT, but also the security of the wider internet.

3.1.2 Formally Verified Crypto Primitives

In the context of IoT, the devices are under the control of potential malicious users. Therefore, cryptographic primitives are particularly vulnerable to side-channel attacks, e.g. timing [73, 206], power analysis [207]. To protect the confidentiality of cryptographic keys, there exist software counter-measures mitigating side-channel attacks [45, 300]. For instance, the cryptographic constant-time programming discipline [45] mandates that conditionals and memory accesses to be independent from secret data. This discipline effectively mitigates cache attacks which are a class of timing attacks exploiting the fact that a cache miss is much slower than a cache hit. Another counter-measure is masking [300] where the secret is split into k+1 shares and the computation is performed in such a way that an attacker probing the memory and accessing k shares is unable to reconstruct a single bit of the secret. This countermeasure improves the robustness against power analyses.

To be effective, software countermeasures need to be preserved by the compilation process. However, existing optimising compilers only preserve functional properties and, therefore, there is no guarantee that security

countermeasures are still present in the binary [135].

Our objective is to design compiler algorithms which preserve security properties and, in particular, protections against side-channel attacks.

3.1.2.1 Semantic Modelling of Side-Channels

A first task consists in accurately modelling side-channels. The difficulty is that side-channels attacks exploit very low-level properties of the hardware. They may exploit details of the micro-architecture but also the spacial placement of transistors and the physical properties of the circuit. Therefore, it seems impracticable to obtain a precise model of the hardware. Moreover, an accurate model would only capture a single machine and therefore would not give a robust guarantee across the high variety of IoT devices.

We propose to explore an alternative approach where side-channels are modelled at a high level of abstraction. This approach, pioneered by Barthe et al. [37] is robust with respect to the implementation of the micro-architecture and versatile enough to accommodate a large range of side-channels. It consists in instrumenting the source and the assembly semantics of the compiler with a leakage function which models what the execution of the program leaks to an attacker. The objective is to model at assembly level what is leaked by the hardware and gives its counterpart at source level. In order to counter timing attacks, actually cache attacks, the leakage function formalises the ”constant-time” programming discipline and leaks to the attacker the result of conditional branches and memory addresses. The strength of the model is that it is robust with respect to the micro-architecture of the memory hierarchy and only makes the assumption that the cache content depends on the sequence of conditions and memory accesses which are leaked to the attacker.

Beside constant-time, we will explore other attacker and leakage models. Attacks based on leakage due to speculative execution have received a lot of attention. Here, a difficulty is to properly model speculative mechanisms at a high level of abstraction. There are other models of interest are ”probing models” [48] where the attacker is given access to a fixed number of observations across the program execution. These models are relevant to model security properties which ensure that sensitive information that is erased from the memory in a timely manner but also to reason about the security of countermeasures based on masking. One of the stronger model is the Hamming Weight Model [207] where the execution leaks at any time a function of the number of bits that are set to 1 by the program. Experimentally, it has been verified that power consumption is correlated to the Hamming Weight. A program secure in this model is therefore robust to power analyses.

3.1.2.2 Security of Compiler Optimisations

For a given model of attacker, we will revisit compiler optimisations and assess their security. It is well-known that optimisations may break the security of secure programs and make software counter-measures ineffective. For instance, compilers may introduce conditionals and thus defeat the purpose of the constant-time programming discipline. Reordering of computations may increase the lifetime of secrets and therefore make a seemingly secure source code vulnerable to probing models.

Our goal is to categorise compiler optimisations depending on their robustness to side-channel attacks. For optimisation that are not secure, we will work at designing a replacement that is secure while aiming for competitive performance. The outcome will be an effective secure compiler where all the compiler passes are secured. Representative passes are translation passes which decompose high-level source constructs (e.g. splitting of source expressions to 3-address code), dataflow optimisations (e.g. constant propagation, common expression elimination), register allocation, and generation of assembly code.

To be robust to certain side-channel, we anticipate that there may be compiler passes requiring some substantial redesign. For instance, a secure compiler in the Hamming Weight Model is a challenge because resource reuse e.g., overwriting a register, generates a leakage that needs to be preserved. In certain cases, it may be impossible to have a secure compiler achieving the exact same level of security. In that case, we intend to investigate probabilistic models where the advantage of an attacker observing the target program is negligible compared to an attacker observing the source program.

3.1.3 Embedded primitives for secure multi-tenant IoT software

Traditional embedded system approaches for modular software updates range from differential binary patches, to dynamic linking of binary modules. More recently alternative approaches have emerged, such as a small, updatable on-the-fly scripted runtime container [1], interpreted directly on low-end IoT devices. This paper

[1] E. Baccelli et al. ”Scripting over-the-air: Towards containers on low-end devices in the Internet of Things.” IEEE PerCom, 2018.

introduces a prototype of Javascript container on RIOT, enabling basic sensor, GPIO, timer, CoAP APIs within the container.

Indeed, recent progress has shrunk minimal resource requirements for embedded interpreters – for example JerryScript [2] and MicroPython [3]. In a first phase we will compare the tradeoffs (w.r.t. metrics such as speed, memory, energy budgets) offered by such a microcontainer approach using different APIs and script languages. Such approaches are promising because it both eases development of high-level logic on microcontrollers, and offers natively some isolation of the module's logic.

However, specific mechanisms are needed in order to harden such isolation so that it amounts to bona fide sandboxing of the software module. Sandboxing is desirable in cases where the module and the rest of the system are managed by different stakeholders. Here, by sandboxing a module, we mean characteristics such as limiting its ability to interact with hardware, limiting its ability to interact with OS (in particular, cannot crash of DoS the system), and denying access to private data (crypto keys, ...).

Sandboxing approaches can be broadly categorized as follows: (i) hardware-based, relying on an MPU, or on TEEs such as ARM TrustZone-M, RISC-V MultiZone etc. (ii) software-based, using some kind of virtual machine e.g. intepreted code in JS, Lua, MicroPython, or some alternative approach such as WebAssembly (iii) hybrid, using both a VM and protection in hardware via an MPU or a TEE, (iv) offline, using methods such as static analysism formal verification etc.

In this task we want to explore the potential of a software-based approach for safe execution of untrusted code while being as hardware independent as possible, and fitting low-end IoT devices resource constraints (memory in kB, small energy budget etc.) Using RIOT as base, we thus plan to both explore ways to guarantee strong isolation of micro-containers in javascript, micropython etc. and explore the potential of WebAssembly for microcontrollers [4] in this context.

3.2 Low-power & Secure Network

This section describes the planned work that concerns IoT network security. The planned work is strongly influenced by the past and future standardization activities within the IETF and gathers the design of new protocols 3.2.1, as well as the evaluation and security analysis of existing end-to-end solutions (3.2.2).

3.2.1 Zero-touch secure low-power network bootstrap

The last couple of years have witnessed a significant progress on secure communication protocols for the IoT. The IETF has taken steps in standardizing new solutions for protecting the communication channel, like OSCORE [158], TLS 1.3 [144] or EDHOC [327], and 3-party authorization protocols, like the ACE framework for constrained environments [326]. These new solutions have been demonstrated as much more efficient than their predecessors TLS 1.2 [346] or OAuth 2.0 [116] as used in the Web, and are expected to be deployed with the next generation of IoT products.

A common assumption for all of these solutions is that the trust relationship between the entities involved in the communication has already been established through common keying material (e.g. pre-shared keys, raw public keys, root trust certificates). At manufacturing time, the trust relationship is typically established between the IoT device and the manufacturer. The domain where the IoT device will be installed is not known at the manufacturing time, and before the IoT device can join a given domain, it needs to be provisioned with domain-specific credentials. Bootstrapping this trust relationship between the IoT device and the domain owner is a non-trivial task with the IoT devices lacking a user interface. Companies typically resort to out-of-band channels (e.g NFC, ad-hoc wireless network, pre-shared keys printed on the back of a device, serial port) or proximity-based authentication, requiring the user to go through a cumbersome process when installing a new IoT device. This opens up various vulnerabilities as the "bootstrapping" solution ends up being designed in-house, without a thorough review of the community and security experts.

As part of this task, we will leverage and complement the work done in the IETF LAKE standardization group [5] that is chartered to specify a lightweight authenticated key exchange protocol for IoT use cases. We will use the LAKE outputs to design and contribute to the standardization of a solution that allows an IoT device to join (mutually authenticate, authorize, be configured with domain-specific parameters) a network in a new domain, with zero pre-configuration of the IoT device required by the user. For a device to join a new domain without any user input, coordination between the manufacturer and the domain is needed and we plan on enabling

[2] E. Gavrin et al. "Ultra lightweight JavaScript engine for internet of things." ACM SIGPLAN, 2015.
[3] https://micropython.org/
[4] https://github.com/intel/wasm-micro-runtime
[5] https://datatracker.ietf.org/wg/lake/about/

it with our solution. A challenge is to make such a solution both flexible and efficient in terms of bandwidth consumption and code footprint. We will use the existing IETF work like EDHOC, OSCORE, Constrained Join Protocol (CoJP) [372] and BRSKI [287] as our starting point to achieve mutual authentication, message protection and parameter distribution during the execution of the protocol. We will define the missing pieces when these protocols are applied in the use case of zero-configuration network bootstrap and contribute to their standardization in the IETF.

3.2.2 Light-weight end-to-end security at transport layer and above

End-to-end encryption (E2EE) is a widely used communication protection approach in which only the end parties can read the messages; it can provide, depending on the protocol, confidentiality, integrity and authentication.

IoT devices are always in need of communicating with applications in order to provide the data required by the particular use case, so network communication is critical for IoT. Because of this criticality, data exchange between IoT devices or between an IoT device and an application server over the network must be made secure.

The main goal addressed by CNIT is: how to embed crypto primitives into end-to- end security protocols?

For concreteness the work has been focused on the integration of E2E on the Riot OS. Two options have been considered and are planned to be analyzed:

1. Transport Layer;
2. Application Layer.

In terms of approach, it is advisable to reuse already existing cryptographic protocols as much as possible. The main protocol used today in desktop applications is TLS (TLS 1.2 [346], TLS 1.3 [144] and the TLS UDP-based implementation DTLS 1.2 [145]), however given the context of this project and the nature of the devices where these protocols will run (IoT devices), these protocols result to be quite overwhelming not only because they require CPU power, but also for memory footprint (especially when asymmetric encryption is used) and network bandwidth consumption. Given the very strict requirements for a general IoT device, other solutions need to be analyzed; depending on the use case, a memory footprint of 100KB can be pretty big for a constrained device whereas in a desktop environment there are usually GBs of memory and this is not a problem.

CNIT addressed the issue of implementing (D)TLS above CoAP [156]. CoAP [405] is an application protocol designed for IoT communication, and “is a specialized web transfer protocol for use with constrained nodes and constrained (e.g., low-power, lossy) networks.”, meaning that it has features specifically designed for low overhead and low general complexity in order to minimize CPU, memory and network bandwidth consumption.

Being CoAP an application protocol, this implementation concerns the first option analyzed by CNIT, i.e., E2E security at the application layer. There are many advantages to using this configuration:

- IoT devices can leverage standard E2E security protocols (i.e. (D)TLS) to establish secure E2E encrypted connections;
- Many applications already leverage (D)TLS ;
- There is no need to invent a new cryptographic system (negotiation, authentication, ...);
- Automatically support new cipher suites by simply upgrading the cipher suites in TLS;
- Leverage a low-overhead application layer specifically made for IoT devices.

CNIT implemented DTLS 1.2 [145], TLS 1.2 [346] and TLS 1.3 [144] all with PSK configuration above the CoAP protocol, leveraging the WolfSSL [382] open source library. The memory footprint of WolfSSL with TLS 1.3 PSK was found to be around 100KB (tested on an ARM Cortex-M4 and an ARM Cortex-M0) while 60kB ca. of Flash is required for TLS1.2 PSK on these same platforms (samr21 and nrf52dk boards): this can be generally fine, but depending on the use case and especially on IoT devices, these numbers can be pretty big.

The other goal is the implementation of OSCORE [158] on Riot OS. OSCORE provides end-to-end protection between endpoints communicating using the CoAP protocol by protecting the CoAP requests and responses with the COSE protocol [193] thus providing “E2E encryption, integrity, replay protection, and binding of response to request”. It was designed for very constrained nodes and networks and it features small messages size and low code and memory requirements in addition to those required by CoAP.

The main advantage of this implementation would be that all the protocols are specifically designed for the IoT, while the main disadvantage is that communication with other nodes implementing a different protocol, e.g. (D)TLS, is not immediate and needs more work.

Benchmarks

Firstly a comparison among (D)TLS implementations is on the roadmap. The metrics are the following:

- memory footprint
- network traffic consumption
- CPU power
- Energy consumption

This last metric is quite interesting because of the impact of energy on IoT devices, the main actors in this task.

Then a broaden comparison with non-PSK configurations is on the roadmap, meaning measuring not only (D)TLS PSK with AES-128 but also asymmetric ciphers configurations, e.g:

- PSK with AES 256 CCM
- ECDSA-ECDHE with AES-128-CCM and P256r1
- ECDSA-ECDHE with AES-256-CCM and P512r1

The two last configurations are interesting in the sense that asymmetric encryption allows for key exchange during negotiation, meaning that it's not required to store pre-shared keys on the devices. There are pros and cons and a detailed comparison with the specified metrics will highlight the better configuration.

Switching to the transport layer, COSE/OSCORE approach also has to be evaluated and compared to (D)TLS implementation using the same metrics described before.

IoT implementation goals

(D)TLS implementation was found to require an actually big memory footprint for a constrained node. To address this issue CNIT aims for a compression of the TLS implementation in order to gain a minimal memory footprint; there exists an IETF draft called Compressed TLS (cTLS) [299] which aims in this direction: "(cTLS) is isomorphic to TLS 1.3 but saves space by trimming obsolete material, tighter encoding, and a template-based specialization technique. cTLS is not directly interoperable with TLS 1.3, but it should eventually be possible for a cTLS/TLS 1.3 server to exist and successfully interoperate.". Both cTLS and a custom tightening of (D)TLS will be taken into consideration, implemented and measured in accordance to the same metrics described before. The latter requires a very specific cut of the unnecessary TLS source code, accurate tightening of the components and a guarantee that the core of the TLS protocol will remain functional.

Security analysis

To implement (D)TLS above CoAP, the WolfSSL library was used. A forward-looking and very interesting work is a security analysis and vulnerability assessment of this library. Even if it is opensource there is no guarantee that the crypto primitives do not leave subtle vulnerabilities such as:

- Side channel vulnerabilities
- Padding oracle type vulnerabilities

Those vulnerabilities can lead to fully compromise the security of the communication. This security assessment would require two different analysis of this library, a black-box one and a white-box one. The first one aims to experimentally assess the security of the crypto functions assuming that the source code is not known while the latter will focus on source code review and static code analysis.

3.3 Securing the Supply Chain of OS Software

3.3.1 Automated security assessment & policies for IoT application update software binary bundles

3.3.1.1 Software Updates for IoT Devices

The IoT software update process is an essential operation for maintaining a suitable level of efficiency and security of IoT devices. Over the last few years, the research community has been working on the definition of several IoT update processes [208], among which the software update for resource-constrained devices is still an open research challenge [6]. Resource-constrained devices, as specified in RFC 7228 [63], use microcontrollers (like the Arm Cortex-M) on which they run a real-time operating system such as Contiki, FreeRTOS or RIOT [173], just to cite a few. To this aim, several firmware update solutions have been proposed in the last years, like FOSE [131], The Update Framework (TUF)[6], and Uptane [215]. However, most of the

[6] https://github.com/theupdateframework/tuf

proposed mechanisms are tied to specific operating systems or hardware architectures, and thus, they are not general-purpose.

To overcome such limitations, the Internet Engineering Task Force (IETF) is defining a standard for firmware updates called Software Updates for Internet of Things (SUIT) [188]. The main goals of SUIT are interoperability (w.r.t. the platform and the firmware distribution technology) and end-to-end security.

3.3.1.2 Security Issues in SUIT

The SUIT information model [189] defines a collection of security threats for the update process. As discussed in [407], such threats can be categorized into: (i) tampered firmware, (ii) firmware replay, (iii) offline device attack, (iv) firmware mismatch, (v) flash memory location mismatch, (vi) unexpected precursor image, (vii) reverse engineering, and (viii) resource exhaustion. Although the SUIT model suggests a set of security requirements and countermeasures, it is worth noticing that all these threats are related to the integrity and the confidentiality of the update process only, while the content of the update is inherently assumed as trusted. Therefore, the SUIT workflow allows an ISM to upload a firmware image containing security vulnerabilities or malicious behaviors. Furthermore, SUIT allows the ISM to transfer its authority to another entity, e.g., a third-party developer, that can deliver to the ISM some components of a software update (e.g., the executable of the application to be updated) or triggers the update process directly. In this case, the ISM has no mechanism to assess the content of the external software components, and must fully trust the external entity.

3.3.1.3 Automated Security Analysis of IoT Software Updates

To reduce the impact of unreliable updates, we argue that the SUIT update process needs to rely on a methodology to assess the security of the firmware image and in particular, of the IoT application. Such a methodology must be able to automatically evaluate the behavior of the firmware according to a set of security requirements, in order to allow the same ISM to deliver only validated and certified software updates. The security requirements can be defined directly by the same ISM, the IoT device manufacturer, or by a trusted third-party entity involved in the update process, like a Network Operator or a Device Operator, as defined in the SUIT standard.

We also argue that the methodology should work as a black box (i.e., without requiring the source code), in order to be systematically applied to any executable provided by third-parties. Finally, we argue that the analysis process must be carried out on the firmware image before it is submitted to the SUIT pipeline, in order to leverage the security mechanisms provided by SUIT to prevent any further modification of the image.

In order to mitigate the aforementioned security concerns, CINI proposes a novel verification solution called the **IoT Application Verification Framework (IoTAV)**. IoTAV allows to automatically evaluate the security of the IoT applications included in firmware images in a black-box fashion. In detail, IoTAV enables the definition of a set of security requirements codified as a *security policy*, that are then automatically evaluated on the application executable using state-of-the-art model checking techniques. IoTAV can be seamlessly included in the existing update pipeline, like the one defined in SUIT. IoTAV is able to detect malicious updates, thereby discarding those that doe n comply with the security policy and notifying the ISM, without affecting the normal operation in case of secure updates (solid arrows).

IoTAV can be adopted transparently by current IoT software updates workflows. Furthermore, CINI started the development of a working prototype for RIOT environments and tested the methodology on a set of actual RIOT OS applications. Experimental results indicated that the approach is viable in terms of both reliability and performance, leading to the identification of 26 security policy violations in 31 real-world RIOT applications.

During the project, CINI will continue the refinement of the IoTAV framework by i) completing the prototype for the RIOT ecosystem and ii) going through an extensive testing and integration phase against real-world scenarios.

3.3.2 Integrated prototype of secure IoT software update for RIOT

We plan to integrate the various security mechanisms (output from efforts described in this section) incrementally extending a prototype based on RIOT, which enables generic and secure software updates on a large variety of low-end IoT devices featuring microcontrollers. An initial version of this prototype has been published [7]. which complies with the SUIT draft specification [8] for authentication and integrity of RIOT firmware update,

[7] K. Zandberg et al. "Secure Firmware Updates for Constrained IoT Devices using Open Standards: A Reality Check," IEEE Access, 2019

[8] draft-ietf-suit-manifest-00

end-to-end, over a network which can contain low-power segments, as well as ”regular” Internet segments. Most recently, the corresponding code as upstreamed in RIOT master branch [9].

More in details, we plan to:

1. Integrate IoTAV in the prototype's update repository checks;
2. Influence next versions of SUIT specs & update implementation;
3. Integrate support for new (formally verified) IoT crypto primitives and new secure transport protocols, output of the work described in this section;
4. Extend the prototype with mechanisms to support and securing multi-tenant cases.

As much as possible, when applicable, we plan to upstream implementation to RIOT master branch, following workflows similar to what became the current SUIT support in RIOT [10].

[9] https://github.com/RIOT-OS/RIOT/tree/master/examples/suit_update
[10] https://github.com/RIOT-OS/RIOT/tree/master/examples/suit_update

Chapter 4 Secure Orchestration of the Intelligent Infrastructure

From the orchestration point of view, the Intelligent Infrastructure (II) architecture covers an extensive scale, from microcontrollers to the cloud, using remote web services and the web as the main platform. Thus, an II application is not only the code which runs on, e.g., IoT devices. All application components need to communicate using different protocols depending on the communication model: thing to application server, thing to human, or thing to thing communication. To program these applications today the developer needs to master a mix of technologies and be capable of ordering the reactive nature of heterogeneous devices, taking security and privacy into account. Several frameworks to simplify IoT development exist, however they are often not designed with security in mind and key security issues, such as key management, identity management and access control as addressed in an ad hoc way.

4.1 Security Orchestration Framework

4.1.1 Intelligent Infrastructure Lifecycle

IIs are systems of the highest complexity in terms of dimensions and variety of the involved technologies. In many cases, they consist of many subsystems that have been combined and integrated through several subsequent phases. These phases contribute to a continuous development cycle similar to that depicted in Figure 4.1.

The development process starts from a design phase. In theory, the design of a new II might start from scratch and include each part of the II to be constructed. In practice, however, the design can also be partial, i.e., only limited to a subsystem, and incremental, i.e., refining an already existing II. The collection of the various specifications produced during the design phase goes under the name of a *blueprint*. Among them, security specifications must be included and correctly integrated with all the other functional and non functional requirements. The validation phase has the role to highlight inconsistencies and errors in the blueprint. These checks include various verification mechanisms that can lead to a correction of the blueprint or even to reject it (thus going back to the design phase). When the blueprint is validated it can be used for the actual deployment of the II under construction. The actual deployment process changes with the type of II and may include instantiating virtual machines in a cloud infrastructure, installing various types of software and physically assembling hardware components. If the deployment is successful, the II is ready for testing. Although up and running, in this phase the infrastructure is not yet in production mode an is must be tested (either partially or entirely). Tests aim at spotting out flaws and violations of the blueprint specifications, including the security ones. If the test highlights no criticality, the infrastructure can pass to the production phase. In this phase the II runs and it is monitored to collect data, prevent and react to incidents. The collected knowledge improves the understanding of the II and contributes to the development of the present, as well as future, II.

4.1.2 Infrastructure Provisioning

In this section, we briefly recall the two infrastructure provisioning paradigms involved in our proposal.

4.1.2.1 Infrastructure-as-a-Service (IaaS)

IaaS [249] aims at providing a flexible and reconfigurable infrastructure development platform. In particular, an IaaS provider allows for a direct control over machines, operating systems, applications, and networking. By relying on virtualization technologies, IaaS platforms hide the underlying, physical infrastructure (a.k.a. bare-metal).

Each II consists of many different elements including hosts (e.g., servers and desktop clients), software (e.g., operating systems and applications) and network facilities (e.g., routers and firewalls). Conveniently, IaaS providers expose APIs for creating, deleting, and reconfiguring these elements. This makes IaaS a suitable paradigm for defining and deploying part of a II. In this setting, the building blocks of any theater are virtual machines (for computing and storage) and virtual switches, routers, networks, and network ports (for implementing the network infrastructure). Although some elements may not allow virtualization, a virtual network can also be connected with some physical resources outside the IaaS platform. For instance, an infrastructure can be connected to the Internet through a gateway.

Figure 4.2 represents the deployment of a fictional II on an IaaS provider. The virtual II is depicted on the top layer. Intuitively, all the elements are virtual with the only exception of the Internet which is only partially simulated. The real Internet is accessible through a gateway that directly connects to the external network. A

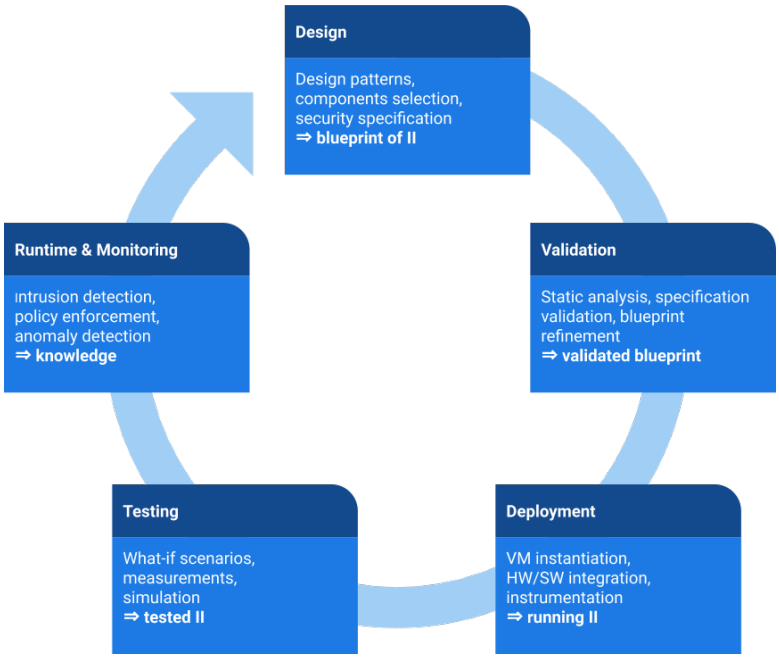


Figure 4.1: An abstract representation of the II lifecycle.

management network responsible for the cross-layer connectivity lays below. Such a network is necessary to support the *orchestration* and *monitoring* of the II.

On an IaaS provider, instantiating the II described above requires the following operations.

1. Create the virtual networks, e.g., Server and DMZ.
2. Create all the virtual machines, e.g., db and www.
3. Connect each virtual machine to the proper networks, e.g., db to Server.
4. Install all the operating systems and applications, e.g., DBMS on db.
5. Finalize the infrastructure by adding configurations, artifacts, and users.

All these operations are carried out by submitting the corresponding commands, e.g., via some APIs, to the IaaS provider. Nevertheless, as the complexity of the infrastructure increases, handling these design and deployment operations without a systematic approach quickly becomes cumbersome and error-prone.

4.1.2.2 Infrastructure-as-Code (IaC)

In the last years *Infrastructure-as-Code* (IaC) [23] emerged as the main infrastructure design approach. A IaC framework uses a specification language to model the desired infrastructure. A provisioning tool, namely the *orchestrator*, takes as input the specification and automatically deploys the infrastructure on an IaaS provider. We propose the following example to clarify the structure of a generic IaC specification language.

Figure 4.3 provides a class diagram representation of the infrastructure introduced above. The box at the bottom, labeled with *Primitive*, contains (some of) the primitive classes defined by a generic IaC provider. These classes abstractly define the building blocks of the infrastructure, e.g., machines and networks.

The *Network* class allows for the creation of virtual networks. Each virtual network is a collection of virtual subnetworks, i.e., the *Subnetwork* class. A virtual subnetwork is labeled with two properties, i.e., *address* and *netmask*, that specify the network address and the netmask of the subnetwork.

The *Compute* class represents a generic host, e.g., a virtual machine. An instance of Compute must declare its *image*, i.e., the installed OS, *flavor*, i.e., the hardware profile, and *init_script*, i.e., the instructions to correctly configure the host. There can also be dependencies between Compute objects. For instance, the www server depends on the db server. Typically, the orchestrator is responsible for resolving the existing dependencies, e.g., by creating the Compute objects in the right order.

Finally, Compute objects can be connected to one or more subnetworks. This behavior is modeled by the *Port* class that defines a generic network port for connecting to a subnetwork. Each port can also carry a (fixed) IP address specified in the *address* and *netmask* properties. In the diagram, the *db_Server_port* and

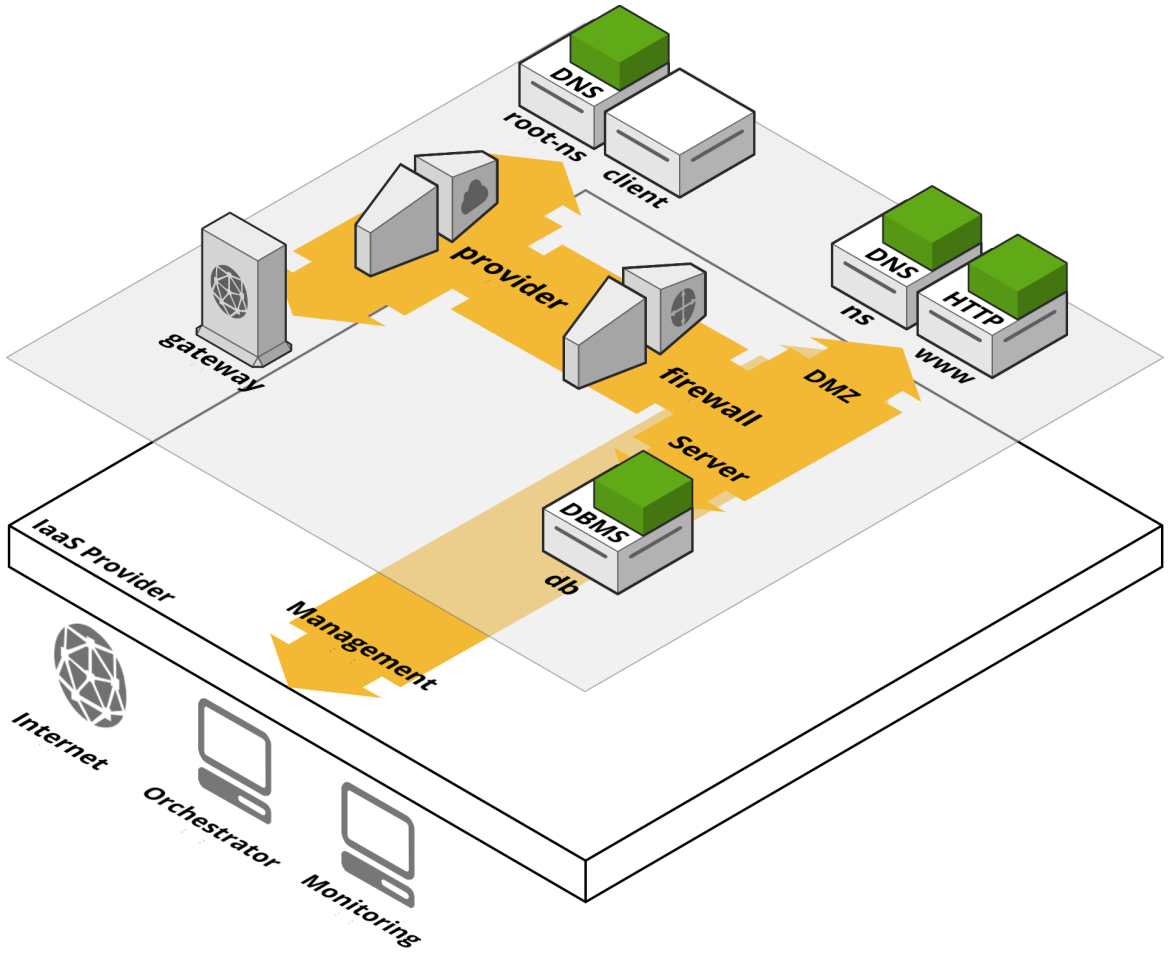


Figure 4.2: Layered view of a virtualized II.

www_DMZ_port are instances of the Port class and connect the db and www server to the two subnetworks with addresses 192.168.2.100 and 198.51.100.5, respectively.

4.1.3 TOSCA

The *Topology and Orchestration Specification for Cloud Applications* (TOSCA) [268] is a YAML-based[1] OASIS standard language for designing the topology and the life-cycle of a cloud application. A TOSCA-enabled IaaS provider must have a suitable TOSCA orchestrator.[2] TOSCA implements the concepts of Figure 4.3 by means of a rich type system. Briefly, the main constituents of TOSCA are the following.

Node types. They define an infrastructure component, e.g., a server or a network, or a component element, e.g., a software installed on a server. A node type can include *properties*, *attributes*, *capabilities* and *requirements*. Properties represent some static, node-specific feature, e.g., the hostname. Attributes resemble properties, but they are used to store a value that is set by the orchestrator after the instantiation, e.g., think of a dynamically assigned IP address. Requirements and capabilities define what the node needs and (optionally) provides to the others. Requirements and capabilities mainly serve as the joints for the relationships (see below).

Relationship and capability types.

They are used to connect nodes and, as it happens for node types, can include properties and attributes, e.g., the credentials for the authenticated service exposed by the node we are connecting. A relationship has a direction, and it connects the requirement of a source node to the capability of a target node. Moreover, each requirement can put a constraint on the types of both the target node and capability. For instance, a WordPress web application requires to connect to (i) a database (ii) endpoint, i.e., a network database

[1] http://yaml.org
[2] Existing TOSCA-enabled orchestrators often accept a slightly extended version of the TOSCA standard, i.e., a TOSCA dialect. If not differently stated, the examples in this paper refer to the ARIA TOSCA dialect [20].

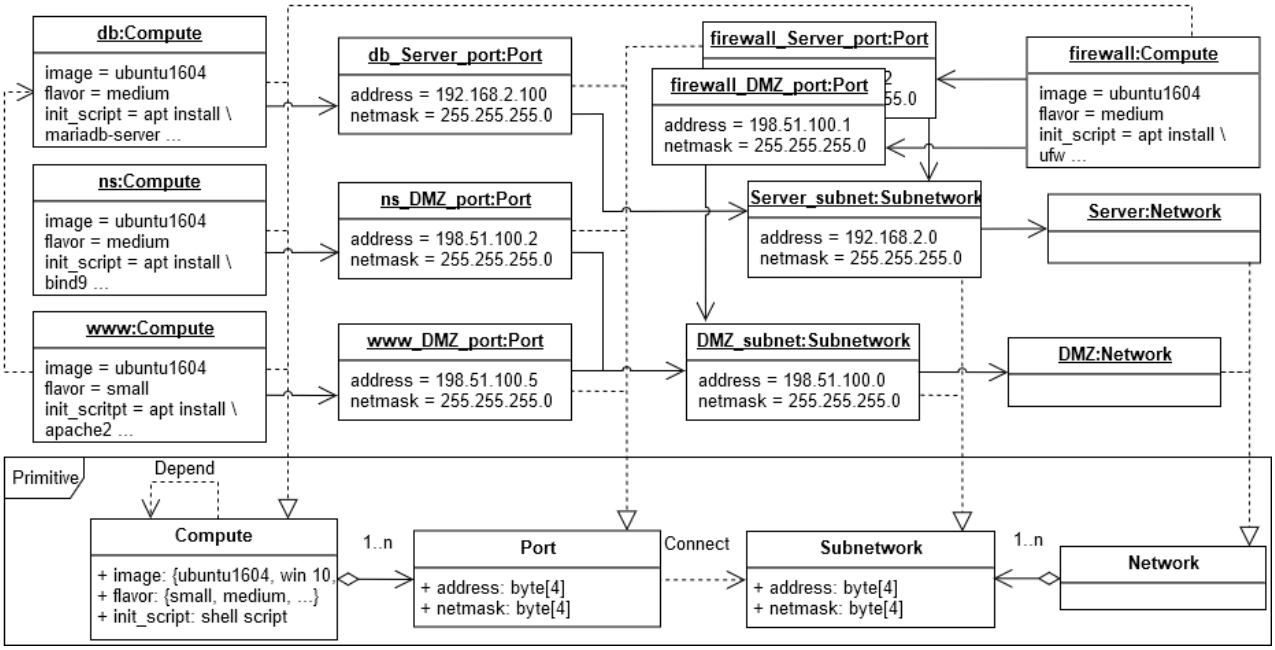


Figure 4.3: A generic IaC specification for the example II.

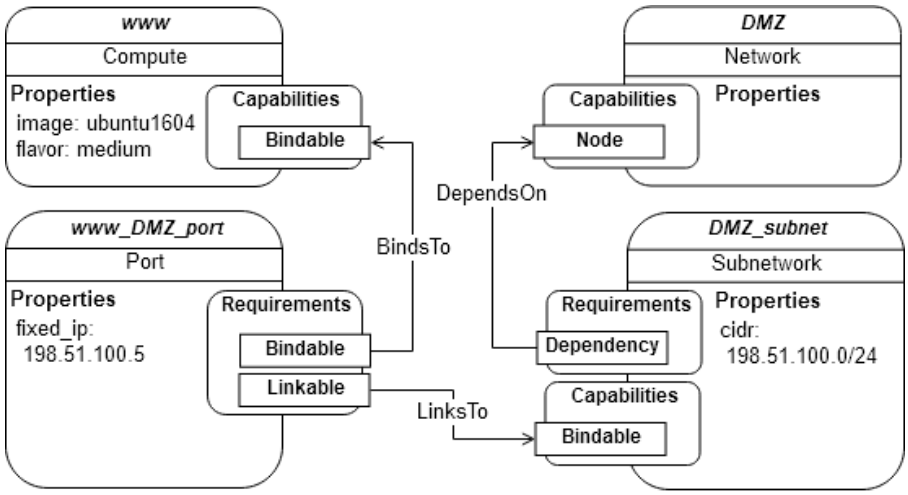


Figure 4.4: An excerpt from the (TOSCA-style) diagram for the virtual II.

(See [268] for more details). To model this, the WordPress node includes a requirement *database_endpoint*. The *database_endpoint* requirement constrains the type of the target node to be *Database* and the type of the target capability to be *Endpoint*. These two constraints capture (i) and (ii), respectively.

Interfaces. Nodes and relationships may have interfaces. An interface defines a custom operation to be invoked by the orchestrator. Two kinds of interfaces exist, i.e., *standard* and *on demand*. A standard interface defines a task related to the life-cycle phases of a node (e.g., create, start, and stop). For instance, one can add a standard, *create* interface to a compute node to ask the orchestrator for installing a certain software package when the node is created. Instead, on-demand interfaces introduce new tasks. The orchestrator permits to invoke the tasks through the definition of a new workflow. For instance, the on-demand interface can be used to implement an application-specific logic (see [313, § 7.3.2]).

A *node template* is a specification of a cloud application obtained through the composition of the elements mentioned above. In particular, each element is obtained by instantiating its base, namely *normative*, type. Roughly speaking, the TOSCA normative types provide a set of primitive classes (see Figure 4.3). Designers can define their own types by extending the normative types. As discussed in [70], the type inheritance enables some well-know mechanisms, e.g., type substitution and reuse, that simplify the design process.

To make a concrete example, consider the diagram depicted in Figure 4.4. It is an excerpt of the specification for the virtual II introduced in the previous section. In particular, it specifies the infrastructure of the web server (www) and the hosting network (DMZ). The www server runs on a virtual machine, an instance of the *Compute* node type. The hardware configuration of www and its operating system image are set by using the *flavor* and *image* properties, respectively. The DMZ network is an instance of the *Network* node type. A DMZ subnetwork, instance of *Subnetwork* node type, is in relationship, *DependsOn*, with the DMZ network and allows to specify its block of IP addresses in the *cidr* property. Moreover, the DMZ subnetwork provides the *Bindable* capability for supplying connections to the DMZ network. The www connectivity is represented by *www_DMZ_port*, an instance of the *Port* node type which also includes the *fixed_ip* property for assigning a fixed address to the connected node. The node *www_DMZ_port* is the source of two relationships, namely *BindsTo* and *LinksTo*. The former connects the *Bindable* requirement to the *Bindable* capability of the Compute node www. The latter connects the *Linkable* requirement to the *Bindable* capability of the *DMZ_subnet* Subnetwork node.

The syntax of the TOSCA language is YAML-based. Node instances are collections containing (i) the entity type, (ii) a key-value dictionary of properties, and (iii) a list of requirement bindings. A relationship between two nodes exists when a requirement of a source node instance is bound to the name of the target.

```
1  www:
2   type: Server
3   properties:
4    image: ubuntu1604
5    flavor: medium
6    requirements:
7     - port: www_DMZ_port
8  www_DMZ_port:
9   type: Port
10  properties:
11   fixed_ip: 198.51.100.5
12  requirements:
13   - network: DMZ
14   - subnet: DMZ_subnet
15 DMZ:
16  type: Network
17 DMZ_subnet:
18  type: Subnet
19  properties:
20   subnet:
21    cidr: 198.51.100.0/24
22  requirements:
23   - network: DMZ
```

Figure 4.5**:** An excerpt of a TOSCA specification.

Consider now the YAML specification given in Figure 4.5. It is the TOSCA encoding of the diagram of Figure 4.4. Node `www` (line 1) represents the compute entity for the web server. It is an instance of the `aria.openstack.nodes.Server` (line 2) type, i.e., a subtype of `tosca.nodes.Compute` denoting a virtual machine that runs on an OpenStack IaaS. [3] This node contains two properties: the name of the base operating system image (line 4) and the flavor (line 5) of the virtual machine. A port requirement (line 7) permits to establish a relationship with the `Port` node `www_DMZ_port` (line 8). The port assigns a fixed IP address to the virtual machine using the property *fixed_ip* (line 11). Also, the port is related with the `DMZ Network` node (line 13) and `DMZ_subnet Subnet` node (line 14). The IP addressing configuration of `DMZ_subnet` is specified in the `subnet` property (line 20).

4.2 Formal Verification of Protocols

In this sub-task security relevant parts of protocols are verified via formal verification.

The general goal is to consider applications of formal methods for verifying IoT protocols and additionally provide an overview of used methods. In order to select the most relevant protocols and to ensure their significance, scenarios in common smart building use cases should be selected. These scenarios should point to the emerging trends in IoT in every-day infrastructures, that are more and more exposed to different cyber threats, and that can affect everyday life to a significant extent.

IoT protocols selected from proposed scenarios will, in the following project stages, be analyzed and formally verified using selected methods. Protocols analysis will include the identification of security-critical parts for specific protocols. Based on this analysis the protocols will be modelled, formally described and finally verified. In case of a successful verification, no changes of the protocol are necessary. If a potential attack trace is found, the results should form the starting point for a suggestion to improve the respective protocol.

Initially, the selected scenario is a Smart Home environment, and it is part of the smart building use case – the common use case for $T6.3$.

The reason for choosing the Smart Home scenario, is the fact that the usage of IoT devices in Smart Home applications is getting more and more prominent, and will in future affect a large part of the population.

[3] For brevity we may omit namespaces such as `aria.openstack.nodes`.

4.2.1 Formal Methods

Formal methods are mathematical approaches to address different requirements, specification, and design level problems in software and hardware engineering. The most common application areas include safety-critical or security-critical components verification.

4.2.1.1 Formal Verification

Formal verification is the act of proving or disproving the correctness of an intended software/hardware component with respect to a certain formal specification or property, using formal methods. These methods can provide security guarantees by mathematically ascertain the correctness of designs. There is a diverse set of mathematical and logical methods that can be used for that purpose.

These methods are particularly useful in order to get quantitative statements about safety and security properties of digital systems [39, 203]. There are two basic types of formal method tools:

- **Model checkers** – they exhaustively and automatically verify a system's model in its model's state space with respect to a given specification
- **Theorem provers** – they often require human expertise to guide a proof of correctness by providing the design and specification characteristics as algebraic constraints or theorems [22, 203].

Although model checkers are usually more convenient to use and target to a specific problem domain and verification of properties in this field, they are also limited in the range of problems that they can handle.

Formal verification can be useful in a huge range of security testing fields to prevent attacks or minimize the attacking vectors. Possible checks usually include the verification or falsification of **security properties**, **functional correctness**, and the **proposed extensions** of protocol's specifications or implementations. Additionally, these methods can help to detect bugs or programming vulnerabilities. Even hardware Trojans can be detected with Formal Verification.

Functional checks

Include checks on functional correctness (if the specification fulfils the desired goals e.g. delivering data, correctness of algorithms, if processes are clearly specified in a protocol specification) but also a qualitative and quantitative analysis, including statements of performance of processes.

Security properties

Formal verification can also be applied to ensure the completeness of security goals, e.g. authentication, confidentiality, integrity, non-repudiation and encryption.

Proposed extensions

If an attack trace is found, usually an extension of the protocol is proposed. By addressing this, the new version of the protocol is formally verified.

4.2.1.2 Formal Verification Tools Overview

An overview of different tools used in literature for verifying protocols in Smart Home environment is given in text bellow. Related review studies (e.g. [22, 39, 184, 185, 281, 284]) also provide a general introduction to model checkers. Additionally, for better overview, an application and comparison of subset of them, including Scyther, Tamarin and ProVerif, applied to the LTE protocol, can be found in [176].

PRISM[4]

Probabilistic model checker developed by University of Birmingham for the quantitative analysis of system properties exhibiting stochastic behavior [181, 216, 217]. PRISM supports several probabilistic models and their extensions, such as Discrete-Time Markov Chains (DTMCs), Continuous-Time Markov Chains (CTMCs), Markov Decision Processes (MDPs), Probabilistic Automata (PAs) and Probabilistic Timed Automata (PTAs). It also includes engines for quantitative abstraction-refinement [202] and statistical model checking [177, 400]. PRISM has been used for *quantitative verification* in a wide spectrum of application domains, ranging from wireless communication protocols to quantum cryptography and systems biology[5] [217].

ProVerif[6]

A command line tool for automatic security analysis of cryptographic protocols based on a representation by Prolog rules [51, 52, 53, 54]. ProVerif, as input language, uses a typed variant of the pi calculus and is able to

[4] https://www.prismmodelchecker.org/
[5] https://www.prismmodelchecker.org/casestudies/
[6] https://prosecco.gforge.inria.fr/personal/bblanche/proverif/

proof reachability properties, correspondence assertions and observational equivalence. It is especially useful for the analysis of secrecy and authentication properties, and additionally properties as privacy traceability and verifiability.

Scyther[7]

A tool developed for the automated verification or falsification of security properties [112] and the user manual [111]. The tool was developed under the perfect cryptography assumption, i.e. it assumes that all cryptographic functions are perfect and the adversary learns nothing from an encrypted message unless he/she knows the decryption key. Therefore, the tool can be used to detect problems arising from the way the protocol was constructed by using the backward search algorithm based on a symbolic representation of sets of protocol runs.

Tamarin[8]

A model checker and theorem prover for symbolic modeling and analysis of security protocols [38, 248]. It generalizes the backwards search used by Scyther and enables: protocol specification by an expressive language based on multiset rewriting rules, property specification in a guarded fragment of first-order logic allowing quantification over messages and timepoints, and reasoning modulo equational theories.

AVISPA[9]

A tool suite, is a push-button tool for the automated validation of Internet security-sensitive protocols and applications [21, 350, 369]; AVISPA stands for Automated Validation of Internet Security Protocols and Application; The main advantage of the AVISPA tool suite is that different verification techniques can be performed on the same protocol specification.

UPPAAL[10]

A toolbox for verification of real-time systems developed by the Department of Information Technology at Uppsala University, Sweden in collaboration with the Department of Computer Science at Aalborg University in Denmark[43, 44]. The toolbox is especially useful for systems that can be modelled as a collection of non-deterministic processes with finite control structure and real-valued clocks, communicating through channels or shared variables. Therefore, typical application areas include real-time controllers and communication protocols, in which timing aspects are critical.

4.2.2 Smart Home Scenario

In Smart Home network applications, numerous IoT devices are used in order to improve every-day life, by using e.g. smart HVAC (Heating, Ventilation and Air-Conditioning), motion sensor and smart wearables. Different vendors nowadays try to agree on good practices in security implementation. Nevertheless, different connection types, the usage of wireless communication and systems set up by non-security experts allow attackers to intercept communication quite easily.

4.2.2.1 Description

Smart Home system can be used to make life easier by control heating, curtains, light systems, security systems, washing machine, television, stereo, etc. Even intelligent, private power supply systems, like solar power systems, with power storage and charger for electronic cars can be included.

The proposed scenario focuses on a system with most commonly used Smart Home devices – a simple Smart Home system with a smart HVAC, a motion sensor and smart wearables. The different devices are communicating via suitable protocols to the home server/hub. The home server is located in the home area network (or local area network) and communicates through a gateway to the Internet and outside users. Via smartphone or tablet the user can get access to the home network and can monitor or configure the system. In case of an unusual event the user will be notified immediately.

4.2.2.2 Cyber Attacks

In order to emphasize potential hazards and importance of security aspects in the Smart Home domain, this section lists some typical attacks in this domain.

Typical cyberattacks for Smart Home are the following:

[7] https://infsec.ethz.ch/research/software/scyther-proof.html
[8] https://infsec.ethz.ch/research/software/tamarin.html
[9] http://www.avispa-project.org/
[10] http://www.uppaal.org/

- *Eavesdropping*, i.e. spy on system.
- *Replay attack*, where (parts) of messages are recorded to use it at a later stage.
- *Man-in-the-middle attack*, where the communication between two communication partners is intercepted, and potentially changed during transmission (*modification attack*).
- *Denial-of-Service attack*.

In the first three attacks (private) data is exposed by the attacker, which is one of the huge risks in Smart Home applications. In case of a Denial-of-Service attack parts or the whole Smart Home system become unavailable.

4.2.2.3 Protocols

Widely used protocols in Smart Home applications are ZigBee, Z-Wave, EnOcean, Thread, KNX, LoRaWAN, Bluetooth, Bluetooth LE, Insteon, LTE and potentially 5G. Since wireless devices are more prominently used in Smart Home applications than wired devices, our research in Smart Home scenario, focuses on wireless protocols.

An overview of Smart Home protocols is given in Table 4.1.

	ZigBee	***Z-Wave***	**KNX**
Range	Local (<100m)	Local (<100m)	Local (<150m)
Data Rate	250 kbps	40–100 kbps	250 kbps
Spectrum	2.4 GHz	900 MHz unlicensed	868 MHz, 2.4 GHz
Power usage	Low	Low	Low
Standard	ZigBee spec., IEEE 802.15.4	ITU-T G.9959	EN 50090, ISO/IEC 14543-3
Alliance	ZigBee Alliance	Z-Wave Alliance	KNX Association
Year	2003	2003	1991

	EnOcean	**Bluetooth**	**Bluetooth LE**
Range	Local (<100m)	Local (<100m)	Local (<100m)
Data Rate	125 kbps	2 Mbps	1 Mbps
Spectrum	900 MHz, 2.4 GHz	2.4 GHz	2.4 GHz
Power usage	Low	Medium	Low
Standard	ISO/IEC 14543-3-10	IEEE 802.15.1	IEEE 802.15.1
Alliance	EnOcean Alliance	Bluetooth SIG	Bluetooth SIG
Year	2012	1999	2011

	LoRaWAN	**6LoWPAN**	**Thread**
Range	Metro (>10km)	Local (<100m)	Local (<100m)
Data Rate	50 kbps	250 kbps	250 kbps
Spectrum	900 MHz unlicensed	2.4 GHz	2.4 GHz
Power usage	Low	Low	Low
Standard	Proprietary	IETF/RFC 4944, IEEE 802.15.4	Thread spec., IEEE 802.15.4
Alliance	LoRa Alliance	6LoWPAN IETF WG	Thread Group
Year	2015	2007	2014

	LTE	**5G**	**Insteon**
Range	Metro (>30km)	Metro (>30km)	Local (<100m)
Data Rate	100 Mbps	10 Gbps	38 kbps
Spectrum	Licensed cellular	Licensed cellular	900 MHz unlicensed
Power usage	Band dependent	Band dependent	Low
Standard	3GPP Release 8 and 9	3GPP 5G	Proprietary
Alliance	GSMA – Cellular Carriers	3GPP ITU-R	Smartlabs
Year	2010	2018	2005

Table 4.1**:** Overview of Smart Home protocols

The initial Smart Home protocol priority list selected for SPARTA research in security-by-design (formal verification) aspect are:

1. EnOcean[11] [12]
2. Z-Wave[13]
3. ZigBee[14] [15]

4.3 Formal evidence language for the II

Today's ICT technologies such as Internet of Things (IoT), Industry 4.0, and Smart Grids, edge, cloud, etc., are enabling important new business capabilities and opportunities for organizations. However, such technologies are also straining organizations' ICT infrastructures to the breaking point as most organizations struggle with an inflexible, non-standardized and overloaded existing infrastructure. The business consequences of this situation are significant. Teams may not be able to collaborate effectively; security may be compromised; customer service and transaction times can suffer, which may lead to the loss of business. Given these challenges, organizations are actively moving towards an Intelligent Infrastructure (II), which is an ICT infrastructure that is more automated, service-oriented and intelligent. An intelligent infrastructure comes with enabling technologies to monitor, learn, predict, manage, optimize, protect and self-heal systems across data center, network, workplace, security and operations capabilities. The building blocks for developing and operating intelligent infrastructures are already known - standardization, consolidation, automation, virtualization and service orientation. The challenge, however, lies in putting those components together in an integrated whole and managing them securely, efficiently and effectively toward business goals.

4.3.1 Evidence-based security frameworks

An evidence-based approach for increasing the trust of systems has been successfully been applied in centralized settings for safety-critical systems, e.g., in the avionics and nuclear domains [16]. Indeed, the safety-cases developed by the centralized processes have made it possible for critical aircraft systems to be certified by certification authorities and for countries to allow their citizens to fly the aircrafts. Moreover, whenever there is an accident, processes to determine the root cause and its underlying evidence have maintained user's high trust levels on aircraft. For intelligent infrastructures, however, this evidence-based approach that has been successful to increase trust is not currently applied. This is due to the challenges described above related to the de-centralized and dynamic nature of II components and services.

4.3.2 Formal evidence language

In an intelligent infrastructure, it is common for its components to spread out and run, for example, on mobile edges rather than being deployed on a single platform. At the same time, services often move around with their consumers in mobile edge computing. In such highly dynamic service deployment scenario, it is important to maintain technology-agnostic formal security arguments for services supported by evidence. The security evidence required in one environment, e.g., highly unsecured environment, may not be the same as another environment, e.g., trusted secure environment. In addition, these security evidence descriptions must carry their associated security policies with them to be able to decide whether existing security evidence are usable and sufficient when upscaling or moving a service.

4.3.3 State of the art for evidence languages

There have been various approaches towards evidence-based security assurance for systems, such as intelligent infrastructure systems, with de-centralized and dynamic components and services. Arguably, the best-known framework is the PCC approach and its derivations, e.g., Proof Authorization Code, etc. In PCC, mobile codes would provide a formal proof for their security.

We provide below a brief overview of different initiatives and projects with the aim of building evidence-based security assurance.

- **MOBIUS** [36] was an initiative to build Proof Carrying Code Framework for mobile code [238]. The essential features of the MOBIUS security architecture include (1) innovative trust management: dispensing

[11] https://www.enocean-alliance.org/what-is-enocean/enocean-wireless-standard/
[12] https://www.iso.org/standard/59865.html
[13] https://www.itu.int/rec/T-REC-G.9959
[14] https://zigbee.org/download/standards-zigbee-specification/
[15] https://standards.ieee.org/standard/802_15_4-2015-Cor1-2018.html
[16] The purpose, scope and content of safety cases http://www.onr.org.uk/operational/tech_asst_guides/ns-tast-gd-051.pdf

with centralized trust entities, and allowing individual components to gain trust by providing verifiable certificates of their innocuousness; and (2) static enforcement mechanisms: sufficiently flexible to cover the wide range of security concerns arising in global computing, and sufficiently resource-aware and configurable to be applicable to the wide range of devices in global computers.

- **CYBER-TRUST** is a project aimed at building a Cyber-Security Intelligence framework for IoT systems [17]. The project focuses on the mitigating zero-day vulnerabilities and accomplish this by maintaining a vulnerability profile of IoT systems. The project is aimed at providing trusted transaction processing and coordination between IoT devices, ensuring security by identifying Data Communication Safeguarding critical files and software binaries, and minimizing the damage caused by tampered devices and malware as well as single points of failure collection and storing of forensic evidence.
- **DARPA-ARCOS** is an initiative by the American DARPA agency in the area of efficient evidence creation and maintenance for software re-certification [18]. According to its coordinator, the American Navy re-certification due to a change of single line of code takes an year and costs around 5 Million USD. The initiative's main goal is to advance the process of certification where a structured argument is built attesting the security risk of a system by building arguments by automating the evidence generation for new and legacy systems, so to scale the process of certification and reduce costs.
- **C3ISP** is an initiative proposing the increase of trust of ICT applications by sharing in a privacy preserving way data and analysis on data [19]. C3ISP's mission is to define a collaborative and confidential information sharing, analysis and protection framework as a service for cyber security management. The initiative also puts a greater emphasis on regulatory aspects, "compliance by design", proposing languages for security policy specifications.
- **FutureTrust** is a project proposing Open-Source mechanisms and models for trustworthy global transactions. Their main focus is on system inter-operability, electronic signatures to build certificates that can be used across different systems [20]. While this project focuses on cross-system certificates, rather than the actual evidence generated, its concepts can be applied to develop a cross-system evidence language.
- **LighTEST** is an initiative proposing to create a global cross-domain trust infrastructure where the trust of electronic transactions is delegated to trusted authorities [21]. The specification of trusted authorities and the delegation relation between authorities can be used to make distinctions between self generated security evidence structures and evidence structures from a delegated authority.
- **AF-Cyber** was an initiative proposing a logical framework for the specification of formal security arguments [22]. The core of this project is a logic-based framework for performing attribution of cyber attacks, based on forensics evidence and an intelligent methodology for dynamic evidence collection. The structure of such dynamic evidences will be helpful in developing an evidence language taking dynamic security aspects into considerations.

4.3.4 Semantic modelling aspects

One critical aspect to consider while developing such an evidence language for security assurance is the definition of a common language among the different involved stakeholders, both in terms of format and semantics. This is aimed at enabling the evidence language to specify security and accountability metrics of the services and components (both software and hardware) of a given intelligent infrastructure. In order to achieve this goal, the definition of a combined ontology that merges these currently separated services and components is a key enabler for an evidence language. This ontology will be focused in the creation and traceability of different evidences that will be used to implement accountability mechanisms throughout the entire intelligent infrastructure.

For cybersecurity, for instance, several ontologies and frameworks have been proposed [1, 262, 348]. One of the most extensive ones is the Unified Cybersecurity Ontology (UCO) [344], based on hierarchical classification of all the possible cybersecurity events by defining concepts such as means of attack, consequences, attacks themselves, attackers, attack patterns, exploit targets and other relevant indicators.

As corroborated by previously [1], these cybersecurity ontologies and in particular UCO represent an adequate starting point and provide enough flexibility to be completed with innovative metrics and indicators coming from different sources enabled by state-of-the-art technologies. In conclusion, an evidence language needs to

[17] CyberTrust https://cyber-trust.eu
[18] DARPA-ARCOS https://www.darpa.mil/program/automated-rapid-certification-of-software
[19] C3ISP https://c3isp.eu
[20] C3ISP https://cordis.europa.eu/project/rcn/202698/factsheet/de
[21] LighTEST https://www.lightest.eu
[22] AF-Cyber https://cordis.europa.eu/project/rcn/210306/factsheet/en

create a semantic model merging cybersecurity with domain-specific concepts to enable cross-domain accountability and auditing in the presence of de-centralisation and dynamism - defining features of an intelligent infrastructure.

4.3.5 Evidence and certificate formats

Security evidences from a given service or component needs to be abstracted and made light-weight to be exchanged among services and components. Therefore, development of an evidence language must also take into consideration the corresponding security certificate format into consideration. The communication of formal certificates has been investigated by the Mobius [36], ProofCert [23] and Dedukti [24] projects. In the former, certificate formats have been proposed to support the PCC framework. Therefore, their formats represented only evidence in the form of formal proofs. ProofCert proposed Foundational Proof Certificates based on solid substructural proof theory results and logic programming. Dedukti proposed formats for theorem provers, such as HOL, Coq, by splitting the deduction content of formal proofs from the computation part.

An evidence language needs to build on both ProofCert and Dedukti: Firstly, its certificates needs to increase the trust of systems by a wider range of types of evidence, and not only formal proofs. This can be done by advancing the mechanisms for checking the validity of certificates. Secondly, an evidence language needs to validate the communication of certificate in intelligent infrastructures by advancing the integration of certificates with technologies such as Internet of Things (IoT), DLT, Industry 4.0, and Smart Grids, edge, cloud, etc.

4.4 Multi-layered security model for Fog computing

4.4.1 The overview of Fog computing paradigm

Fog computing is a new paradigm of Cloud computing which aims to address some specific issues which cannot be solved using traditional Cloud computing architectures. The OpenFog Consortium established in 2015 defines Fog computing architecture as “a horizontal, system-level architecture that distributes computing, storage, control and networking functions closer to the users along a cloud-to-thing continuum” [272]. Fog computing architecture consists of 3 layers: a) Cloud; b) Fog; and c) Edge devices. The Edge layer would include all low power data collecting nodes like sensors, actuators, as well as smart devices in cases when they are used to capture data rather than to process it. The middle Fog layer is made of local medium power devices which collect and process data, make local decisions, controls actuators in the Edge layer and act as data gateway to the Cloud services. The top layer belongs to a Cloud with powerful servers in remote large scale data centers.

Bonomi et al. [61] lists the specific requirements of an IoT solution which are best addressed by using the Fog computing architecture:

1. Edge of the network placement, low latency communications and location awareness. The Fog computing is capable to provide good solution in cases where low latency between data collection and solution is required. This architecture also works in cases of Internet connection failures and Cloud based services outages.
2. Wide-spread geographical distribution. It opposes the idea of Cloud Computing which is very centralized. Fog Computing services and applications are widely distributed over the network.
3. Mobility. It is essential for a number of Fog applications and services to have a direct connection with mobile devices. Mobility techniques such the LISP protocol are required for this.
4. Very large number of nodes. A large number of the Edge nodes is available due to generous geographical distribution.
5. Predominant role of wireless access. It enhances the mobility.
6. Strong presence of streaming and real time applications. Real-time communication is prioritized in the Fog Computing over the data batch processing.
7. Heterogeneity. Fog nodes are very different by their form which are used in different environments.

Multi-Layered Fog computing architecture and its main properties by layers is summarized in Fig. 4.6 on the following page.

Data life cycle in Fog Computing architecture starts with data acquisition inside the Edge node layer. The data is transferred to the upper Fog layer to be stored and/or processed. Fog nodes are also able to send feedback

[23] ProofCert https://team.inria.fr/parsifal/proofcert/

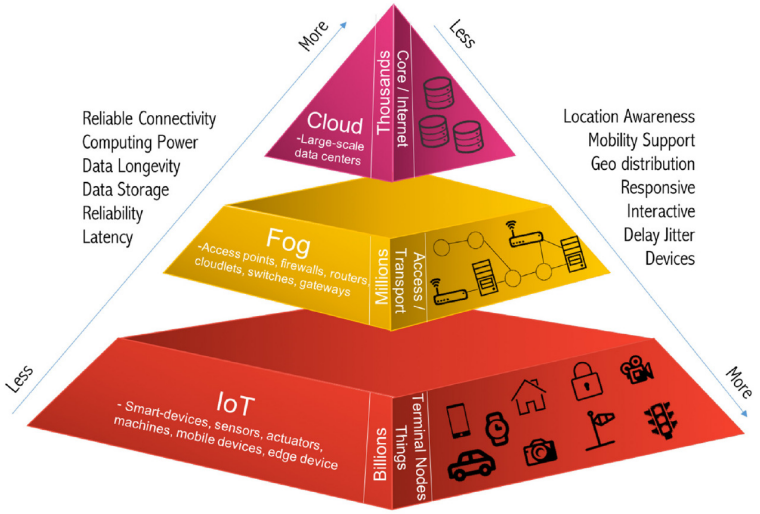


Figure 4.6: Properties distribution in the Multi-layered Fog Computing architecture [401]

data and control commands back to the Edge layer. If more resource intensive data processing is required the Fog node devices send data to the Cloud based services.

The main application areas of the Fog architecture according to the [397] are:

1. Smart Home. The development of Internet of Things leads to situations where home owners use lots of home automation devices and sensors connected to their home network. These devices are often produced by different manufactures and it is hard to make them compatible with each other. Fog Computing allows one to ingrate all these devices into one platform. It can provide 1) a unified interface to integrate different devices; 2) adaptive computation power and storage; 3) real-time data processing at low-latency.
2. Smart Grid. Smart meters which are integrated in different locations are meant to measure the readings in a real-time manner. The centralized hub collects and analyses any status related data. It also sends its commands to stabilize the power grid as a response to the change of power demand. Fog Computing can help providers to break the complex global grid into micro-grids. This will contribute to scalability, lower expenses, better security, and lower latency. Switch to Fog computing architecture could also help to integrate renewable power generators as wind turbines, solar panels, etc. In this case the Fog Computing layer would be responsible for the micro-grid and communications with surrounding Fog devices.
3. Smart Vehicles. Fog Computing architecture could be used for vehicular networks. The Fog Computing nodes could be installed by independent entities alongside the roads. These should communicate with each other and interact with passing vehicles. Other Fog computing application areas for smart vehicles include traffic light control, parking management, traffic data sharing, etc.
4. Health data Management. Health related data contains priceless and private information. Fog Computing might enable the patients to have their sensitive health data processed locally. Data can be uploaded to a Fog node as a smartphone or local network connected PC. If a patient needs help in a medical laboratory or a doctor's room, the preprocessed data could be securely transmitted to cloud based storage.

4.4.2 Main issues of the Fog computing

The Fog computing paradigm solves several shortcomings of cloud computing by providing solution in the situations where low latency and jitter, mobility support, constrained devices support, context and proximity awareness is required. On the other hand Fog computing also introduces some new challenges in various fields, including data privacy and security.

Distributed service infrastructure in the Fog layer may be owned by different entities and using different software and hardware which requires effective collaboration and communication solutions. The new standards which define how different components of the architecture can communicate are still under development. Fog layer services are often implemented as virtual machines, so solutions for lightweight virtual machine lifecycle management (creation, deployment, migration, context preservation, etc.) are also very important. Resources are residing in different locations distributed over various devices, so the effective mechanism for service discovery and orchestration is required. Another big issue is the mobility of both the Edge nodes and the Fog

nodes, which causes the need for specific protocols and solutions which could enable the synchronization of various states of services devices on the heterogeneous infrastructure [304] hosting the Fog services.

Security assuring functions must be installed in every Fog computing based solution to make it responsive, available, survivable and trusted [244]. All edge paradigms, including Fog computing, uses lots of different building blocks (such as various wireless communication protocols, constrained devices and networks, distributed and peer-to-peer systems, virtualization platforms, etc.) to comprise the final solution. It is essential to protect not only the building blocks themselves, but also to orchestrate the diverse security mechanisms [304]. One needs to have the full, unified and transversal view of all security mechanism available in heterogeneous infrastructure in order to ensure effective and secure integration and interoperability. Moreover by ensuring the security of all building block individually we do not necessary ensure the security of the solution as the whole.

Additionally Fog computing paradigm introduces specific requirements e.g. the security methods and protocols should not be centralized as central infrastructure may not be available due to the strict requirements for the latency or simply be offline due to malicious attacks. On the other hand some Fog Edge devices may be constrained [63] and support only the simplest authentication protocols and limited connectivity. One not only needs to ensure sufficient security in all building blocks based on different architectures and technologies but also to ensure secure global connectivity and accessibility in a heterogeneous ecosystem [303].

According to [290] security problems specific to Fog computing could be grouped into three parts by the place of occurrence in the overall Fog architecture: security threats in Edge or sensing layer, security threats in network infrastructure and security threats in Fog layer. The remaining part of Fog architecture, the Fog to Cloud part, doesn't introduce any Fog specific challenges.

4.4.2.1 Security threats in the Edge layer

Edge layer uses various sensing technologies forming wireless sensor and actuators networks and different communication means, such as wireless networks, radio-frequency identification, near-field communications, etc. [355]. Security threats in the Edge layer include:

Edge node capture and tampering. Edge sensors and actuators may be captured and analyzed, important end user data may be extracted. Authentication keys may be captured thus compromising security of the whole solution.

Spoofing attack. Attacker may send malicious data to the Fog layer by pretending to be the legitimate sensors.

Signal jamming. Attacker may generate strong signal causing the interference in the wireless communications of the devices.

Malicious node and data. Attacker may add malicious node to the sensors network and generate malicious data thus causing undesirable behavior of the whole system.

Denial of service (DoS) attack. Attacker may flood the sensor nodes with lots of malicious packets. This attack may exhausts batteries of the constrained devices and utilize all the network resources thus causing the reduced responsibility and availability of the whole solution.

Node outage. Some of the nodes are cut down causing degraded performance of the whole solution.

Replay attack. Original data packets may be captured and resent pretending to be the new data. If the authentication of the data sources is not strong enough this attack may cause undesirable behavior of the whole system.

Existing solutions for these challenges include: Mutual authentication of Edge and Fog nodes and authorization of the data; usage of strong cryptography and steganography for data encryption and integrity protection; spread spectrum communications; jamming report generations; usage of error correction codes and collision detection.

4.4.2.2 Security threats in Network Infrastructure

Network infrastructure is used to transmit data from Edge layer to Fog layer, various local area network technologies and physical transfer media, such as wired or wireless networks, Wi-Fi, Bluetooth low energy, Zigbee, etc. may be used here [254]. Classical security triplet of Confidentiality, Integrity and Availability (CIA) is at risk at this layer. Different technologies introduce different security challenges:

Selective forwarding. Some selected packet may be dropped by malicious software causing degradation of QoS of the whole system.

Black hole and Wormhole attacks. Malicious routing information may be inserted into network, causing selected packets to be routed to the sink hole and dropped or collected and stored in different storage location for further analysis.

Various flooding attacks (Hello Flood, Acknowledge Flooding, etc.) may be used to flood the network channel with malicious data causing network congestions.

Heterogeneity and scalability of network infrastructure caused by migrating nodes and numerous different network technologies and security protocols introduces difficulties in coordination of all transfers making the solution more vulnerable.

Possible solutions for these issues are: usage of standard TLS/DTLS, IPSec security protocols where possible; installing firewalls and intrusion detection systems; link layer encryption; multipath routing; strong identity verification and continuous data packet authentication; secure decentralized authentication information management.

4.4.2.3 Security threats in the Fog layer

In the Fog layer data is collected, stored and processed. Thus causing security problems related to data integrity and confidentiality in storage and during processing. The service provisioning and availability is also very important. Fog Nodes usually are managed remotely which enables owners to control numerous devices in an efficient way at the same time enabling the attackers to launch various network-based attacks. Fog layer devices frequently have classical wide spread security problems often encountered in web applications, web services and related technologies.

Multi-tenancy related issues. Frequently Fog Nodes support multi-tenancy, where the same service may serve multiple user groups (tenants). This requires strict isolation of virtual machines, runtime environments and/or data pools related with different user groups.

Sniffers, Loggers, Phishing Attacks. These attacks may be used trying to capture passwords, logins, keys and other important information from sessions of user interaction with Fog devices.

Injection attacks. Attacker may use these techniques to cause malicious code execution and/or loss of data on the Fog nodes.

Session hijacking. Attacker may try to steal sensitive information from user trying to use her identity to gain access to personal information and get administrative rights to Fog nodes.

Physical damage and tampering. Unlike cloud servers Fog nodes sometimes are physically exposed and vulnerable to physical attacks. Even if the device itself is secured, some not wireless input/output interfaces may become the targets of hardware tampering and eavesdropping.

Data security. Fog nodes collect, store, process and send aggregated results and/or commands to cloud layer services or actuators in the Edge layer. Data, metadata and software must be protected in Fog nodes during all three states of its life: data in use (while it is processed residing in operative memory), data in rest (stored on non-volatile media), and data in motion (while being transferred through the network infrastructure).

The already existing solutions for these challenges include: safe programming and testing during the development of the software; usage of antivirus software on the end users computers; extensive data verification on the Fog node; effective access control, identification and authentication; extensive session inspection; secure data encryption during storage and network transfer.

4.4.3 Existing solutions to Fog computing related issues

One of the duties of the Fog Computing is to connect its components, but managing such complex, heterogeneous and constantly changing network, ensuring the connectivity of its components and providing relevant services is not easy task. The authors of [398] propose to use emerging concepts such as Software-Defined Networking (SDN) and Network Function Virtualization (NFV) which can help to create a flexible and easy-to-maintain network. These technologies might help Fog Computing to improve its network scalability at the reduced costs.

The authors of [338] propose to use SDN concept and implement it with physically centralized control with Fog devices acting as the centralized controllers. In such scenario each node on the Fog layer is expected to work as a router for other closely located Fog nodes. Each Fog node also has to support the mobility of End nodes. SDN integration in such solutions causes issues related to the ability to cope with mobility and potentially unreliable wireless connection.

NFV technology replaces the network functions by virtual machines. The key element of this concept is virtualization. The virtual machines are dynamically created, removed and moved between Fog nodes as required. The advantages of NFV in the Fog Computing is the ability to virtualize relevant gateways, network switches, firewalls and intrusion detection devices. The newly introduced issues are the throughput and latency of virtualized devices and the proper placement of the virtualized devices in the dynamic constantly changing network.

The framework based usage of SDN and Fog Computing architecture as solution for issues related to Vehicular Ad-hoc Networks (VANETs) is proposed in [204]. The authors claim that the upcoming 5G network will help to cope with different communication requirements and addition of centralized and flexible approach of the SDN and Cloud-RAN (CRAN) will help to increase flexibility and implement effective resource allocation and distribution techniques.

A model-driven framework proposed by authors of [366] is designed to develop a Personalized Health Monitoring system. The purpose of such system is to integrate different medical instruments in order to ensure an uninterruptable remote monitoring of the patient health while using an internet communication. The model-driven framework has a multi-layered structure with feature-based modeling and feature model transformations positioned at the top. Meanwhile the application software generation is located at the bottom. Intermediate levels serve to narrow down any design options in order to be able to apply the model transformations. These transformations enable creation of customized models which are required in order to come up with the implementation layer.

Service orchestration is another option for Fog Computing. The paper [69] proposes a orchestration solution based on Docker Containers. There is number of container orchestration solutions currently available like Kubernetes, Mesos Marathon, Docker Swarm as the authors admit but they are not flexible enough. Their solution adds two basic components: Fog Orchestrator (FO) and Fog Orchestration Agent (FOA) to the Fog layer nodes to adapt traditional Cloud services orchestration solutions to the needs of Fog computing.

4.4.4 Fog orchestration solutions

Fog computing paradigm extends cloud computing and inherits several important properties such as resource orchestration, multi-tenancy, elastic provisioning, etc. [197]. In such implementations Fog nodes use their computational resources to provide required services to the Edge nodes. If the orchestration framework is designed correctly it allows to partition required services between different Fog nodes and/or some Cloud based services in such way, that delay-sensitive tasks are executed by physically close Fog nodes and resource hungry non real-time tasks are executed by cheap and powerful Cloud services. This sort of affinity-aware software offloading is shown in Fig. 4.7.

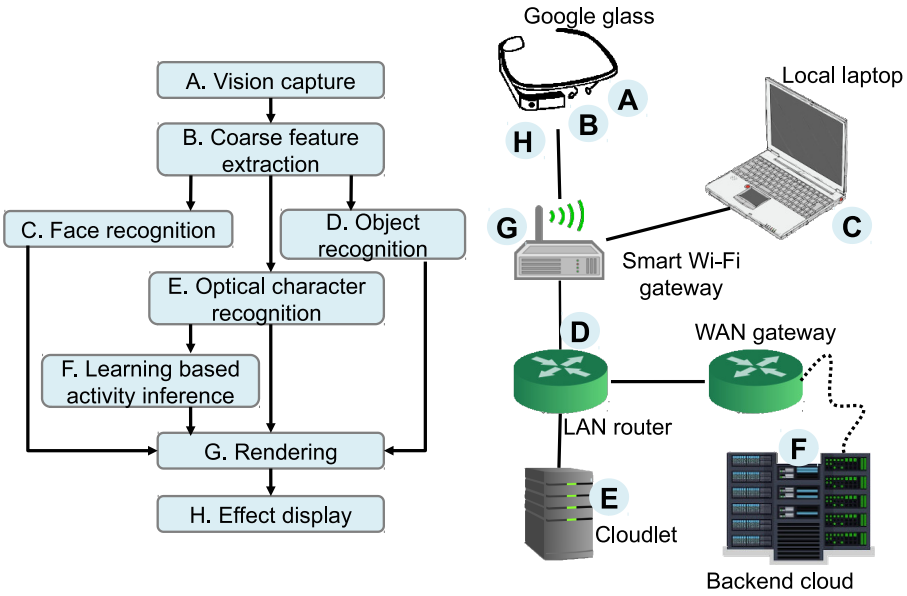
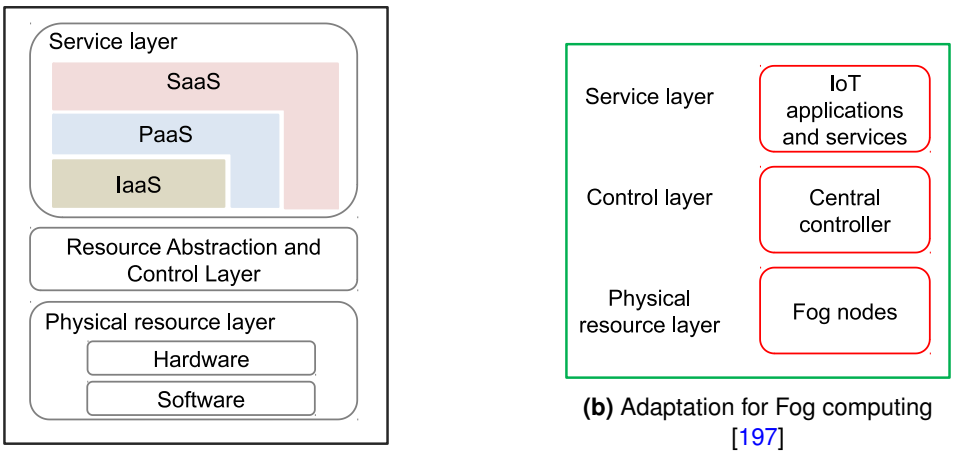


Figure 4.7: Fog and Cloud service partitioning example

Authors of [197] provide an overview of the Fog services orchestration framework which bridges the gap between the infrastructure and the applications. They propose to use classic architecture introduced by National Institute of Standards and Technology (NIST) [231] and already widely adopted in modern cloud computing systems, such as OpenStack. This three layer Cloud computing orchestration framework (see Fig. 4.8 on the following page) could also be successfully used in Fog computing.

The central role in this three layer architecture plays the control layer or central controller which makes decisions how to efficiently allocate available IoT applications and services to the lower layer of Fog nodes. Several new challenges specific only to Fog computing paradigm are needed to be addressed: scalability of the control layer, affinity based service provision, and heterogeneous nature of the Edge and Fog nodes.



(a) Cloud computing [231]

(b) Adaptation for Fog computing [197]

Figure 4.8**:** Orchestration framework architecture

The control layer of Fog computing must be able to coordinate interactions between massive number of Fog nodes with very different capabilities and constraints forming the physical resource layer. Two methods for dealing with scalability of control layer are used in Cloud computing solutions: hierarchical controller approach (e. g. OpenStack Cells) and the flat controller approach. The hierarchical controller layer uses additional higher level controllers forming the multi-level structure which helps to minimize computational requirement for controller layer. The flat controller layer uses peer-to-peer architecture with periodic gossiping between different controllers trying to collect information on global structure of the system and making decisions for local device management. For the Fog computing systems where Fog nodes are owned by different independent parties (home owners, public institutions, universities, etc.) with heterogeneous hardware and software resources the flat controller approach is more suitable. On the other hand the Fog layer may include millions of different nodes and controllers may not be able to store and exchange the information about the whole system. To solve scalability problem authors of [197] propose that each controller should collect and maintain information on the structure of the system in its close proximity and exchange information only with its neighboring controllers. Such solution ensures that required Fog services are naturally offloaded into physically close Fog nodes and the amount of information needed to maintain and exchange is largely reduced. Fig. 4.9 on the next page shows three layer Fog orchestration framework and data flows required for effective orchestration. Three types of data flows or data exchange interfaces may be distinguished:

The southbound interface connects Fog nodes with orchestration controller and is used to collect hardware information and to offload required services into corresponding Fog nodes for physical execution.

The westbound interface is mainly used for communications between neighboring orchestration controllers in peer-to-peer manner for information collection about current state of physically closely located Fog nodes and exchange of data on services' requirements. This interface helps the controllers to form and maintain the aggregated picture of their Fog and Edge devices in their close proximity and to rapidly change the picture in events when Fog or Edge nodes moves to different locations. Broadcasting information to its neighbors and using peer-to-peer communications prevails in this kind of communications.

The northbound interface is used for access and management of actual IoT applications and services

Wen et al. [379] discusses the possible architectures of controller or Fog orchestrator. The main Fog specific constraint and requirements for Fog orchestrator are: Scale and Complexity of the Fog layer; Security criticality; Inherent properties of Dynamicity of Fog and Edge layers; the need for Fault diagnosis and Tolerance. The authors propose architecture of Fog Orchestrator summarized in Fig. 4.10 on the following page. The orchestrator consists of three main elements. The planning element selects appropriate services and places them into corresponding Fog nodes; Execution monitoring element monitors the system during runtime assuring the required level of QoS and security. After considerable changes in system (due to dynamic nature of Fog computing elements) required data is collected and Optimization element finds the new best scenario of service deployment in Fog nodes. Parallel genetic algorithm is used to solve optimization problems which arise in the planning and optimization phases.

Brito et al. [69] present another architecture of Fog orchestrator and show what modifications of Fog nodes are necessary for efficient interaction with Fog orchestrator (see Fig. 4.11 on page 51).

The special Fog Orchestration Agent is added into each Fog node and is responsible for communications with Fog Orchestrator as well as monitoring resources of local Fog node, efficient resource management, security

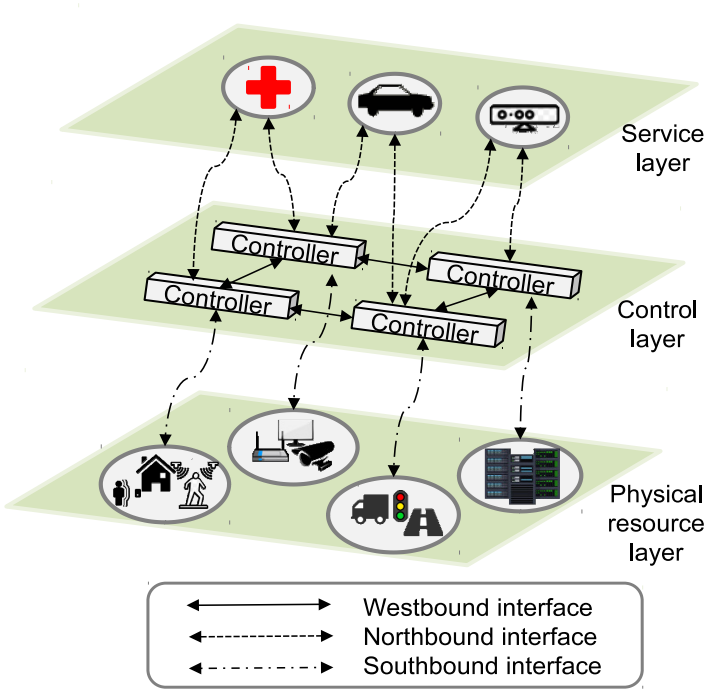


Figure 4.9: Fog computing orchestration architecture with flat control layer [231].

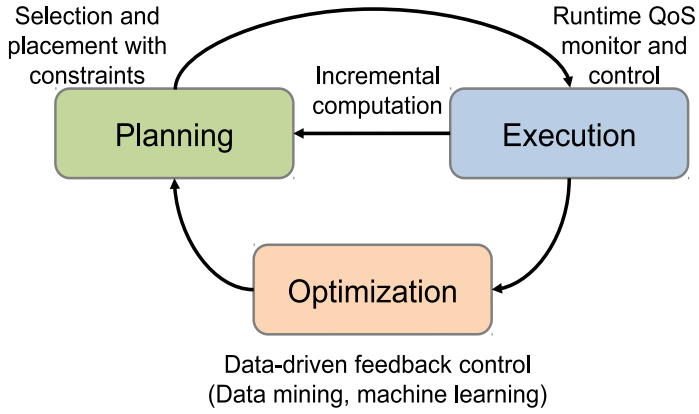


Figure 4.10: Fog orchestrator architecture [379].

and QoS assurance. The authors use Docker Swarm for virtualization and Open MTC M2M Framework for communications.

Velasquez et al. in [365] summarizes the challenges specific to Fog computing paradigm and analyzes four already proposed and implemented Fog orchestration solutions: Supporting the Orchestration of Resilient and Trustworthy Fog Services (SORTS) by Velasquez et al. [364]; The Service Orchestrator Architecture for Fog enable Infrastructure (SOAFI) proposed by Brito et al. [69]; The reference architecture for Mobile Edge Computing (ETSIGSMEC) introduced by the ETSI Industry Specification Group; and a Cloud-based architecture for next-generation cellular systems (CONCERT) by Liu et al. [232].

4.4.5 Security orchestration challenges in Fog computing

Fog nodes are computational nodes which talk with different sensors and actuators and provide required services for local data filtering, processing and control as well as act as intermediaries for communications with remote cloud based services. Frequently these Fog services are deployed dynamically into corresponding Fog devices as required. For example, if Edge device requires “serviceX”, which is currently provided by Fog node “FogA”, and moves to the close proximity of Fog node “FogB”, the whole Fog infrastructure must adjust itself in such a way that “FogB” node starts to provide “serviceX” to the Edge node. This kind of Fog infrastructure behavior is assured by using Fog services orchestration techniques discussed in previous chapter. Usually Fog service orchestration is implemented by customizing and adapting already existing Cloud

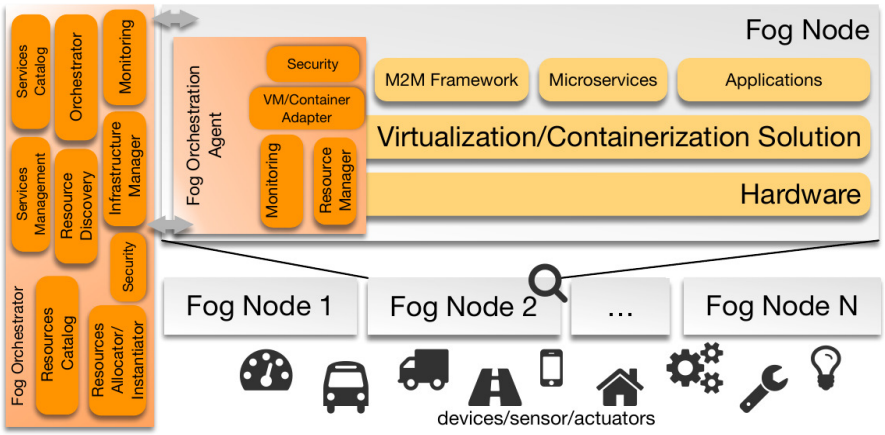


Figure 4.11: Fog orchestrator interaction with Fog node [69].

services orchestration solutions [69, 197]. One of very important issues in such solutions is the QoS assurance and security of the whole system. Fig. 4.12 summarizes different challenges which have to be addressed while designing security orchestration solution for Fog nodes.

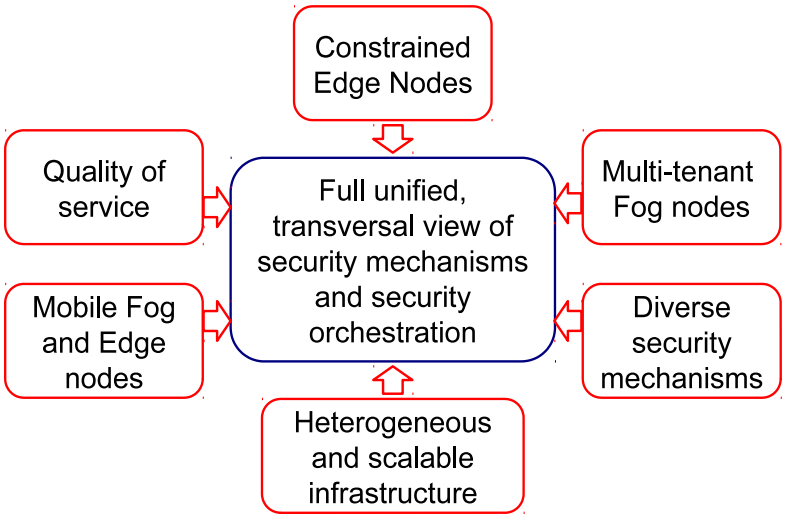


Figure 4.12: Security orchestration challenges in Fog computing.

Mobile Edge nodes may change location and approach different Fog node which has different architecture and computational or communication capabilities. Some Edge nodes may be constrained devices with strict constrains on available security protocols and/or communications methods. The Fog nodes may also be mobile, and approach different Edge nodes with sibling Fog nodes providing services for them. In all cases the service provision should be optimally divided by all Fog nodes in near proximity of the Edge nodes. On the other hand the requirements of QoS i. e. latency, communication bandwidth, data security, etc. must be preserved.

This continuously changing situation requires effective orchestration of services in Fog nodes, as well as security and QoS assurance. Essentially after the each significant change in the local situation in close proximity of the each Fog node the optimization problem must be solved in order to provide best possible service for the Edge devices.

The constraint categories for such optimization problem are the following:

- Security level requirements. Different Edge devices may collect data of different importance and adequate data protection must be provided while transferring, storing and processing.
- Edge node physical constraints. If the Edge node is constrained it may not be able to use some of the security and communication protocols due to the limitations in available processing and/or transmitting power.
- Communication bandwidth requirements. There is the essential difference in bandwidth requirements between Edge nodes which provide continuous data streaming and one time sensors/actuators (e.g. real-time video streaming vs. temperature sensor which sends several bytes of data every minute).

- Communication protocol requirements. Some Edge devices may need one way connectionless broadcasting and others may require strict two way communication with acknowledgements. Some solutions may work using client/server architecture, and the others may communicate peer to peer using mesh networks.
- Environment requirements [200]. Some Edge nodes may have short range high bandwidth (i. e. WiFi, ZigBee, BLE, etc.) communication hardware requiring closer proximity to the Fog node, while the others may have long range low bandwidth communication hardware (i. e. LoRa).

While implementing orchestrator which is able to collect all security and QoS related information from neighboring Fog and Edge nodes (thus forming unified view of security and QoS requirements), and in most efficient way distribute the required services between available Fog and Cloud resources (i. e. solving the problem of optimal service partitioning, see Fig. 4.7 on page 48) one has to address the following considerations:

- The flat controller (orchestrator) approach is more suitable due to the constantly changing Fog and Edge layers.
- The peer-to-peer architecture with periodic gossiping is most suitable for the Fog services orchestrators.
- To solve the scalability problem each controller should collect and maintain information on the structure of the infrastructure only in its close proximity and communicate only with its neighboring controllers.
- Southbound interface (see Fig. 4.9 on page 50) communications protocol for exchanging security orchestration specific information between orchestrators and Edge nodes must be developed.
- Westbound interface (see Fig. 4.9 on page 50) communications protocol for peer-to-peer gossiping between neighboring orchestrators must be developed.
- Methodology for solving QoS and security related optimization problem of orchestration of mobile, heterogeneous and scalable Fog layer must be proposed.
- Continuous monitoring of the situation during runtime must be assured by the orchestators and after the considerable changes in the system the new optimal (QoS and security vice) partitioning scheme of the services must be found and deployed (see Fig. 4.10 on page 50).

4.5 Key Components of an Intelligent Infrastructure

Intelligent Infrastructures are capturing data, analyse that data and, as a result, are invoking an autonomic response to decisions that have been taken on the findings [286]. In that process, several targets for privacy and security issues exist and have been addressed by existing Intelligent Infrastructures.

This chapter is discussing properties of existing intelligent infrastructure frameworks and tries to identify those key components (functions and services) that are needed to ensure the fulfilment of specific needs, esp. in the context of intelligent infrastructure orchestration. A special focus is also taken on the management of privacy and security in such an infrastructure, which includes a look at the security principles that are relevant as well as some security and privacy-specific key components.

4.5.1 Components of an Intelligent Infrastructure

Intelligent infrastructures heavily base on the use of IoT devices that provide data and which are the target for the automatic responses. As a consequence, the identification of the key components of intelligent infrastructures and their impact on privacy and security mainly is equivalent with the identification of key components of IoT frameworks and IoT platforms. In the recent years a generally accepted technology architecture for such IoT frameworks and platforms has been crystallised. This architecture with all layers and relevant components is shown in Figure 4.13 on page 54 [190, 191, 320, 321, 322, 351]. The overview already provides the main security principles that are relevant regarding the different layers of the entire chain between IoT device and IoT application. The following table 4.2 on the following page gives an overview about the purpose of each component in the figure, as explained in [190] and [270].

From a security point of view, the entire set of used technologies defines the attack surface of the IoT architecture. Security principles that try to reduce the number of threats and vulnerabilities focus on the securing of the devices, the securing of the communication between the involved systems, the cloud infrastructure itself and everything that is related with the lifecyle management of each system. IoT devices that are deployed in an uncontrolled environment require a protection against physical attacks as well as attacks against the integrity the operating system that is running on the device. IoT devices that are communicating directly with the platform as well as gateway devices require a protection of the communication link or end-to-end security. The IoT platform itself requires knowledge of each controlled device and therefore requires secure device identification or authentication. Confidentiality of the stored data is another important principle in the framework which not

Table 4.2: Software components of Intelligent Infrastructures

COMPONENT	FUNCTIONALITY
Hardware: IoT Device	
Device Operating System	Managing the computer hardware and software resources of the device Provides common services
Communication: Gateway Device & Network Connection	
Sensor Handler	Connects or interfaces with sensors and actuators Aggregates data
Management Agent	Handles manageability primitives for gateways, sensors and actors (provisioning, error handling, alerting, eventing)
Data Agent	Gathers and formats data from different sensors Controls actuators based on commands from the cloud
Analytics Agent	Learns actionable data in local context Near real time
Security Agent	Handles security primitives for gateways, sensors and actors (authentication keys, certificates)
Software: IoT & Cloud Platform	
Network Management Traffic Management	Manages and optimise the flow of traffic between the involved parties and applications Interacts with the management agent of the network device
Device Management	Configures and controls manageable primitives of the device Device discovery
Configuration Management	Ensures on-premise configuration management including devices and security
Resource Management	Resource and workload optimisation Provisioning and deployment of servers Starting and stopping of servers. Acquisition and assignment of storage capacity Creation of virtual machines.
Service Orchestration Software	Ensures service level agreements across resource managers
Processing and Events/Action Management	Rule engine that allows for actions base on incoming sensor and device data
Data Ingestion Software Data Collection and Storage	Ingests and stores data coming from the devices Makes the data available to other cloud software Interacts with the data agent of the devices
Analytics Software	Big data analysis on the data gathered from the devices Machine learning Interacts with the analytics agent of the devices
Presentation and Visualisation	Graphical depiction of sensor data
Identity Management	Manages the identities of the devices Manage the access to sensitive and non-sensitive data
Security Management	Configures and controls security primitives of on-premise equipment Interacts with the security agent of the devices
Application: IoT & Cloud Application	
Business System Integration	Connecting the IoT/Cloud solution into a single larger system that functions as one
Reporting	Presentation of data in an understandable way
Alerting	React on findings in the given data with an alert

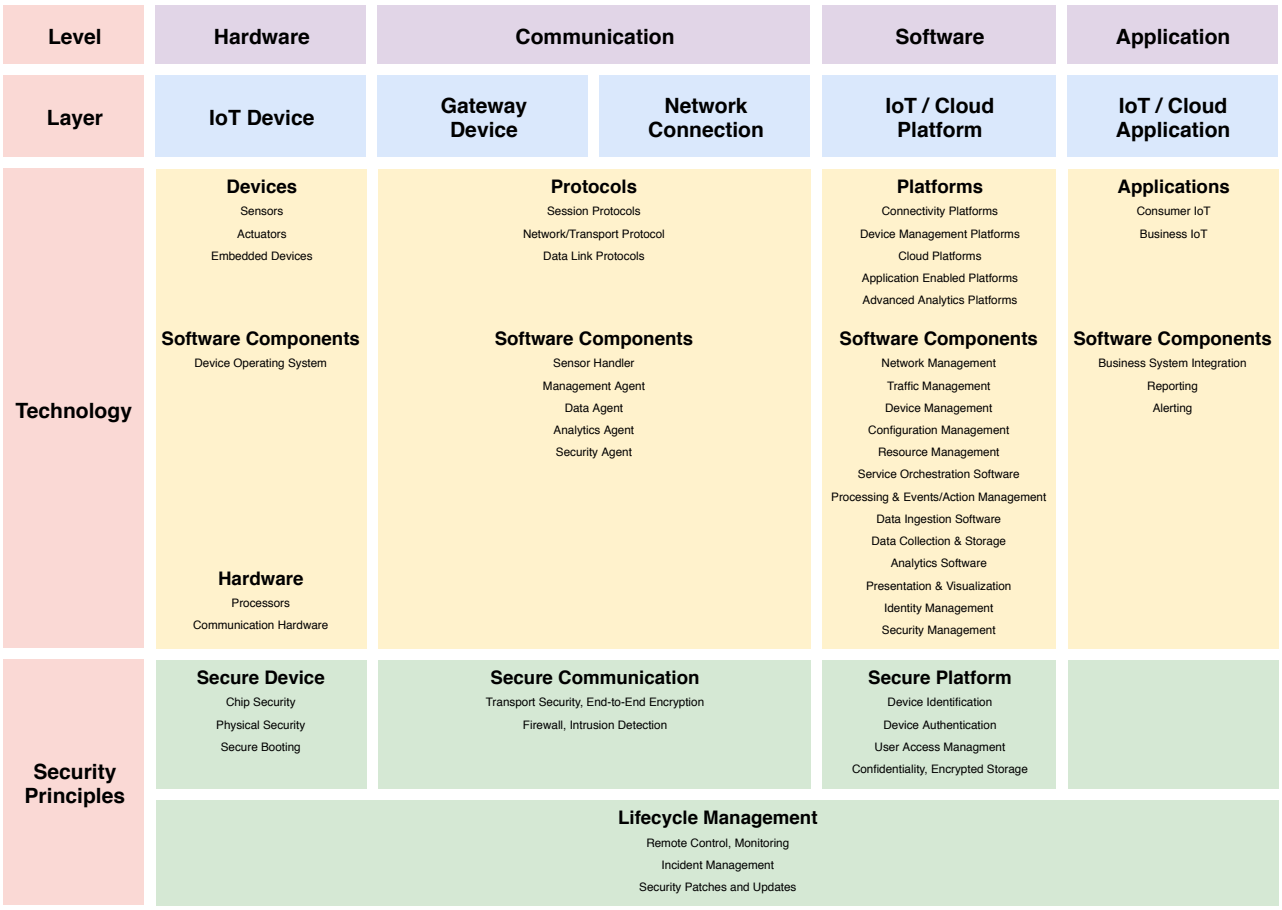


Figure 4.13: IoT technology architecture

only requires encrypted storage but also the management of access rights of users. Above all, it must be ensured that all involved systems and devices face regular security updates and patches. The entire framework must be controlled and monitored to identify and react on incidents.

4.5.2 Key Components Classification

There are several ways to name key components from all shown components, depending on the context that is discussed.

Minimal

In a minimal setup, at least one component of each layer or level must be considered. Beside the essential devices and communication protocols that are needed to connect the devices to the cloud, some components are additionally important: First, the device management is crucial for a successful deployment of all devices. Because sensor devices produce data, the data collection and storage component is essential. The processing and event management component is needed to provides event and time-based automation. The analytics component is needed to extract information from the collected data, while the presentation and visualisation components helps to display the data in a user friendly way.

Orchestration

From an intelligent infrastructure orchestration point of view, key components are foreseen to automate the management, the coordination and the organisation of the entire IoT architecture. These components manage the following elements: network; traffic; devices; configurations; resources; identities; security.

Security

In the described architecture, some components or elements are of relevance in regards to security. First, the device operating system as well as all used communication protocols. In the platform, the cloud security management component and its security agent counterpart play an important role in the entire security architecture. Linked to that is the identity management to ensure identification and authentication of devices and users, as well as the management of access rights, which is linked to the storage and collection of the data.

Chapter 5 Resilience-by-design of Intelligent Infrastructures (II)

Emerging Critical Information Infrastructures (CII), supporting critical assets, are using IoT devices increasingly, and massively. In these cases, both the complexity, increasing vulnerability to faults and attacks, and the relevance, making targeted attacks probable, call for defenses complementing the standard, industry-practice paradigms oriented to intrusion prevention and detection, defeating extreme adversary power and sustaining perpetual and unattended operation. Furthermore, under the scope of resilience design, intrusion detection can provide crucial components to resilient-by-design approaches in order to discover and handle unauthorized actions of adversaries, e.g., to identify and attenuate the communication between the affected elements of a system and the adversarial domain. In Task 6.4, "resilience-by-design of intelligent infrastructures (II)", we intend to investigate mechanisms that allow building resilience in systems, giving some steps further in achieving security-by-design, in areas meeting, especially for CII: threats, uncertainty, real-time needs, etc. Stronger, dynamic and automated paradigms for systems are required, and we will develop innovative protocols creating the last line of defense in case intruders manage to bypass classical security measures. Namely, by leveraging existing Byzantine fault and intrusion tolerance techniques, and self-healing and diversification mechanisms. The rest of this chapter is organized as follows. Section 5.1 discuses the intrusion detection approaches. Resilience techniques based on fault and intrusion tolerance is given in Section 5.2. The chapter is concluded in Section 5.3.

5.1 Intrusion detection

The proliferation of communication and commerce over networking architectures has put reliable cyber security solutions in high demand. Cyber security issues concerning individuals or organizations can develop into disastrous scenarios, like losing critical information, promoting relentless attacks (on critical infrastructure as well), or contributing to a distributed denial of service (DDoS) attack [81, 375].

The security of any given system comes in a direct proportion to the decisions both the organisations and the individuals make with regards to security requirements and their relative prioritization. Cyber-attacks have been on the rise for the past few years due to a slew of factors. Many research and policy changes have been undertaken to control this phenomenon. However, a clear path to a safe and secure cyber-physical system is not yet apparent. To understand the motivations behind the cyber-physical attacks, it might be necessary to underpin those attacks as societal events associated with economic, cultural and organizational factors. These factors seem to be key to understanding the *how* and *why* behind the security decision processes.

Intrusion detection is one of the components of a global security strategy. It is the second line of defence against attacks. A complex information system supported by a large-scale network composed of multiple heterogeneous devices is rarely completely secure and free of security flaws. Despite all the upstream efforts that are made during the design and development of a critical information system, malicious activities can most likely succeed and compromise the confidentiality, availability, or integrity of the system during its life cycle. An Intrusion Detection System (IDS) aims at detecting such attacks against a computer system and network. To deal with latent threats, an IDS monitors continuously the running system and analyses the gathered information to detect if an attack occurs or not. When the monitoring mechanism suspects that an attack has occurred (or is in progress), an alert is raised. As for any detector, the provided quality of service can be analyzed by considering the two types of error that can occur, namely false positives (no attack is conducted but an alert is triggered by the intrusion detection system) and false negatives (an attack takes place but it is not detected by the intrusion detection system). When the rate of false negative increases, the usefulness of the detection tool decreases. Worst, when the number of false positives is too high, the many irrelevant alerts make its use harmful.

The analysis performed by an IDS is based on local data provided by a set of independent probes deployed in different places of the monitored distributed system. Each of these observation devices is responsible for monitoring a small, well-defined part of the entire system and reporting what is going on. Depending on whether a probe monitors the activity of a particular machine or the activity at a particular point in the communication network, IDSes are classified in three main classes, namely HIDS (Host based Intrusion Detection System), NIDS (Network based Intrusion Detection System), and Hybrid IDS. Whatever the observed targets (computation devices or communication links), the reported information either corresponds to a raw observation of an activity (i.e., an occurrence of an event) or is the result of a first low level analysis that identifies a suspicious local behavior (i.e., a low level alert). The locally generated information is usually stored in logs and journals. It is also sent in messages to an analysis tool that centralizes the data and carries out a deeper global analysis: this activity is a key feature of a Security Information and Event Management (SIEM). Note that this two-level analysis (local and global) can be replaced by a more complex hierarchical structure dedicated to data collec-

tion and analysis. In any case, the gathered information is ordered according to the occurence date of each element and thus a unique flow (or stream) of timestamped data (events and low level alerts) is created. While combining outputs from different sources/probes, a pre-processing step is often performed to homogenize, correct, and reduce the flow of elements [361]. Indeed, since the probes are heterogeneous, the information they provide in their messages has to be restructured and standardized using a unique description language. At the same time, the collected information can be enriched (for example, the location of the probe is added if this information is missing). Additional controls allow also to discard some erroneous low level alerts (for example, knowledge databases can be used to assert if the exploit of an attacker which has been suspected by a probe is possible or not). Reducing the number of elements contained in the flow is usually achieved by merging similar elements: on the one hand, the same action is sometimes detected by several different probes and, on the other hand, a probe can generate several elements in the flow that are related to the same action. During this first correlation phase, the goal is to identify elements of the flow that occurred at close dates and have a strong relationship since they concern the same action (malicious or not). All the treatments described above aim to produce a cleaned and reduced flow of events and low level alerts. The main difficulties come from the fact that any used solution must be adapted to the specificities and weaknesses of the deployed observation tools and that it must be implemented efficiently to cope with a very large amount of data (in terms of elements per second).

In the context of SPARTA, the activities on intrusion detection conducted by the different partners focus on the analysis of the resulting flow that will be carried out on a single machine (i.e. after all the previously described steps). The lowest common denominator of cyber-attack detection are adversarial identification and machine learning methods: the detection is as weak (or strong), as weak are the data processing approaches. Three detection approaches are covered and discussed separately, even if one of the proposed challenges is to combine them: signature-based approach (cf. Section 5.1.1), anomaly-based approach (cf. Section 5.1.2), and control-theoretic intrusion detection (cf. Section 5.1.3).

Note that some challenges are independent of the approaches. In particular, an important one is related to the availability of an appropriate dataset which is critical in the development of an Intrusion Detection System (IDS). A bulk of state-of-the-art research does not provide reliable performance results since they rely on either the KDD99 or NSL-KDD benchmark datasets, which is concocted of traffic that is over 20 years old, hence it does not represent recent attack scenarios and traffic behaviours. Obtaining traffic from simulated environments can help overcome this issue when merged with testing more recent datasets, such as the CICIDS 2017 [62]. Published datasets are available for different domains, such as industrial control systems (ICS) [180]. The observed data that will be provided to us in the context of Sparta will complete the currently limited panel of available datasets.

5.1.1 Signature-based intrusion detection

A signature-based approach relies on the apriori knowledge of some possible attacks. Patterns of malicious traffic/activity are compared to current samples, and if a match is found an alarm is raised (cf. Figure 5.1).

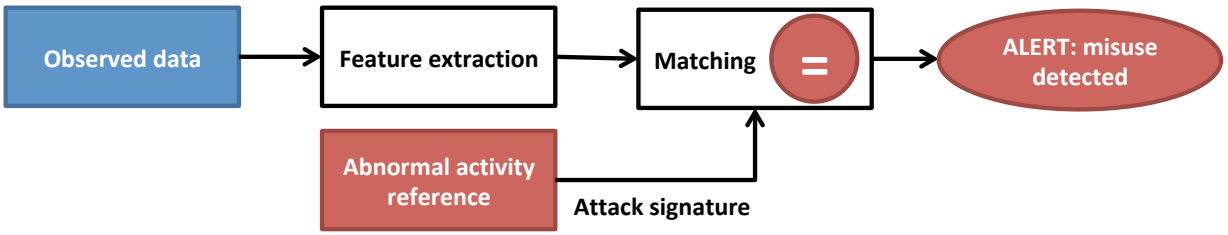


Figure 5.1: Signature-based Approach for Cyber-attack Detection

This approach is already applied at the level of an observation device to generate low level alerts but it can also be applied a second time during the centralized analysis to take into account multi-steps attacks. In this latter case, the detection of correlations between elements of the incoming flow aims to generate high-level alerts. Here the targeted attacks are not those that can be reduced to a single action performed by the attacker on a unique device. During a multi-steps attack, an attacker will perform a series of malicious actions not necessarily all on the same device and not necessarily in any order [86]. An attack can be conducted quickly or spread over a very long period. A complex multi-steps attack can be described using various description techniques. For example, when the description relies on an attack tree [333], the leaves of the tree correspond to the actions that have to be executed by the attacker and the internal nodes of the tree correspond to operators (AND, OR, SEQ, ...) that identify the logical and temporal constraints between these actions. From the description of an attack called a signature (i.e., the point of view of the attacker), it is possible to determine how an instance

of this attack will be observed if it impacts some specific nodes of the system (i.e., the point of view of the defender). To be able to identify both the possible targets of the attacker and the corresponding reactions of the associated probes, knowledge about the network topology (location of devices, communication links, ...), about the cartography (types of device, operating systems, installed software, ...), and about observation tools (nature of the probes, locations, configurations, ...) is needed [259]. Different instances of a same attack may target different parts of the system that are not necessarily monitored by the same kind of probes. Consequently, each attack is associated to several patterns (and not just a single one) that have to be searched within the flow of observations. To assert that an attack occurs, one of its well-identified patterns must appear in the flow of events and low level alerts. Correlation rules aim at identifying all the patterns that have to be search within the flow: each pattern is a description of the possible reactions of the observers to a specific instance of the attack [252, 357]. A correlation engine is an analysis tool that takes as input the flow of observations and a list of specified correlation rules and checks for each possible attack if an attacker is currently progressing along the corresponding attack path. Creating correlation rules by hand is a time consuming and error-prone task. This explains why only a few simple attacks where considered in many used tools. Recent work aimed to automate the creation of correlation rules [162]. Starting from the description of an attack (often provided by a security expert), the automatic production of the correlation rules requires to have a knowledge database with a precise description of the whole system. The main difficulty lies in the fact that a link has to be established between any action of a specified attack and its possible observations by various probes. This mapping is sometimes trivial (for example, in the case of an exploit of a well known and referenced vulnerability) and sometimes less obvious (for example in the case of a more harmless action for which the descriptions and the classifications may differ from one probe to another).

The signature-based approach has the advantage of causing very few false positives if the description of the attack and the associated correlation rules are sufficiently accurate. Regarding the false negatives, only the known attacks can be detected. A new attack or even an attack that is intentionally slightly different from a known one are not necessarily detected (0-day exploit). Moreover, the detection of a known attack occurs only if the probes are in sufficient number, well placed to cover all the system and correctly configured.

Current challenges

The prospect of automatic rule generation creates new challenges. As the human cost of creating rules decreases, the number of attack signatures can now increase. Consequently, a correlation engine has to face a scalability issue as it may have to cope with thousands of correlation rules while maintaining a rather high analysis speed (i.e., it may have to consume hundreds of elements of the incoming flow of events and low alerts per second) [220]. Moreover, as the set of correlation rules becomes larger, it has to be updated more frequently: some devices are removed or reconfigured, others are added and countermeasures taken for security reasons (applying a patch, closing a port, ...) also affect the system. Restarting the analysis from scratch each time a dynamic update is done is a risky solution: old observations (done before the update) will be ignored during the analysis of the new correlation rules. Thus a trade-off has to be found between the cost of a partial re-examination of past observations and the risk of missing the first steps of a new specified attack.

In a signature-based approach, the correlation rules mainly refer to low level alerts (and abnormal events) and to a lesser extent to normal events. On the contrary, the second intrusion detection approach that will be discussed next, namely the anomaly-based approach, focuses rather on the normal events. Despite the differences between the approaches, combining them seems to be of interest especially when the goal is to detect muti-steps attacks.

5.1.2 Anomaly-based intrusion detection

The signature-based approach is inefficient in detection of attacks that are either new or were modified (obfuscated) to a sufficient extent, and thus constitute the so-called 0-day exploits. Therefore, a solution to that problem comes in the form of anomaly detection. The following procedure can serve as an outline for the approach: firstly, the pattern of normality (normal traffic/activity) has to be established and then matched against the current traffic/activity samples. Whenever, the pattern deviates from the established model an alarm is raised (See Figure 5.2). This approach, however, is plagued by false positives (false alarms). Quite frequently, when the characteristics of network traffic (or e.g. HTTP requests in the application layer) evolve, such situation is interpreted as anomalous, even though it is just an intrinsic feature of network usage and of network users behaviour [59].

Concisely put, in setups where new attacks (or even slightly modified families of malware) emerge continuously, the standard protection systems become inconsequential until relevant signatures are collected [101]. On the other hand, anomaly-based approaches (systems which detect abnormalities in traffic, e.g. abnormal requests

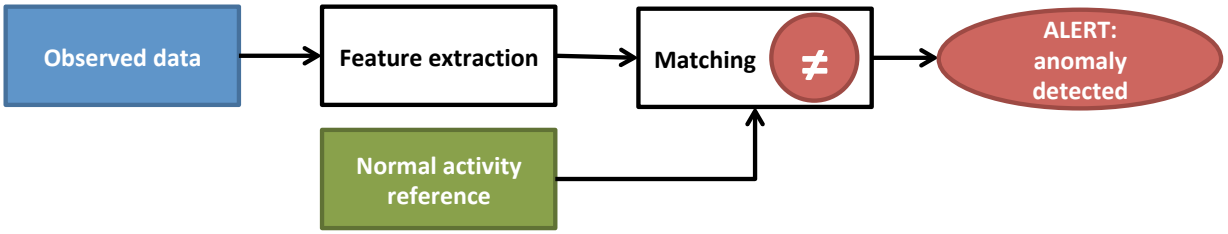


Figure 5.2: Anomaly Detection Approach for Cybersecurity Detection

to databases) tend to produce false positives (false alarms) [18, 315].

Various techniques can be used to obtain a model that characterises the normal patterns. All of them are related to machine learning. Thus numerous machine learning anomaly detection techniques are evaluated by the scientific community. Algorithms like the Random Forest, Support Vector Machine (SVM) or clustering techniques like the k-nearest neighbour or k-means are amongst the most proliferated. In [329] a method based on an SVM in conjunction with fuzzy C-means clustering is utilized. The clustering is used as a feature extractor with the SVM doing the classification work. An interesting approach is depicted in [224], where the authors generate normalized entropy of 6 netflow-based features for anomaly detection, and then follow with turning the issue back into a classification task for their hybrid Particle-Swarm-Optimisation-SVM model. [295] does a comprehensive survey of Intrusion Detection and Prevention Systems for Smart Grids. Among these 17 anomaly detectors were considered, including a system based on clustering data collected by a honeypot [376], a false-data injection detector relying on spatiotemporal evaluation [92], and an anomaly detector using a one-class SVM trained on MMS and GOOSE protocol data [399]. In [129] a k-nearest-neighbour is used as an outlier detector relying on distance and it is applied in urban traffic flow. In [196] the authors cluster netflow data basing on a sliding window technique. When the goal is to describe normal behaviors at the application level, models based on automata or likely invariants are used as they are close from the way the applications have been specified during the design phase.

In practice, while designing and developing intelligent systems for anomaly and cyber-threat detection one can draw the following conclusions:

- When it comes to the cyber security and cyber-attack detection, there is no single classifier or IDS system that will allow the recognition of all kinds of attacks. Likewise, the same system (even if it trained to detect the same type of attacks) has to be re-trained when changing the monitored network (topology, services, characteristics, etc.). In that regard, a transfer learning mechanism seems necessary to enable the detection of attack B based on the knowledge acquired for attack A.
- There seems to be an overlap of knowledge that an intelligent and adaptive system will need to be aware of. This phenomenon can be leveraged both to facilitate the learning of new tasks and to improve the effectiveness in the execution of the old ones.
- An IDS trained in one network will use already established knowledge to detect attacks in another network in a more accurate way (than without the lifelong learning approach).

Consequently, based on the above observations, we identify challenges in three main directions:

5.1.2.1 Lifelong learning for cybersecurity

Originally, lifelong learning was established as a sequence of learning tasks that need to be solved using the knowledge previously acquired and stored in classifiers that have already learnt [96]. According to [278] and [279], the theoretical considerations on lifelong learning are relatively widely described in the literature, in particular in the light of the growing popularity of machine learning approaches and applications. However, scientific communities usually put more attention to aspects of learning based on well-known knowledge domains and well-labeled training datasets, while approaches to lifelong learning (or learning to learn) without observed data, e.g. to perform new, unforeseen tasks are not yet very popular. In [40], one of the first attempts to describe the model of lifelong learning can be found. The author introduced a formal model called inductive bias learning, that can be applied when the learner is able to distinguish novel tasks drawn from multiple, related tasks from the same environment. Those considerations focused only on the finite-dimensional output spaces, and mainly on linear machines rather than nonlinear ones, in contrary to [247], additionally extending earlier research with algorithmic stability aspects. In [28], an approach to the problem of learning a number of different target functions over time is introduced, with assumptions that they are initially unknown for the learning system and that they share commonalities. Different approaches to solve this sequence of tasks include transfer learning [323], multitask learning, supervised, semi-supervised, reinforcement learning [16], and

unsupervised techniques. There are also works defining strong theoretical foundations for life-long machine learning concepts. Particularly, in [278] authors worked on a PAC-Bayesian generalization bound applied for lifelong learning allowing quantification of relation between expected losses in future learning tasks and average losses in already observed (learnt) tasks. The bulk of approaches so far assume that the problem representation is not changing, (i.e. the feature space). It is a common method in classical event correlation based solutions [99, 100]. However, recent works increasingly consider that also the underlying feature space can fluctuate. To overcome those challenges, solutions such as changing kernels for feature extraction [293], changing latent topics [97], or the underlying manifold in manifold learning [386, 387] are proposed. The Hybrid Intelligent Systems paradigm naturally addresses all the challenges of lifelong machine learning such as learning new tasks while preserving the knowledge of the preceding ones. In fact, classifier ensemble management resembles some of the algorithms proposed for lifelong learning. For example, a critical aspect of the lifelong learning systems is the ability to detect the task shift, which is quite similar to concept drift detection [381] and can be tackled by hyper-heuristics [332]. To deal with debatable cases in ensemble learning and to increase transparency in such debatable decisions, our hypothesis is that argumentation could be more effective than current resolution methods. Moreover, recent work on hybrid classifiers has demonstrated promising results of using an argumentation-based conflict resolution instead of voting-based methods for debatable cases in ensemble learning [107], showing that the hybridization of ensemble learning and argumentation fits the decision patterns of human agents.

The concept of a task appears in many formal definitions of lifelong machine learning models [279]. For example, when considering telecommunication network monitoring for cyber security purposes, it is often difficult to distinguish when a particular task finishes and the subsequent one starts, i.e. when a different family of attacks has started. Therefore, the lifelong learning approach fits very well with the reality in the cybersecurity domain.

5.1.2.2 Information granules/granular computing in cybersecurity

One of the most serious challenges of the methods and algorithms used in cybersecurity is being able to reach a correct understanding of the network data. Undeniably, cyber-ecosystems are quickly changing, as are the characteristics of the data. This fluctuation of properties produces uncertainty and difficulties in data partitioning/clustering. It is profoundly challenging to construct correct generalizations, rules and thresholds, and substandard choices greatly decrease the efficiency of typical pattern recognition and anomaly detection algorithms. In addition, many of the used pattern recognition techniques do not try to incorporate or even take into account the semantics of the analyzed network data. This sub-section addresses the utilization of the practical elements of Granular Computing for anomaly detection as a solution of the preceding problems.

Granular computing refers to a general data analysis and recognition framework, incorporating data partitioning into so called information granules. Granular Computing emerged as a general structure of data processing and knowledge discovery utilizing items called information granules. The very concept of granulation appeared independently in an array of fields, including fuzzy and rough sets or cluster analysis [35]. Granules are alignments of elements drawn together by their similarity, closeness or functionality [406]. A granule which occurs at a particular granularity level conveys a certain aspect of the modelled issue [34]. In situations with a certain degree of uncertainty granules can provide a convenient solution. This property can be translated into a certain economy when dealing with intricate problems. The tolerance for uncertainty bears a degree of resemblance to human thinking itself [406]. What follows is the utilization of Granular Computing in designing real-life smart systems. The hierarchical nature of Granular Computing in conjunction with the basics of human reasoning conveyed in its premise makes it a perfect match for meaningful abstraction on various levels of detail [395].

Granules are essentially tiny parts of a larger construct, which describe a particular facet of that construct, when viewed from a particular level of granulation [396]. As an illustration of this principle one can consider how in cluster analysis objects can be grouped together based on similarity or distance functions. Since objects grouped in one cluster should exemplify a strong degree of similarity, clusters can be considered as granules [395]. Granules can be, thus, amassed into larger collections, which are then perceived as new, larger granules or divided into smaller pieces, which are more specific [396].

Ideally, the extracted information granules should comply with the Principle of Justifiable Granularity (PJG) [277]. PJG is a guideline for information granules to best comply with two competing requirements: justifiability and specificity. It stipulates that the constructed granules cover the relevant portion of the data, but should not be highly dispersed across the dataset. This can be achieved by selecting granules that resemble relevant semantics describing the data. Typical practical methods of granular computing are fuzzy sets [373], rough sets [15, 165, 291], and intuitionistic sets [187].

Granular computing allows for better data understanding through the incorporation of semantic aspects, similarities and uncertainties. Granular computing has been used recently for the analysis of spatiotemporal

data [336], to concept-cognitive learning from large and multi-source data in formal concept analysis [267].

To the best of our knowledge, granular computing has not yet been widely examined nor adapted for cybersecurity application purposes. One of the rare published papers is authored by Napoles et al. [265]. The authors addressed the problem of modeling and classification for network intrusion detection by utilizing a recently proposed granular model named Rough Cognitive Networks (RCN). The authors both proposed and defined RCN for detection of atypical (abnormal and potentially dangerous) patterns in the network traffic. RCN has been delineated as a sigmoid FCM (Fuzzy Cognitive Map). Map concepts denote information granules corresponding to the RST (Rough Set Theory) -based positive, boundary and negative regions of decision classes. Learning mechanisms for RCN are based on a self-adaptive Harmony Search algorithm. The proposed model has been evaluated with the NSL-KDD dataset and is shown to be a suitable and promising approach for detecting abnormal traffic in computer networks. Future work will address validation and further evaluation of the model based on real network traffic.

5.1.2.3 Anomaly detection at the application level

Processes running on any distributed system cooperate and synchronize to achieve common tasks [159]. Various models can be built to characterize some normal behaviors of the running distributed application. At runtime, the flow of observations (mainly information about events at the application level) is used to check if the observed behavior corresponds to a normal activity or not. Any deviation from the model is interpreted as a consequence of an ongoing or completed attack.

In some rare cases, a specification of the normal behaviors is available and is used [263]. But most of the time, the model has to be built during a learning phase [47, 236, 241]. During this phase, multiple executions are performed in a safe context (without any occurence of attacks). Traces are collected and analysed to build models that reflect the characteristics of some normal behaviors. During the learning phase, only a finite (and rather small) number of behaviors can be learned while an application may exhibit an infinity of distinct behaviors. Therefore a rather high false positive rate is often observed when the model only accepts learned behaviors. In many cases, the constructed model is generalized in a second step to become more permissive and to accept close behaviors that have not been learned [240]. Of course, among the behaviors accepted by the model but never learned, some may unfortunately correspond a behavior corrupted by an attack (false negative). Generalization techniques are often ad-hoc mechanisms that can be tuned approximatively. If different models are used, a more global strategy can perhaps be adopted to have a better control on the obtained generalized models.

Usually, a (normal or malicious) behavior of a distributed computation is represented by a sequence of events. Such a knowledge requires to have a global clock to order all the events. To avoid this problem (and also to obtain more general models), a computation can also be represented by a partially ordered set of event [160, 358] (for example, the dependancy relation defined by Lamport [219] can be used). The interest of such an approach has been studied in the context of intrusion detection and in particular when the model of the normal behaviors is based on automata [237, 263] or likely invariants [46, 384]. Managing simultaneously several types of models has an interest [296] (especially if the construction of all these models relies on a common representation of the behavior [160]). In particular, it allows sometimes to decrease the false negatives (complementarity between some models) or to decrease the false positives (redundancy between some models). To limit false positives, an alert can be raised when a given subset of models detects an anomaly. On the contrary, to limit false negatives, an alert can be raised as soon as one model in a given subset detects an abnormal behavior.

As many models can be defined, one challenge is have a better knowledge about the specificities of each model and the possible complementarities between them. In the particular case of the IoT, and for the applications targeted in this study, it is also important to determine if some models are more appropriate than others (cost of the construction of the model, cost of the detection, quality of the detection, ...). This objective will probably be only partially reached. Indeed, whatever the used model, it is difficult to assert if an anomaly detection solution allows to detect a particular attack or not. Experiments show that an attack on a given application can be detected by a model in a particular context and not in another: in particular, the normal activity that occurs just before an attack may have a major impact on the detection [221].

The interest of combining a signature based approach and an anomaly-based approach deserves also to be studied. As both approaches can analyse the same incoming flow (composed of low-level alerts and events), they can be integrated in the same tool and complement each other. Indeed the boundary between both approaches is not so clearly marked. For example, to control that an invariant is satisfied, in some particular cases, the solution consists in checking that the negation of the property is never true. Thus, in this example, the fact that the application exhibits a normal behavior (anomaly detection approach) may be evaluated by a mechanism in charge of detecting occurrences of the corresponding bad behaviors (signature-

based approach). In the context of SPARTA, we aim also to investigate one of the differences between the two approaches. While a signature-based approach considers an incoming flow composed of low level alerts and events, an anomaly-based approach focuses mainly on the normal events contained in the flow. Yet, in the anomaly-based approach, the model can also be trained to learn both events and low level alerts that are generated during an execution free of any attack. In that case, the model takes into account the behavior of the application and the behaviors of the monitoring mechanisms that are rarely silent even when no attack is performed. A quite similar idea has been proposed in [370] where the authors suggest to model regularities in alert flows with classical time series methods.

5.1.3 Control-theoretic detection of cyber-physical attacks

Recent approaches in the intrusion detection literature propose the adaptation of traditional control-theoretic techniques to handle cyber-physical attacks. Such attacks target the physical process associated to infrastructures enabled with computing and communication capabilities (e.g., attacks against IoT-enabled infrastructures). Hence, the goal is to build detection techniques capable of identifying intentional cyber-physical attacks, in addition to failures and errors. From a control-theoretic standpoint, the protection of such infrastructures requires the maintenance of three crucial properties: observabilty, controllability and stability. Observability means that a defender must always be able to accurately estimate the state of the physical process. Controllability implies that the defender is all the time able to act upon the process. Stability is preserved when the defender manages to keep the process at or near the desired operating point. Cyber attacks targeting a disruption of the physical process aim to compromise such three properties (observability, controllability and stability) while evading detection, e.g., by hiding the actions; or concealing them to the eyes of the defender as failures, if detected.

5.1.3.1 Representative cyber-physical attacks

Table 5.1 [305] shows some representative cyber-physical attacks in the literature. They all assume a control-theoretic modeling approach in which intruders are manipulating inputs and outputs of the system, as depicted in Figure 5.3. The figure represents intruders perpetrating attacks against networked-control systems (e.g., industrial control systems, power grids, smart vehicles, industrial internet of things, etc.). The system is composed of a controller and a system containing the physical process under control (e.g., interfaced to the controller via sensors and actuators) and whose communications are indirect through a network. We assume that the defender is placed within the controller. The symbol $\bigoplus$ is a *summing junction*, an element that calculates the sum of signals. Controllers rule the overall system using the *feedback* received from the system, with the measured system output provided by the system sensors. The distance between a required reference output and the measured system output determines corrective control actions to handle faults, failures and errors. The goal of the intruder is to interfere the control loop communications. The intruder is not required to own a model of the system. However, access to actuators and sensors is assumed (e.g., in order to learn and get knowledge about the model of the system). Formally, the intruder can modify the genuine control input u_k, and inject a fake input u'_k to disrupt the system evolution. The intruder can also have access to all the components of the output vector y'_k, modify y'_k and produce a fake vector y_k with measurements that are consistent with the fake input u'_k. Insecure or vulnerable communication protocols (e.g., Ethernet and TCP/IP-based communication protocols) are enablers for the attacks of the intruder. Moreover, the attack may remain invisible to defenders

Table 5.1: Sample list of attacks reported in the control-theoretic literature

Attack name	Summary	References
Cyber-physical replay	Intruders replay previous measurements (corresponding to normal operation conditions) and modify control inputs to disrupt the system.	[257, 258], [306], [352]
Stealthy (false-data) injection, using bias and geometric techniques	Intruders drive the system to unstable states, by using system vulnerabilities, and injecting false data constructed to evade feedback-control detectors.	[256, 276], [307], [58, 80, 117, 234], [309], [352]
Zero dynamics	Intruders make unobservable an unstable state of the system using controller vulnerabilities.	[95], [308], [353]
Covert disruption	Intruders hold complete knowledge about the system dynamics, to impersonate the feedback controller and evade fault detection.	[183, 334, 335]

that look solely for faulty measurements, specially when the intruder succeeds at creating new measurements or control commands that are consistent with the original inputs and outputs of the system.

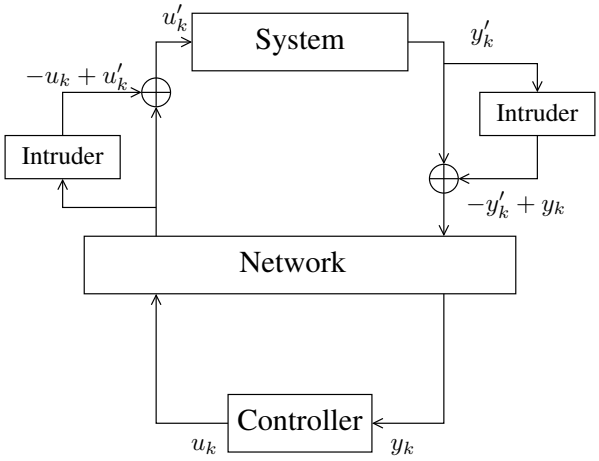


Figure 5.3: Cyber-physical adversarial model

The first attack listed in Table 5.1, cyber-physical replay, is depicted in Figure 5.4. We can identify an intruder conducting the attack by modifying some control inputs that disturb the physical process of the system, while modifying the sensors readings (e.g., by replicating previous measurements, corresponding to the nominal conditions of the system) to evade detection. The intruder is not required to have knowledge about the physical process, but only previous information generated by the sensors of the system. This type of attack is non detectable with a system monitor which is only verifying the absence of errors from the sensor measurements.

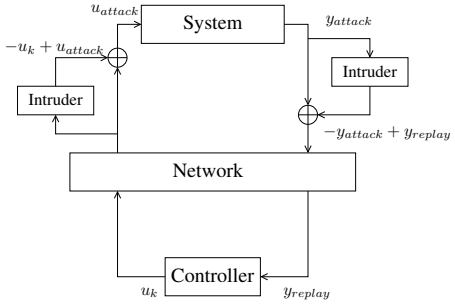


Figure 5.4: Cyber-physical replay attack

The following attack in Table 5.1, often referred to as stealthy or false-data injection attack [80, 117, 233] assume intruders modifying the sensors measurements by applying physical interferences through the communication channel (cf. Figure 5.5). The intruders require precise knowledge about the physical behavior of the system and the control laws. The intruders inject a bias in the sensors readings, y_k. The goal is to lead to wrong control decisions and cause large-scale malfunction. Extended versions of this attack assume the use of *geometric-injection techniques*, in order to gradually increase the bias using geometric control laws, as well as the existence of faulty dynamic models of the physical process, making an unobservable state unstable (referred to as *Zero Dynamics*t in Table 5.1). The attacks may easily evade detection in systems holding unstable modes, by simply hiding the disruptions under such unobseravble states.

The last attack in Table 5.1, denoted as covert disruption in the related literature [183, 334, 335] is depicted in Figure 5.6. The intruder manipulates measurements from sensors and commands from controllers. A precise knowledge about the physical system process and control design is required. The attack is considered as undetectable, if measurements are compatible with the forged commands. In other words, the attack cannot be distinguished from the regular system operation [335].

5.1.3.2 Detection of cyber-physical attacks

The attacks listed in Section 5.1.3.1 assume powerful adversarial models which, in addition to bypassing traditional cyber-security protection, suppose that intruders can observe and change measurements generated by sensors or commands provided to actuators. I.e., they assume a model in which the intruder can operate with the information going through a networked-control system. The models may also assume an

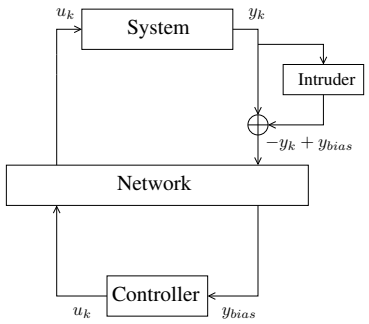


Figure 5.5: Cyber-physical stealthy attack using either bias injection techniques

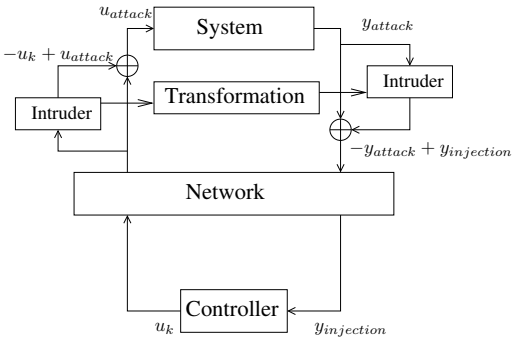


Figure 5.6: Cyber-physical covert disruption attack

initial phase of adversarial learning, in which the intruder observes and derives the physical model of the physical process. The intruder can use artificial intelligence techniques (e.g., machine learning and system identification tools [31, 266]) to obtain enough knowledge about the physical process under attack. Under such assumptions, we survey next a specific attack model and two representative detection methods.

Attack model — The state space representation is used to define the attack to the integrity of a system:

$$x_{k+1} = Ax_k + Bu_k + w_k \tag{5.1}$$
$$y_k = Cx_k + v_k \tag{5.2}$$

Eq. (5.1) models the evolution of the system. At time k, given input u_k, state x_k is transformed into state x_{k+1}. The transition is also affected by random noise w_k. At time k and in state x_k, the sensor measurements are y_k. The sensor measurements are also affected by random noise represented by v_k. Matrices A, B and C are respectively called the state, input and output matrices. Their exact dimensions and content depend on the application. For instance, a cyber-physical covert attack can be defined as follows [353]:

$$x_{k+1} = Ax_k + B(u_k + u_k^a) + w_k \tag{5.3}$$
$$y_k = Cx_k + v_k + s_k^a \tag{5.4}$$

where

$$x_{k+1}^a = Ax_k^a + Bu_k^a \tag{5.5}$$
$$s_k^a = -Cx_k^a \tag{5.6}$$

The variable u_k^a represents the contribution of the intruder to the input. Eq. (5.3) is the system transition following the injection of the signal by the intruder. Eq. (5.5) is the state transformation due to the intruder. In Eqs. (5.4) and (5.6), the term s_k^a represents the manipulation done by the intruder of the sensor measurements such that attack is not visible to the operator. It erases the effect of its input on the output. Existing detection techniques to identify this attack are discussed next.

Watermarking-based detection — Watermarking is a detection technique built upon an authentication

scheme proposed in [257]. It adapts an error detector to make an anomaly detector. The result is a cyber-physical attack detector using a *linear time-invariant* model of the system. Built upon *Kalman filters* and *linear-quadratic regulators*, the scheme uses authentication watermarks to protect the integrity of physical measurements communicated over the cyber and physical control domains of a networked control system. It is assumed that, without the protection of networked messages, malicious actions can be conducted to mislead the system towards unauthorized or improper actions, i.e., by disrupting the system services.

The authentication scheme proposed in [257] can be used as an anomaly detector of malicious stationary signals, to protect a networked-control system controlled by a feedback controller when the covert attack defined in Eq. (5.3) is perpetrated. We denote by u_k^*, the output of the controller, and u_k, the control input that is sent to the system (cf. Eq. (5.1)). The idea is to superpose to the optimal control law u_k^* a watermark signal $\Delta u_k \in \mathbb{R}^p$ that serves as an authentication signal. Thus, the control input u_k becomes:

$$u_k = u_k^* + \Delta u_k \tag{5.7}$$

The watermark signal is a Gaussian random signal that is independent both from the state (w_k) and measurement noises (v_k). The authentication watermark is used by the detector to identify the malicious signals originated by an intruder. Since the optimal control law u_k^* is combined with an authentication signal Δu_k, the detector (physically co-located with the controller) triggers an alarm whenever a malicious signal is observed, i.e., whenever the challenge sent by the controller over the system is not observed within the measurements returned by the system. Towards this end, [255, 258] propose to employ a χ^2 detector, i.e., a well-known category of real-time anomaly detectors classically used for fault detection in control systems [72], for the purpose of signaling anomalies identified in the system behavior.

The strength of watermarking is that it does not require any modification to the system. However, the latter must be tolerant to the injection of noise in the control input u_k, i.e., the Gaussian random signal. Further details about more powerful detectors, capable of identifying intruders empowered by learning and identification tools such as ARX (autoregressive with exogenous input) and ARMAX (autoregressive-moving average with exogenous input) [266], e.g., using such tools to evade detection, are also available in [309, 310].

Auxiliary system-based detection — The concept of auxiliary states can also be used in order to identify injection cyber-physical attacks [182, 319]. Under this second detection model, the CPS is augmented with a synthetic auxiliary state, synthetic outputs and optionally new inputs. The auxiliary state has a linear time-varying dynamics that is evolved in parallel with the CPS. The dynamics is concealed to the intruder. Because it is time-varying, it becomes a moving target that is challenging to identify by an intruder, a precondition to the covert attack. But, it is known to and used by the operator to detect the covert attack. The operator is in synchrony with the linear time-varying dynamics. It is therefore able to track it properly and compare the actual evolution of the auxiliary dynamics with the expected evolution. Significant discrepancies indicate the presence of anomalies, which can be used to identify the intruder.

The model is extended with the auxiliary state $\tilde{x}_k$ and additional sensors $\tilde{y}_k$, that measure the auxiliary state. The state x_k and auxiliary state $\tilde{x}_k$ are correlated. Together with the auxiliary state, the state transformation model is:

$$\begin{pmatrix} \tilde{x}_{k+1} \\ x_{k+1} \end{pmatrix} = \mathcal{A}_k \begin{pmatrix} \tilde{x}_k \\ x_k \end{pmatrix} + \mathcal{B}_k \begin{pmatrix} \tilde{u}_k \\ u_k \end{pmatrix} + \begin{pmatrix} \tilde{w}_k \\ w_k \end{pmatrix} \tag{5.8}$$

Together with the additional sensors, the sensor measurements are:

$$\begin{pmatrix} \tilde{y}_k \\ y_k \end{pmatrix} = \mathcal{C}_k \begin{pmatrix} \tilde{x}_k \\ x_k \end{pmatrix} + \mathcal{D}_k \begin{pmatrix} \tilde{u}_k \\ u_k \end{pmatrix} + \begin{pmatrix} \tilde{v}_k \\ v_k \end{pmatrix} \tag{5.9}$$

with matrices $\mathcal{A}_k, \mathcal{B}_k, \mathcal{C}_k, \mathcal{D}_k$ defined as follows:

$$\mathcal{A}_k = \begin{pmatrix} A_{1,k} & A_{2,k} \\ 0 & A \end{pmatrix}, \mathcal{B}_k = \begin{pmatrix} B_k \\ B \end{pmatrix},$$

$$\mathcal{C}_k = \begin{pmatrix} C_k & 0 \\ 0 & C \end{pmatrix}, \text{ and } \mathcal{D}_k = \begin{pmatrix} D_k & 0 \\ 0 & D \end{pmatrix}.$$

Hidden to the intruder, the state sub-matrices $A_{1,k}$ and $A_{2,k}$, the input matrix B_k, output matrix C_k and direct transmission matrix D_k are random variables. According to the approach proposed in [319], the actual matrices are randomly switched from time-to-time. The auxiliary system is a switched system [230]. The operator and CPS are synchronized on the switching sequence, perhaps through a switching signal. This secret is not

shared with the intruder. Sensor measurement $\tilde{y}_k$ is visible to the intruder, but changes over time in a random way. The intruder is challenged with learning the random auxiliary system state, input, output and direct transmission matrices.

The state, input and output matrices of the auxiliary system may be chosen such that the latter is asymptotically stable, i.e., small variations in the input generate small variations in the output. The output stays bounded for any bounded input and there are no oscillations. Notice that the auxiliary state model does not require injection of a noise signature. However, the system needs to be extended with a dynamics, which evolution is a secret shared between the controller and system. The management and exchange of such a secret is the main limitation of the approach. In the context of SPARTA, we aim to tackle and address such a limitation.

5.2 Resilience techniques based on fault and intrusion tolerance

Dependability has been defined as that property of a computer system such that reliance can justifiably be placed on the service it delivers. The service delivered by a system is its behaviour as it is perceptible by its user(s); a user is another system (human or physical) which interacts with the former [26]. Dependability is a body of research that hosts a set of paradigms, amongst which fault tolerance, and it grew under the mental framework of accidental faults, with few exceptions [130, 154], but we will show that the essential concepts can be applied to malicious faults in a coherent manner.

5.2.1 Intrusion tolerance concepts

The tolerance paradigm in security assumes that systems remain to a certain extent vulnerable; assumes that attacks on components or sub-systems can happen and some will be successful; ensures that the overall system nevertheless remains secure and operational, with a quantifiable probability. In other words:

- faults – malicious and other – occur;
- they generate errors, i.e. component-level security compromises;
- error processing mechanisms make sure that security failure is prevented.

Obviously, a complete approach combines tolerance with prevention, removal, forecasting, after all, the classic dependability fields of action!

5.2.1.1 AVI composite fault model

The mechanisms of failure of a system or component, security-wise, have to do with a wealth of causes, which range from internal faults (e.g. vulnerabilities), to external, interaction faults (e.g., attacks), whose combination produces faults that can directly lead to component failure (e.g., intrusion). An intrusion has two underlying causes:

Vulnerability – fault in a computing or communication system that can be exploited with malicious intention

Attack – malicious intentional fault attempted at a computing or communication system, with the intent of exploiting a vulnerability in that system which then lead to:

Intrusion – a malicious operational fault resulting from a successful attack on a vulnerability

It is important to distinguish between the several kinds of faults susceptible of contributing to a security failure. Figure 5.7 represents the fundamental sequence of these three kinds of faults: attack, vulnerability, intrusion and failure. This well-defined relationship between attack/vulnerability/intrusion is what we call the AVI composite fault model. The AVI sequence can occur recursively in a coherent chain of events generated by the intruder(s), also called an intrusion campaign. For example, a given vulnerability may have been introduced in the course of an intrusion resulting from a previous successful attack. Vulnerabilities are the primordial faults existing inside the components, essentially requirements, specification, design or configuration faults (e.g., coding faults allowing program stack overflow, files with root setuid in UNIX, naive passwords, unprotected TCP/IP ports). These are normally accidental, but may be due to intentional actions, as pointed out in the last paragraph. Attacks are interaction faults that maliciously attempt to activate one or more of those vulnerabilities (e.g., port scans, email viruses, malicious Java applets or ActiveX controls). The event of a successful attack activating a vulnerability is called an intrusion. This further step towards failure is normally characterized by an erroneous state in the system which may take several forms (e.g., an unauthorized privileged account with telnet access, a system file with undue access permissions to the hacker). Intrusion tolerance means that these errors can for example be unveiled by intrusion detection, and they can be recovered or masked. However, if nothing is done to process the errors resulting from the intrusion, failure of some or several security properties will probably occur.

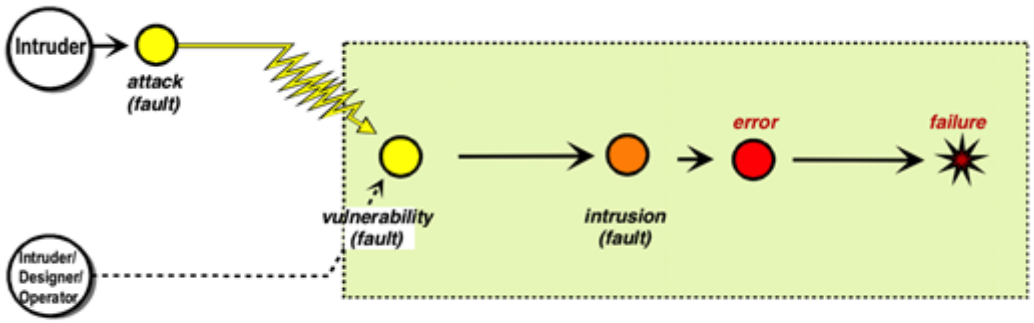


Figure 5.7: AVI composite fault model.

The composite fault model firstly describes the mechanism of intrusion precisely: without matching attacks, a given vulnerability is harmless; without target vulnerabilities, an attacks is irrelevant. Secondly, it provides constructive guidance to build in dependability against malicious faults, through the combined introduction of several techniques. To begin with, we can prevent some attacks from occurring, reducing the level of threat, as shown in Figure 5.8. Attack prevention can be performed, for example, by shadowing the password file in UNIX, making it unavailable to unauthorized readers, or filtering access to parts of the system (e.g., if a component is behind a firewall and cannot be accessed from the Internet, attack from there is prevented). We can also perform attack removal, which consists of taking measures to discontinue ongoing attacks. However, it is impossible to prevent all attacks, so reducing the level of threat should be combined with reducing the degree of vulnerability, through vulnerability prevention, for example by using best-practices in the design and configuration of systems, or through vulnerability removal (i.e., debugging, patching, disabling modules, etc.) for example it is not possible to prevent the attack(s) that activate(s) a given vulnerability. The whole of the above-mentioned techniques prefigures what we call intrusion prevention, i.e. the attempt to avoid the occurrence of intrusion faults.

Figure 5.8 suggests, as we discussed earlier, that it is impossible or infeasible to guarantee perfect prevention. The reasons are obvious: it may be not possible to handle all attacks, possibly because not all are known or new ones may appear; it may not be possible to remove or prevent the introduction of new vulnerabilities. For these intrusions still escaping the prevention process, forms of intrusion tolerance are required, as shown in the figure, in order to prevent system failure. As will be explained later, these can assume several forms: detection (e.g., of intruded account activity, of trojan horse activity); recovery (e.g., interception and neutralization of intruder activity); or masking (e.g., voting between several components, including a minority of intruded ones).

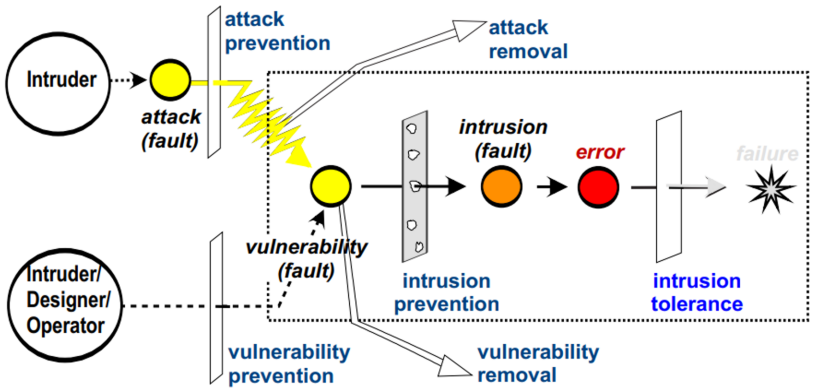


Figure 5.8: Preventing security failure.

5.2.2 Intrusion Tolerance Strategies

Not surprisingly, intrusion tolerance strategies derive from a confluence of classical fault tolerance and security strategies [367]. Strategies are conditioned by several factors, such as: type of operation, classes of failures (i.e., power of intruder); cost of failure (i.e., limits to the accepted risk); performance; cost; available technology. Technically, besides a few fundamental tradeoffs that should always be made in any design, the grand strategic options for the design of an intrusion-tolerant system develop along a few main lines that we discuss in this section. We describe what we consider to be the main strategic lines that should be considered by the architect of IT systems, in a list that is not exhaustive. Once a strategy is defined, design should progress along the guidelines suggested by the several intrusion-tolerance frameworks just presented.

5.2.2.1 Fault Avoidance vs. Fault Tolerance

The first issue we consider is oriented to the system construction, whereas the remaining are related with its operational purpose. It concerns the balance between faults avoided (prevented or removed) and faults tolerated.

On the one hand, this is concerned with the 'zero-vulnerabilities' goal taken in many classical security designs. The Trusted Computing Base paradigm [368], when postulating the existence of a computing nucleus that is impervious to hackers, relies on that assumption. Over the years, it became evident that this was a strategy impossible to follow in generic system design: systems are too complex for the whole design and configuration to be mastered. On the other hand, this balance also concerns attack prevention. Reducing the level of threat improves on the system resilience, by reducing the risk of intrusion. However, for obvious reasons, this is also a very limited solution. As an example, the firewall paranoia of preventing attacks on intranets also leaves many necessary doors (for outside connectivity) closed in its way.

Nevertheless, one should avoid falling in the opposite extreme of the spectrum – assume the worst about system components and attack severity – unless the criticality of the operation justifies a 'minimal assumptions' attitude. This is because arbitrary failure protocols are normally costly in terms of performance and complexity.

The strategic option of using some trusted components – for example in critical parts of the system and its operation – may yield more performant protocols. If taken under a tolerance (rather than prevention) perspective, very high levels of dependability may be achieved. But the condition is that these components be made trustworthy (up to the trust placed on them, as we discussed earlier), that is, that their faulty behaviour is indeed limited to a subset of the possible faults. This is achieved by employing techniques in their construction that lead to the prevention and/or removal of the precluded faults, be them vulnerabilities, attacks, intrusions, or other faults (e.g. omission, timing, etc.).

The recursive (by level of abstraction) and modular (component-based) use of fault tolerance and fault prevention/removal when architecting a system is thus one of the fundamental strategic trade-offs in solid but effective IT system design. This approach was taken in previous architectural works [285], but has an overwhelming importance in IT, given the nature of faults involved.

Attack tolerance is an extended in-depth strategy proposed to defend a system against any particular attack using several independent methods [213]. Several proposed security solutions with a focus on detection and attack prevention. However, preventing every single possible attack is hard to achieve. Despite the efforts, attacks can happen and be successful. Attack tolerance is the capability of a system to continue functioning properly with minimal degradation of performance, despite the presence of attacks. Some techniques proposed in the literature to achieve attack tolerance follow.

Early work [11] uses *indirection* to separate components using an additional layer that works as a protection barrier. For instance, proxies, wrappers, virtualization and sandboxes can play that role. *Voting* can resolve differences in redundant responses, to reach consensus w.r.t. the responses of perceived non-faulty components. The process involves comparing the redundant responses and reaching to an agreement on the results to find the appropriate response. It masks the attacks, thus tolerating them and providing integrity of the data.

Redundancy and diversity and often combined together to ensure protection beyond breach [194, 195]. Redundancy assumes the use of extra reserved resources allocated to a system that are beyond its need in normal working conditions. If the system finds that the output values of a primary component are not correct, then the responsibility is transferred to one of the redundant components. On the other hand, diversity means that a function should be implemented in multiple ways, differently at different times. For example, research has made it practical to automatically generate diverse functionality from the same source code or automatically change the configuration of a system from time to time to confuse the adversary.

Dynamic reconfiguration takes place after the detection of an attack. In traditional systems, reconfiguration is mostly reactive and generally performed manually by the administrator. Thus, it involves some downtime. Survivable systems need an adaptive reconfiguration to be proactive, instead. Under the context of distributed

trust, e.g., dividing trust into separated shares, decentralized strategies can be used to assure that the system needs to reach a given threshold prior granting authorization measures. Below the threshold, information gets concealed to the eyes of the adversary. When this approach is combined with recovery strategies, reaching a threshold allows the proper modification of the system to a state that ensures the correct provision of the required functions.

Other techniques emerging from the resilience literature include the use of self-healing and reflection, i.e., to programmatically assist a system to adapt itself while handling adversarial attacks. For instance, programmable reflection may enable a system to react and defend against disruptions [324]. When a malicious activity is detected, the system dynamically changes its behavior and enables techniques that ensure the correction of malicious events. The idea is to combine dynamic reconfiguration and recovery approaches, via control-theoretic and programmable networking tools.

Works exist in terms of neutralizing the adversarial actions via dynamic traffic sanitization [325]. Efficient network configuration plans can be used for neutralizing attacks. New networking functionality can be programmed using a minimal interface that can be used to compose high-level services. This idea was proposed as a way to facilitate the network evolution. Some solutions such as Open Signaling [79], Active Networking [354], and Netconf [146], among others, are early programmable networking efforts and precursors to current technologies such as SDN (Software Defined Networking) [212]. In particular, SDN is a programmable networking paradigm in which the forwarding hardware is decoupled from control decisions. The SDN model proposes three different functionality planes: data, control and management.

The data plane corresponds to the networking devices that are responsible for forwarding the packets. The control plane represents the protocols used to manage the data plane, such as, to populate the routing tables of the network devices. Finally, the management plane includes the high-level services and tools used to remotely monitor and configure the control functionality. A given security aspect may be related to different planes. For example, let us consider a network policy. It is defined in the management plane. The control plane enforces the policy while the data plane is instructed by the policy. The idea of using programmable networks for improving the security is not new. Some examples include its use for conducting a denial-of-service attack mitigation [317] and segmentation of malicious traffic [172, 282, 311]. Programmable networks provide a higher global visibility of the system, which favors attack detection. In addition, a centralized control plane may allow further possibilities to achieve dynamic reconfiguration of network properties, e.g., for the application of countermeasures.

Using the aforementioned ideas, reflection can be seen as a flexible way of equipping a system with the ability of examining and modifying its own behavior at runtime. Authors in [108, 199] proposed to implement programmable networks using reflective middleware platforms. They use reflection in order to configure and adapt at runtime nonfunctional properties such as timeliness and resourcing. To achieve this, the architecture is based in different components that may be loaded and unloaded dynamically in order to change the behavior of the platform and structure the programmable network. As a mitigation technique, reflection has also the potential to allow a system to react and defend itself against threats. When a malicious activity is recognized, the system can dynamically change the implementation to activate mitigation techniques to guarantee that the system will continue to work [324, 325].

5.3 Final remarks

Awareness of the imminent danger of targeted and persistent attacks by highly skilled and well-equipped adversarial teams, is increasing in modern societies. Yet, the body of techniques currently used in industry practice seems to lag behind in resisting those threats, as the many public accounts of breaches, black-outs, and reliability and integrity failures of important infrastructures show. Advances in building systems resilience will allow reaching a higher level of maturity in cybersecurity, and the possibility of making further steps in achieving security-by-design, in areas where: threats, uncertainty, real-time, etc.; require stronger, dynamic and automated paradigms for systems. Due to the objectives of Task 6.4, “resilience-by-design of intelligent infrastructures (II)”, this Chapter presented an overview of the main concepts and design principles relevant to Intrusion Tolerant (IT) architectures. In our opinion, Intrusion Tolerance as a body of knowledge is, and will continue to be for a while, the main catalyst of the evolution of the area of dependability. The challenges put by looking at faults under the perspective of “malicious intelligence” have brought to the agenda hard issues such as uncertainty, adaptivity, incomplete knowledge, interference, and so forth. We believe that fault tolerance will witness an extraordinary evolution, which will have applicability in all fields and not only security-related ones. Following the Task 6.4 roadmap, we aim to validate the proposed fault intrusion tolerance schemes as a next plan in future.

Chapter 6 Privacy-by-Design

This chapter focuses on current privacy threats and privacy challenges for Intelligent Infrastructures (II). Intelligent and connected devices expand the opportunities to collect personal and vital data, both in volume and precision. Therefore, privacy challenges should be addressed at design level (notion of "privacy by design"), for instance with data minimization principles and data usage control, so privacy risks are mitigated as much as possible. There are several Privacy-Enhancing Technologies (PETs) that use modern cryptographic schemes, such as the attribute-based credentials and groups signatures, and that can contribute in privacy-preserving design concepts. Nevertheless, only few PETs are suitable and ready for constrained devices deployed in various IoT scenarious. This chapter contains 4 sections. Section 6.1 presents current security and privacy threats in Intelligent Infrastructures and IoT. Section 6.2 deals with management and regulations from a data privacy point of view. Section 6.3 briefly introduces the readiness analysis for the adoption and evaluation of privacy-enhancing technologies for intelligent infrastructure. Suitable privacy evaluation techniques are then presented in Section 6.4.

6.1 Analysis of Privacy Threats and Attacks: Technical Attacks, Privacy-Based Leakages, and Social Aspects

6.1.1 Introduction

Internet of Things (IoT) encompasses systems responsible for the collection, storage, transmission and manipulation of data involving individual participants and devices, mobile devices and infrastructure [280]. There exist a number of surveys related to the IoT privacy and security risks and threats [5, 9], frameworks [17, 390] or specific components [29, 175, 225].

The rest of this section is structured as follows: In Subsection 6.1.2 three architectural layers of the IoT systems are introduced. Then Subsection 6.1.3 overviews different privacy threats and leakages. Subsection 6.1.4 includes the concluding summary.

6.1.2 Asset Description

The IoT system components include (*i*) systems that collect data (*ii*) systems that transmit collected data and (*iii*) systems that provide the data to end-users following a predefined process [280]. The intelligent infrastructure is a type of IoT system as it encompasses cooperative interactions of a variety of things or objects, to reach a common goal [25]. The IoT system consists of three architectural layers [226, 391, 393, 402, 409, 410]:

- Perception: The perception layer consists of hardware and software components (sensors, actuators, visioning, and positioning devices), carrying out basic functions of collection, controlling and storage of data.
- Network: The network layer facilitates wireless or wired transmission (in-vehicle, vehicle to vehicle, and vehicle to infrastructure) of collected data from perception components.
- Application: In the application layer, the network layer meets the end-user, application processes, computing, and storage, allowing high-level intelligent processing of the generated and transmitted data.

6.1.3 Privacy Threats and Leakages

6.1.3.1 Technical Threats

A risk is defined as an event where the *vulnerability* of a system asset is exploited by an attacker (*threat*) leading to the *impact* – a negation of the criteria of the business asset in a system. Table 6.1 summarizes the threats at the different architecture layers. The threats are categorized following the STRIDE threat model based on the first impact experienced [7].

Perception layer threats attack the sensing, vision, positioning and actuating components. Following [7] Table 6.1 includes 24 threats. *Network layer threats* affect the system assets' ability to transmit the necessary data for an IoT function. Data in is typically transmitted through local/ internal network, device-to-device, and device-to-infrastructure communication technologies. To illustrate the network layer threats, Table 6.1

assembles 47 threats [7]. *Application layer threats* involve attacks to disrupt or corrupt high level IoT processes and services. To illustrate them, Table 6.1 includes 12 threats.

Table 6.1: Summary of technical threats

System Asset	Threats					
	S	T	R	I	D	E
Perception layer	Spoofing, Node Impersonation, Illusion, Replay, Sending deceptive messages, Masquerading	Forgery, Data manipulation, Tampering, Falsification of readings, Message Injection	Bogus message	Stored attacks, Eavesdropping	Message saturation, Jamming, DoS, Disruption of system	Backdoor, Unauthorized access, Malware, Elevation of privilege, Remote update of ECU
Network layer	Sybil, Spoofing (GPS), Replay attack, Masquerading, RF Fingerprinting, Wormhole, Camouflage attack, Impersonation attack, Illusion attack, Key/Certificate Replication, Tunneling, Position Faking	Timing attacks, Injection (message, command, code, packet), Manipulation/Alteration/ Fabrication/Modification, Routing modification/manipulation, Tampering(broadcast, message transaction, hardware), Forgery, Malicious update (software/-firmware)	Bogus messages, Rogue Repudiation), Loss of event trace-ability	Eavesdropping, Man-in-the-middle, ID disclosure, Location tracking, Data sniffing, Message interception, Information disclosure, Traffic analysis, Information gathering, TPMS tracking, Secrecy attacks	DoS/DDoS, Spam, Jamming, Flooding, Message suppression, Channel interference, Black hole.	Malware, Brute Force, Gaining control, Social engineering, Logical attacks, Unauthorized access, Session Hijack
Application layer	Spoofing, Sybil, Illusion attack	Malicious Update		Eavesdropping , Location tracking, Privacy leakage	DoS	Jail-breaking OS, Social engineering, Rogue Data-center, Malware

6.1.3.2 Threats to Private Data Publishing

Typically private data are published in the following form [271]: Explicit_identifier, Quasi_identifier, Sensitive_attributes, Non-sensitive_attributes). Here Explicit_identifier are attributes that directly identify the person; Quasi_identifier (QID) are attributes whose combination could possibly identify. Sensitive_attributes are sensitive attributes that are specific to the person. Non-sensitive_attributes are all other attributes that do not fit into previous categories [157]. Private data publishing could suffer from few threats.

Privacy-preserving data leakages are *record linkage*, *attribute linkage* and *table linkage* [271]. During *record linkage* an attacker tries to map one or more records released dataset to victim. To do this, attacker will matches victims' QID from released data. This could lead to exposure of owner's sensitive data [157]. During *attribute linkage* the attacker who already knows QID of his target, can infer sensitive attribute of target based on sensitive attributes in target's QID group. The main goal of countermeasure methods is, then, to reduce the correlation between QID and sensitive attribute [157].

Record linkage and attribute linkage attacks are based on assumption that attacker already knows victim's record. But, sometimes it is enough for the attacker to know whether or not victim is released table. This happens when the sensitive attributes are too specific. In this case (i.e., *table linkage*) just knowing that the victim is present in the table can be already damaging [157].

6.1.3.3 Privacy Leakages Due to Poor Design

Access control is the process of mediating every request to data and services maintained by a system and determining whether the request should be granted or denied. Access control is a necessary condition to build privacy into IoT solutions and to demonstrate compliance with the General Data Protection Regulation (GDPR).

Consent unawareness. The content unawareness privacy threat indicates that a user is unaware of the information disclosed to the system. Thus, the user could either provide too much information allowing a malicious agent to retrieve the user's identity or, on the contrary, inaccurate information, which can lead to wrong decisions or behaviors. The proper design and enforcement of access control policies is a necessary condition.

Policy and consent non compliance. Policy and consent non compliance is a relevant privacy threat [126]. A policy specifies one or more rules, determined by stakeholder, with respect to data protection. Consents

specify one or more data protection rules, determined by the user and only relate to the data regarding this specific user. Policy and consent non compliance means that even though the system shows its privacy policies to its users, there is no guarantee that the system actually complies to the advertised policies. Therefore, the user's personal data might still be revealed.

Information disclosure due to wrong design and/or implementation of access control. The information disclosure threats expose personal information to individuals who are not supposed to have access to it [126]. This can happen in case the access control mechanisms in place are wrong. The modern approaches decompose access control in three main components: policy language, model, and enforcement [371]. However, writing and maintaining policies is an error-prone activity because of the possibility of inserting redundancies, conflicts, and other logical problems.

6.1.3.4 Social Aspects

In this subsection, we give an overview of possible data leakage threats regarding social bots, web search personalization and personalized ads.

Social bots and data leakage. Deception in Online Social Networks (OSNs) can take many different forms, such as *spammers*, *bots*, *cyborgs*, *compromised accounts*, *sybils*, and *fake followers* [232]. *Spammers* are those accounts that advertise unsolicited and often harmful content [339]. *Bots* are computer programs that control social accounts, as stealthy as to mimic real users [64]. *Cyborgs* interweave both manual and automated behavior [103]. These accounts are controlled by their rightful owners. *Compromised accounts* (are similar to cyborgs) are accounts that have been taken over my malicious users [408]. *Sybils* are multiple fake identities, created by a malicious user in order to unfairly increase their power [392]. *Fake followers*, are massively created accounts that can be bought from online markets to follow a target account and apparently inflate its popularity [113, 340]. Each of these categories has been the matter of several investigations, all sharing the ultimate goal of developing techniques for automatically detecting (and consequently removing) the different kinds of deceptive accounts.

Web search personalisation. E-commerce websites let users search for products by simply issuing a keyword search. Then, a number of filters can be activated to constraint the search results. The filters allow to narrow the results *wrt* user characteristics (e.g., location, profiles, and personalized items). The dark side of personalization relies however in the fact that filters can be activated, or changed, without the user's awareness and consent. In this case, the search engine acts in a *not transparent* way [283]. Consequences of lack of transparency are, e.g., to hide potentially interesting products [275], give relevance to some news with respect to others [110], expose different prices for the same product, depending, e.g., on the characteristics of the user making the search [253], and even reveal users' private information [106].

Online Advertising. With the online adverting it became possibility to target the specific interests of the user, other than on the content of the website hosting the advertisement. The literature highlights several privacy threats [84]. In [85], Castelluccia *et al.* show how to reconstruct user profiles from targeted ads displayed on the users' browser. For instance, from profile-based ads they infer users' interests. In [211], Korolova considers privacy violations in Facebook through microtargeted ads. Similarly, in [74, 83], Cascavilla *et al.* show several techniques to retrieve supposedly hidden information from Facebook user profile, while they do not rely on advertisements.

Targeted ads are connected with users' behavior. In [250], the authors show that publishers can alter the user's profile, in order to make them the target of the most remunerative ads. In [41], Bechmann highlights how profiling is related to privacy violations through a media economics and management perspective. In [142] authors analyse how the popular brand traces users' navigation behavior through its e-commerce website, it collects data about users and it sends them to third party websites that provide ads, without the user's explicit consent. Work in [106] showed how to maliciously exploit the Google Targeted Advertising system to infer personal information in Google user profiles.

6.1.4 Concluding Summary

We overview the technical privacy threats, threats to private data publishing, privacy-based leakages, and social aspects. Based on the systematic review [7], we classify technical threats to the STRIDE categories at three asset layers. We discuss threats to private data publishing and present the privacy leakages due to the poor design through the access control and GDPR perspectives. Social aspects are considered taking into

account social bots, Web-search personalization, and overview of the online advertising threats.

6.2 Privacy-Preserving Management and Regulations

The purpose of this section is to elaborate a general presentation of the General Data Protection Regulation ("GDPR") and its implications for the specific issue of IoT. We will successively develop the scope of the regulation and its key definitions, its general principles for ensuring the legal processing of personal data, the security requirements and the sharing of responsibility in the event of infringement of the GDPR.

6.2.1 Challenges

There are several challenges that IoT poses to the protection and preservation of personal data. Firstly, the data processing may be invisible to the data subjects. Indeed, they may not be really aware of the data used, the various processing and the potential consequences. Secondly, the IoT is also characterized by a multitude of actors and stakeholders in the development process that has an impact on increasing number of processing of personal data and the exchange of information. In addition, another important challenge is the fact that we are facing with a miniaturization of these connected objects. They are smaller and smaller and easily transportable which light lead to geo-tracking and profiling activities.

6.2.2 Privacy-Preserving Management: a GDPR Perspective

6.2.2.1 Personal Data Management

In order to have an efficient management of personal data, two concepts are fundamental. First, the privacy by design obligation requires the data controller to adopt internal policies and implement measures that comply, from the design of the tool or service, with the regulation on the protection of personal data. These principles include notably:

- the obligation to define, before any processing operation, an objective/a purpose,
- to then determine the data strictly necessary to achieve this objective,
- to set the storage period for personal data,
- to put in place an appropriate level of security of the personal data.

The purpose limitation is also crucial for a data controller active in IoT application and services. On this purpose, the Article 29 Working Party states that "*The increase of the amount of data generated by the IoT in combination with modern techniques related to data analysis and cross-matching may lend this data to secondary uses, whether related or not to the purpose assigned to the original processing. Third parties requesting access to data collected by other parties may thus want to make use of this data for totally different purposes*"[1]. IoT stakeholders must therefore be vigilant concerning the compatibility test for raw data, extracted data or displayed data[2].

Particularly in the context of IoT, the period of conservation could be different according to the various stakeholders (e.g. creator of the algorithm, creator of the sensors, vehicle manufacturers). Indeed, the Article 29 Working Party indicated that "*This necessity test must be carried out by each and every stakeholder in the provision of a specific service on the IoT, as the purposes of their respective processing can in fact be different. For instance, personal data communicated by a user when he subscribes to a specific service on the IoT should be deleted as soon as the user puts an end to his subscription. Similarly, information deleted by the user in his account should not be retained. When a user does not use the service or application for a defined period of time, the user profile should be set as inactive. After another period of time the data should be deleted*"[3].

The principle of transparency is crucial in the IoT context. Indeed, the Article 29 Working Party highlights the lack of control for the users and information asymmetry[4]. For example, in the case of connected vehicles, it is particularly important to inform the person of the possibilities of geolocation and when the data subject is geolocated or not, the operations to be carried out to stop the geolocation, etc. Indeed, as explained by the Article 29 Working Party, the "*personal data should never be collected and processed without the individual*

[1] Art. 29 Working Party, Opinion 8/2014 on the on Recent Developments on the Internet of Things, 16.09.2014, WP 223, p.7.
[2] Art. 29 Working Party, Opinion 8/2014 on the on Recent Developments on the Internet of Things, 16.09.2014, WP 223, p.8.
[3] Art. 29 Working Party, Opinion 8/2014 on the on Recent Developments on the Internet of Things, 16.09.2014, WP 223p. 17.
[4] Art. 29 Working Party, Opinion 8/2014 on the on Recent Developments on the Internet of Things, 16.09.2014, WP 223, p.6.

being actually aware of it. This requirement is all the more important in relation to the IoT as sensors are actually designed to be non-obtrusive, i.e. as invisible as possible"[5].

6.2.2.2 Security Policy

The second fundamental concept is part of the security policy. Indeed, the General Data Protection Regulation adopts a risk-based approach. The Article 29 Working Party insists on the fact that the risk-based approach is not a basis for the diminution of the effectivity of the right to privacy and the protection of personal data but must be understood as a scalable and proportionate manner to be compliant with the regulation[6]. Indeed, the Article 29 Working Party highlights that "*the scalability of legal obligations based on risk addresses compliance mechanisms. This means that a data controller whose processing is relatively low risk may not to do as much to comply with its legal obligations as a data controller whose processing is high-risk*"[7]. It adds that "*There can be different levels of accountability obligations depending on the risk posed by the processing in question. However controllers should always be accountable for compliance with data protection obligations including demonstrating compliance regarding any data processing whatever the nature, scope, context, purposes of the processing and the risks for data subjects are*".

Thus, by considering the nature of the personal data, the volume of the personal and the processing operations, the data controller must evaluate the risks, the probability that these risks will occur and the seriousness of the risks for people[8]. This does not include solely risks to privacy and the protection of personal data but also to freedom of speech, freedom of thought, freedom of movement, discrimination, etc. [124].

In order to examine the probability of a risk occurring, ENISA has put in place a methodology based on the network and technical resources, the processes and procedures of the processing of personal data, people and parties involved in the processing of personal data and business sector and scale of processing[9].

Traditionally, the security of personal data means the respect of the integrity and confidentiality of information. The data controller has to prevent unauthorised access and unauthorised use. Furthermore, the integrity requirement imposes to ensure that the personal data has not been altered before, during and after the processing.

In addition to these two obligations, there is the availability of personal data and the authenticity requirements. The notion of availability refers to the possibility of the information, the systems and the processes to be accessible and usable on demand by an authorised natural personal or an entity. The Article 29 Working Party includes in this notion the destruction of the personal data, the accidental or unlawful loss of personal data and the accidental or unlawful loss of access to the personal data[10].

In addition to preventive measures, it is necessary to provide for control measures as well. This is the authenticity requirement. The data controller must therefore keep, for a certain period of time, the information on who had access to which personal data[11].

6.2.2.3 The Data Protection Impact Assessment (DPIA)

Article 35 of the GDPR provides for a new obligation attributed to the data controller. Firstly, the data controller has to verify if the nature, the volume and the processing of personal data may entail a high risk for the protection of the rights and freedoms of the data subject. Secondly, if there is a high risk, an impact assessment needs to be achieved.

This assessment must include the measures, safeguards and mechanisms envisaged for mitigating the risk, ensuring the protection of personal data and demonstrating compliance with the GDPR[12]. The Article 29 Working Group recommends that all relevant parties be involved in carrying out the impact assessment. The controller shall require help from any person who is competent to assess the risk, such as the data protection officer or security advisor. The risk assessment (with or without an impact analysis) and the impact assessment carried out should be submitted to the highest person in the hierarchy[13].

[5] Art. 29 Working Party, Opinion 8/2014 on the on Recent Developments on the Internet of Things, 16.09.2014, WP 223, p. 16.
[6] Art. 29 Working Party, Statement on the role of a risk-based approach in data protection legal frameworks, 30.05.2014, WP 218
[7] Art. 29 Working Party, Statement on the role of a risk-based approach in data protection legal frameworks, 30.05.2014, WP 218
[8] Art. 32 of the GDPR
[9] ENISA,"Handbook on Security of Personal Data Processing", December 2017. Available at: https://www.enisa.europa.eu/publications/handbook-on-security-of-personal-data-processing.
[10] Art. 29 Working Party, Guidelines on Personal data breach notification under Regulation 2016/679, WP 250.
[11] E.C.H.R., I v. Finlande, 17 July 2008, n. 20511/3 ; C.J., College van burgemeester en wethouders van Rotterdam v. M E.E. Rijkeboer, C-553/07 ; F. DUMORTIER, "La sécurité des traitements de données, les analyses d'impact et les violations de données", in Le règlement général sur la protection des données (RGPD/GDPR) – Analyse approfondie, C. DE TERWANGNE et K. ROSIER (coord.), Brussels, Larcier
[12] Article 35.7 of the GDPR.
[13] Art. 29 Working Party, Guidelines on Data Protection Impact Assessment (DPIA) and determining whether processing is "likely to result

6.2.3 Privacy by Design

GDPR model. A representation of the GDPR model using the UML class diagram notations can be found in [356], [201]. This model introduces the major principles for the personal data processing. *Personal data* [149] (Art. 4(1)) is represented with the class PersonalData. Data processing [149] (Art. 4(2)) is captured with the DataProcessing class, which also covers the cross-border processing [149] (Art. 4(23)) of personal data.

Controllers can also be *Processors*. The LegalGround presents that *data processing* must have a legal ground (whether consent or other). Consent is seen as a separate class that manifests one legal ground. The LegalGround, in turn, guides DataProcessing by setting the limits to the processing of personal data. Classes LegalGroundDataTransfer, LegalGroundSpecialCategory, and DataProtectionImpactAssessment represent regulation Art. 45-59, 9(2) and 35-36 respectively. The model also includes an obligation to issue the notification in case of a data breach (see, DataBreachNotification). The ProcessingLog artefact is created to meet [149] Art. 30, which requires maintenance of records of the processing activities.

Technical measures [149] (Art. 32(1)) are represented with the TechnicalMeasures class. The OrganisationalMeasures class describes how Controller should apply the organisational measures to *Data processing*. The model also describes the data processing principles and the principle of accountability (e.g., Controller *isAccountable* to PrinciplesOfProcessing) as described in [149] Art. 5. The detailed description of the GDPR model including the graphical representation, description of the data subject rights and overview of the special processing cases can be seen in [356] [201].

A Method for Achieving Compliance (see Fig. 6.1) consists of four steps:

- **Extract as-is compliance model**: First, one needs to check the current level of process compliance. This includes analysis of the business process and extraction of the GDPR model instance of the current state (see, *Extract AS-IS compliance model*). Hence the input is the business process model (expressed in *business process model and notation* together with GDPR model which serves as the means to extract the information.
- **Compare two meta-models**: Once the AS-IS model is constructed, it can be compared to the defined GDPR model.
- **Define compliance issues**: The result of the third step is a list of the non-compliance issues. This step gives a binary answer to the question whether the extracted compliance model is GDPR-compliant or not and give a detailed descriptions of business process non-compliances. Depending on the non-compliances, one makes a decision whether the model needs changes or not.
- **Change business process model**: In case of non-compliance, in the fourth step one *changes the business process model* so that the non-compliance in removed from the model. The compliance checking, then, continues with the first step taking the updated business process model as the input.

The GDPR model and the compliance checking method was already applied in several illustrative examples. For instance in [245] the feasibility is studied using the *tollgate scenario*. Elsewhere in [87], the GDPR model is used to support *modelling of the goal-actor-rule perspective*. The study shows how modelling language could be extended to capture infringement and to solve it using embodiment, finding irregularities, compliance checking and irregularity resolution activities. In [4] the GDPR model is applied in an *airline contact centre processes*. The results of both cases ([87] and [4]) was introduced to the domain experts who found the application of the GDPR model intuitive and helpful to achieve business process compliance.

In [143] the manual application of the method to achieve regulation compliance is compared to the tool-supported analysis. The results indicate a high correspondence between the number of found non-compliance issues. In addition the tool-supported application is able to highlight non-compliance issues (e.g., application of the technical measures), which were omitted from manual analysis.

in a high risk" for the purposes of Regulation 2016/679, 04.10.2017, WP 248, p. 15.

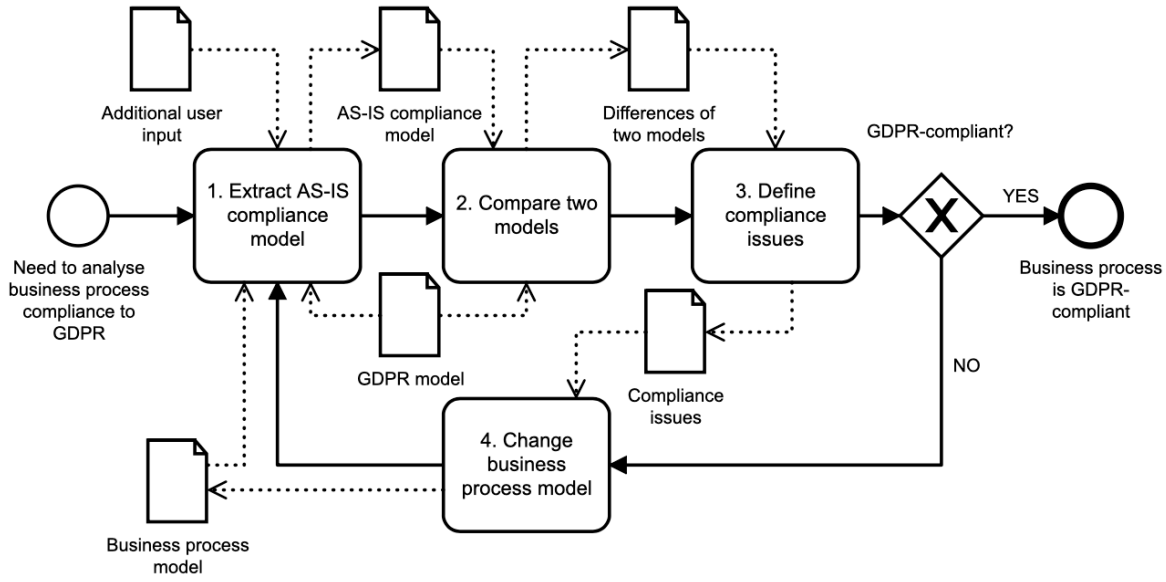


Figure 6.1: Method for achieving regulation compliance, adapted from [143] [201].

6.3 Privacy-Enhancing Technologies: Readiness Analysis for the Adoption and Evaluation of Privacy-Enhancing Technologies for Intelligent Infrastructures

This section categorizes privacy protection technologies and solutions based on the information, documents and reports from well-established institutions and expert networks such as the European Union Agency for Network and Information Security (ENISA) (the report: Privacy and data protection by design-from policy to engineering [118]), National Institute of Standards and Technology (NIST) (the report: An introduction to privacy engineering and risk management in federal systems [71]) and the Internet Privacy Engineering Network (IPEN) initiative (https://ipen.trialog.com/wiki/Wiki_for_Privacy_Standards). ENISA has been active in PETs for many years by collaborating closely with privacy experts from academia and industry. The categorization of PETs is based mainly on ENISA publications. ENISA defines PETs as the broader range of technologies that are designed for supporting privacy and data protection. These technologies and solutions are based on the principles of data protection, data minimization, anonymization, and pseudonymization. The ENISA report [118] provides a fundamental inventory of the existing approaches and privacy design strategies and the technical building blocks of various degree of maturity from research and development.

Further, ENISA released the report [294] that maps privacy-focused standards including concrete technical standards based on cryptographic techniques such as: ISO/IEC 18033 Encryption algorithms; ISO/IEC 18370 Blind digital signatures; ISO/IEC 20008 Anonymous digital signatures; ISO/IEC 20009 Anonymous entity authentication; and other standards.

PETs have been also studied in many research papers and surveys, e.g. [150, 164, 178, 330, 374, 378]. These documents present various insights and taxonomies of privacy protection techniques, privacy scenarios and privacy features.

In this section, we define and present a list of technical-based privacy-enhancing technologies that can be used in current ICT systems to protect data and user privacy within various scenarios such as computation, communication, user/data authentication, data encryption and data processing. The list is based on the most important national and international standards for data privacy and cybersecurity in the world as ENISA, NIST, IPEN etc. Table 6.2 presents privacy-enhancing categories and names of the privacy-enhancing technologies. The following subsections introduce a short version of our comprehensive analysis. The full version of our analysis will be published as an open access publication. To be noted that the following list of PETs does not contain all kind of privacy-preserving technologies and schemes.

6.3.1 Privacy-Enhancing Digital Signatures

Group Signature (GS) is a digital signature providing group-based authentication to achieve the privacy of signers against verifiers. GS schemes allow any group member (a user) to anonymously sign a message on behalf of the group. In two past decades, extensive research focused on group signature schemes has been done ($>$ 5k papers in Scopus). There are many variants of GS schemes providing various features. In general, GS can be used as basic layer/cryptographic primitive in privacy-preserving ICT services, mainly for proving

Table 6.2: Categories of privacy-enhancing (PE) technologies.

Privacy-Enhancing (PE) category	Technology name
PE digital signatures	Group signatures Ring signatures Blind signatures
PE user authentication	Attribute-Based credentials Anonymous and pseudonymous entity authentication
PE communication systems	Mix-networks and proxies Onion routing Privacy preserving techniques for wireless access network
PE encryption technologies	Attribute-Based encryption Homomorphic encryption Searchable encryption
PE computations and data storing	Secure multi-party computations Data splitting
General anonymization technologies	Statistical disclosure control Differential privacy algorithms

membership in a group and/or within signing a data behalf of the group. Moreover, several group signature schemes are included in the standard ISO/IEC 20008-2:2013 [152] and several public libraries including GS schemes are released in public repositories. Several papers focusing on group signatures in IoT have been recently published, e.g. [148], [383], [147]. Nevertheless, there is still ongoing work on the design of efficient group signatures with immediate revocation features appropriated for constrained devices and on the design of new GS schemes based on quantum-resistant assumptions. **Ring Signature (RS)** is a digital signature providing the privacy of signers. RS schemes are similar to GS schemes and some studies call them as ad-hoc group signatures. Nevertheless, RS schemes remove the centralization point of a group manager and RS do not need centralized initial setup (i.e. a join phase between a user and a manager). Users easily adhere to ring signatures by using prescribed cryptographic parameters and create non-closed groups. RS schemes usually provide a perfect privacy (untracebility) because there is no authority that can revoke the anonymity of signers. Ring signature schemes have been studied since 2001 ($>$ 0.6k papers in Scopus). There are several variants of RS schemes providing various features. In general, RS can be used as basic layer/cryptographic primitive in ICT services with strong privacy-preserving requirements, e.g. e-voting and e-cash. Nowadays, ring signatures are employed in several cryptocurrencies and altcoins such as Monero, CryptoNote, TokenPay, etc. Nevertheless, ring signatures produce sized signatures by adding multiple public keys and require several expensive asymmetric cryptographic operations depending on the ring size. Therefore, RS schemes are more appropriate for desktop applications and webservices that run on non-constrained nodes. Several papers focusing on the implementation of ring signatures in IoT have been recently published, e.g. [125], [362], [139]. Nevertheless, there is still ongoing work on the design of efficient and logarithmic-sized ring signatures appropriated for constrained devices and on the design of new RS schemes based on quantum-resistant assumptions. **Blind Signatures (BS)** are a form of digital signatures which hide (blind) the content of a message to signers. However, resulting the blind signature can be publicly verifiable against the original (unblinded) message in the manner of a standard digital signature. Generally, we can consider blind signatures mature and ready to be used in digital systems. Many BS, e.g. [77], [337], are based on standard signature schemes which are widely applied in many security systems. These standard digital signatures have hardware support also on many constrained IoT devices such as smart cards. BS are mostly used in payment systems such as PayCash. Officially there is no standard which deals with BS, however, BS are based on standard digital signatures, hence we can consider their standardizations. The main goal of the current proposals is to build efficient and post-quantum resistant schemes, e.g. [411], [55].

6.3.2 Privacy-Enhancing User Authentication

Attribute-Based Credential (ABC), sometimes called **anonymous credential** or **private certificate**, is a core technology used in privacy-friendly authentication systems. The authentication is based on personal characteristics instead of user identity (i.e. full name, unique identifier, digital certificate X.509), which is widely used in current systems. In ABC context, the digital identity is considered to be a set of characteristics (personal attributes) that describe certain person. The attributes are grouped into credentials (cryptographic containers) and can be shown selectively, anonymously and without anyone's ability to trace or link the showing transactions. Many research articles focused on ABC technology were published in last years. We can consider this technology mature and ready to use in current ICT systems. In fact, there is already a running IRMA (I Reveal

My Attributes) pilot project with the IRMA card and mobile application product for privacy-friendly authentication. Furthermore, current ABC schemes are enough efficient to run even on constrained devices in IoT. For example, the article [76] presents anonymous scheme that runs `show protocol` less than 500 ms (in case of 3 stored attributes) on current smart cards. The necessity of this technology in authentication/identification systems have been also demanded by U.S. and E.U. institutions. The main drawback of the technology remains the revocation, however this issue have been solved in last years, for example in the paper [75]. Nevertheless, there is still ongoing work on the design of new ABC schemes. Other directions in future research are to provide ABC decentralized system in order to increase privacy and security and/or to transform ABC schemes to quantum resistant forms. **Anonymous Authentication (AA)** preserves user privacy. In an AA system, a user can get an access to the service without disclosing his/her identifier. This method prevents the verifier to track and profile them. However, the verifier can still reliably determine whenever the user is authentic or not. The authenticated user only provides a proof of knowledge of the secret for some chosen claims, e.g. a user belongs to the group with specific privileges. The most of the current AA schemes are formed by group signatures (ISO/IEC 20009-2 [153]), blind signatures (ISO/IEC 20009-3 [151]) or identity escrow schemes, see [205] for more details. AA can be applied in a range of applications and use cases including electronic voting, electronic identities, social networks or mobile payments.

6.3.3 Privacy-Enhancing Communication Systems

Mix networks (Mixnets) represent a basic privacy technology that is used for privacy-preserving communication via public networks such as Internet. Mixnets enable users to create an anonymous communication network that is protected against traffic analysis. Mixnets (introduced in 1981) have been actively studied since 2000 ($>$ 1.6k papers in Scopus). There are several variants and strategies of Mixnets protocols. In general, Mixnets provide anonymous communication which could be used as basic primitive for many use cases, e.g. anonymous email services, web browsing, message exchange and e-voting. Nowadays, Mixnets are offered to users via several open source tools and web projects. Mixnets technology has been studied primary for classic networks, nevertheless, there are few papers focusing on the implementation of Mixnets solutions on constrained devices (and IoT), e.g., [89], [318]. **Onion routing** is an anonymous communication technique used in computer networks. Onion networks employ an onion encryption approach. A sender with each onion router establishes a single encryption layer. The encrypted data are transmitted via a series of network nodes called onion routers. The data are encapsulated in several layers of encryption, analogous to onion layers. The most mature project is Tor (The onion router). Tor [128] is based on a circuit-based low-latency anonymous communication service and onion routing. Works such as [179, 298] deal with the deployment of DTLS in onion routing and its efficiency. The paper [179] employs Datagram Transport Layer Security (DTLS) in order to tailor onion routing to IoT and presents the practical evaluation of the tailored solution in IoT.

In wireless access networks the traffic carried over the wireless link is in general encrypted (e.g., by WPA in IEEE 802.11 Wi-Fi networks). However, the headers and the content of management frames are not protected and are thus available to snoopers. The exposure of this information poses serious privacy threats that are made critical by the massive adoption of portable devices and wireless networks. Historically, two types of problems have been identified [114] [115] [168] [169]: The first one concerns the scan for nearby Wi-Fi access points actively sending probe requests. Another aspect with 802.11 frames is the use of the device MAC address, a globally unique identifier tied to the device. Using this identifier, it is possible to detect the presence of people and track them in the physical world. The use of wireless access technologies, e.g. Wi-Fi, BlueTooth, or BLE, in mobile equipments including IoT connected devices, raises privacy concerns. Informed of such problems, the manufacturers and the standards developing organizations have improved the practices (e.g., by banishing SSID disclosure in Wi-Fi access point active search mechanisms) and have designed **privacy extensions** for **wireless access technologies**, in particular the use of randomized MAC addresses during several modes of operation. However research has shown that this is not sufficient to fully prevent privacy risks (e.g., re-identifying an equipment that uses MAC address randomization is often possible). The feasibility of tracking wireless access network devices in the wild has been identified by several research works, namely [168] [169] [114] [115].

6.3.4 Privacy-Enhancing Encryption Technologies

Homomorphic encryption (HE) is a special form of encryption technique providing data security. In contrast to standard encryption methods, HE allows an evaluator (third party) to apply specific functions (computations) on encrypted data. However, both data and result remain encrypted and inaccessible to evaluator throughout the whole process. Especially fully homomorphic encryption (FHE) technology has become more interesting research area in the last decade. This increase is caused mostly by the growing of cloud services and outsourced computations. Currently. there are around 1k papers dealing with FHE technology and around 3k pa-

pers focused on HE technology. There are several proposed FHE schemes targeting shortcomings of existing solutions. HE can be used wherever the computations on encrypted data are required. Nowadays, there is no official standardization of this technology. The pioneer standardization document is the document [12] created by the consortium of international industries, government and academia sectors. Furthermore, several public FHE libraries are released in public repositories. **Searchable Encryption (SE)** is a cryptographic technique which enables performing search operations using some keywords over encrypted data without disclosing any useful information about the actual content of the encrypted data and the searched keywords [174]. Using SE, any user, having proper credentials, can delegate the search capabilities to the cloud service provider without disclosing any useful information. SE has already become a promising privacy-preserving technology. From the last two decades, many schemes have been proposed to address various security issues and to provide different functionalities. In SE, it is very important to find and retrieve the requested data as quickly as possible. It is still remained as a challenge to design a computationally inexpensive SE mechanism. It has been observed that there is still much work to do for improving its efficiency while keeping strong security to adopt SE widely in IoT based applications. **Attribute-Based Encryption (ABE)** is a one-to-many public encryption mechanism, i.e., the same data can be shared with several users. ABE is categorized into two groups, namely, Key-Policy ABE (KP-ABE) [166] and Ciphertext-Policy ABE (CP-ABE) [49]. ABE has already emerged as a promising cryptographic technology. It has been used in a wide variety of environments such as cloud computing [242, 385, 389], mobile cloud computing [198, 239, 273, 394] and so on. However, ABE has several practical challenges that are hindering its wide adaptation in practical applications. First, revocation is a challenge in ABE systems. Each user may share the same set of attribute types. As such, revocation of a user may affect other non-revoked users who share their attributes with the revoked user. Second, ABE systems need costly cryptographic operations, e.g., pairing, elliptic curve multiplication and exponentiation operations to perform encryption and decryption. As such, ABE may not be suitable for the environments where devices have fewer resources in terms of computing and storage power unless computationally expensive operations are outsourced. Third, ABE systems suffer from the key-escrow problem, as the AA knows all the master secrets. Hence, they can decrypt any ciphertexts of their choice.

6.3.5 Privacy-Enhancing Computations and Data Storing

Secure Multi-party Computation (SMC) is a cryptographic problem in which n parties collaborate to compute a common value with their private information without disclosing to others [136]. Secure Multi-party Computations (SMCs) have already emerged as a promising and well-established privacy-enhancing technology. This can be observed from the available research projects and products. SMC can be suitable for various IoT/I-IoTs use cases where privacy-preserving computation is needed, e.g., smart metering, voting, auctions, etc. Although many works have been published for the use of SMC in practical applications [14, 19, 264], there is still much work to do in terms of reducing computation and communication overhead for wider use of SMC. **Data splitting (DS)**, data partitioning or fragmentation means dividing an original sensitive data set into fragments and storing each fragment in a different site, in such a way that the fragment in any site considered in isolation is no longer sensitive. Data splitting is used mainly in privacy-friendly cloud computation services for outsourcing user sensitive data as alternative to fully homomorphic encryption which is currently considered to be computationally inefficient. Currently, there are various DS schemes using different methods and processing differ type of data such as numerical (data consists only from numerical values), categorical (where data can be represented with string values) or files, e.g., Li *et al.* [227], Yang *et al.* [388], Domingo *et al.* [134]. The technology was used for example in European research project CLARUS, see http://www.clarussecure.eu/.

6.3.6 General Anonymisation Techniques

To support research and policymaking, there is an increasing demand for microdata, which is often collected from individuals. For service providers, microdata dissemination increases returns on data collection and helps improve data quality and credibility. But publishing the microdata poses the challenge of ensuring individuals' confidentiality/privacy while making microdata files more accessible. In order to preserve the privacy of individuals as well as the utility of the data, **Statistical Disclosure Control (SDC)** methods need to be applied before data release.

SDC methods have received a lot of attention from both academia and the organizations which need to deal with microdata data publication. In academia, researchers have been active in examining the limitations and improvements with respect to existing notions, e.g. [78, 132, 133]. Many new notions have been proposed, e.g. the p-sensitive k-anonymity [78]. SDC methods are typically vulnerable when the attacker gain unexpected background knowledge and access to auxiliary data. **Differential privacy** [141] is a formal mathematical concept for guaranteeing privacy protection when analyzing or releasing statistical data. In the book of Dwork and Roth [140], an example application is illustrated for social science research: in order to collect statistical

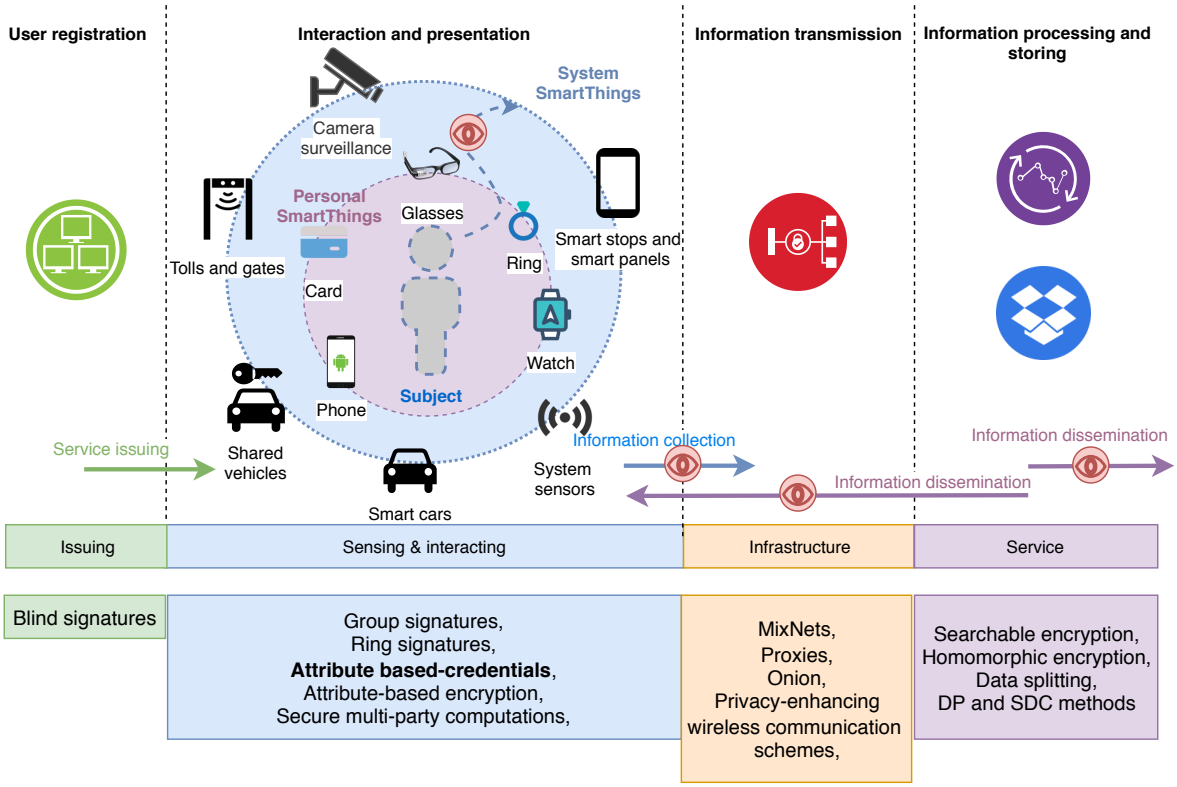


Figure 6.2: Privacy-enhancing technologies in the intelligent infrastructure environment.

information about embarrassing or illegal behavior (captured by having a property P), a randomized process can be implemented and produce some randomized responses. After the concept of **differential privacy** was proposed, the SDC methods have received more criticisms, due to the fact that these methods are vulnerable to background knowledge of the attacker while differential privacy methods normally allow the attacker to have unlimited background knowledge. Clifton and Tassa [104] gave a good comparison study to SDC methods and differential privacy. Recently, researchers have attempted to combine these concepts. For example, Li *et al.* [228] showed how to achieve differential privacy and k-anonymity in the same data release.

6.3.7 Concluding Summary

We overview 15 privacy-enhancing technologies that are divided into 6 privacy-enhancing categories: digital signatures, user authentication, communication systems, encryption technologies, computations and data storing, and general anonymization technologies. Several technologies such as attribute-based credentials, group signatures, mixnets have been already considered in IoT. Figure 6.2 shows the indicative positions of analyzed privacy-enhancing technologies in the intelligent infrastructure environment. Nevertheless, only the appropriate combination of PETs that cover various properties can protect privacy in more complex systems such as Intelligent Infrastructures. In the full version of our analysis, we explore PETs' properties, existed significant pilots/products/projects, pioneer and recent schemes, and suitable use cases for current ICT technologies including Internet of Things. The full version of the analysis will be published as an open access publication.

Furthermore, we have published publications focusing on a privacy-enhancing framework for Internet of Things services [243] and our proposal of fast keyed-verification anonymous credentials implemented on standard smart cards [76].

6.4 Privacy Evaluation Techniques and Methods

6.4.1 Introduction

This section presents a brief review of different techniques for assessing the satisfaction of (personal) data privacy properties as dictated by privacy requirements and privacy policies, possibly derived by the newly adopted General Data Protection Regulation (GDPR), the new regulation in EU law on data protection and

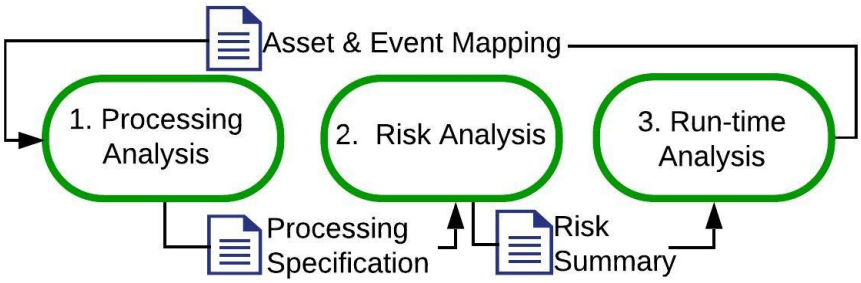


Figure 6.3: An overview of our methodology.

privacy for all individual citizens of the European Union (EU) and the European Economic Area (EEA). GDPR became enforceable on May 25, 2018. We first introduce a methodology and tool for Data Protection Impact Assessment (Section 6.4.2) and we then consider how data policies can be expressed in a machine-readable language, amenable for policy analysis and policy enforcement. Finally, we illustrate the main state-of-art techniques for privacy policy enforcement.

6.4.2 Tool-Assisted Methodology for Data Protection Impact Assessment

In this section we describe a tool-assisted methodology for the Data Protection Impact Assessment. According to the Working Party 29,[14] a Data Protection Impact Assessment (DPIA) "*is a process designed to describe the processing, assess its necessity and proportionality and help manage the risks to the rights and freedoms of natural persons resulting from the processing of personal data by assessing them and determining the measures to address them.*" DPIA is one of the most important activity for an organization to demonstrate compliance with the General Data Protection Regulation (GDPR). Unfortunately, it is complex, time-consuming, and requires expertise in several domains. It is rarely the case that organizations—especially small or middle size ones—can afford the burden of developing an interdisciplinary portfolio of competencies, including cybersecurity and privacy. For larger organizations, another issue is to maintain uniformity of the DPIA across different departments.

To alleviate these problems, we propose a pragmatic methodology based on our previous experience in designing a methodology for the DPIA of the public administration of the province of Trento in Italy[15] and previous academic work on compliance of security policy [170, 297].

Our methodology is based on a tool that is capable of assisting users with the three main activities of DPIA, namely (1) the analysis of the data processing activities, (2) the assessment of the risks, and (3) the run-time monitoring.

For activity (1), the tool helps users in carrying out crucial activities such as the functional specification of the data processing activities, the identification of the entities involved, their legal roles, and the access control policies that they must satisfy. For activity (2), it checks whether access control policies are compliant with the provisions of the GDPR and computes the risk level of a data processing activity in terms of the likelihood and impact of a data breach. The ultimate goal of our tool-based methodology is to assist organizations to master the complexity and the interdisciplinary competencies needed for the correct implementation of the DPIA. Indeed, the effective use of the tool must be complemented by adequate training on the key notions of the GDPR and the DPIA.

In each one of the three steps in our methodology (shown in Figure 6.3), the user is assisted by a tool in gathering the necessary data to produce three documents containing the description of the DPIA activities. Such documents can be used to satisfy the accountability requirements of the GDPR (art. 5.2) and to support the auditing process by, e.g., a (national) privacy authority.

1. The step **Processing Analysis** outputs a document, called *Processing Specification*, that contains a precise description of the data processing activities, including the collected data, their classes, the data subject categories involved, the purpose, etc.
2. The step **Risk analysis** outputs a document, called *Risk summary*, that reports the compliance check of access control policies against the GDPR (this is crucial to ensure that data subjects can control the sharing of their personal data in compliance with legal provisions) and the risk levels associated to each defined data processing.

[14] https://ec.europa.eu/newsroom/document.cfm?doc_id=44137
[15] See resolution n. 450 of March 23, 2018 available at http://www.delibere.provincia.tn.it/.

3. The step **Run-time Analysis** outputs a document, called *Asset and Event Mapping*, that contains the associations between the assets (identified in the previous step of the methodology) and the actual entities in the system together with the events that are relevant for data protection so that an Inventory Management system and a Security Information and Event Management can use the associations to detect, at run-time, possible deviations from the protection profiles previously specified or violations of compliance.

6.4.3 Privacy Languages for Data Policies

Controlled Natural Languages - CNLs - are instruments that help write requirements, still without departing from the natural language playground which remains pivotal for humans to express themselves and communicate ideas. But differently from natural languages, CNLs use a controlled grammar, a precise semantics, and the possibility to process statements automatically. For instance, from requirements written in *Controlled Natural Language for Data Sharing Agreement* (CNL4DSA), the CNL we consider in this section, it is possible to generate access control policies and enforcement points.

Central to CNL4DSA is the notion of *fragment*, *i.e.*, a tuple $f = \langle s, a, o \rangle$, where s is the subject, a is the action, o is the object. A fragment simply says that 'subject s performs action a on object o'. By adding the $can/must/cannot$ constructs to the basic fragment, a fragment becomes an authorisation, an obligation, or a prohibition. Such *composite fragments* are by the following BNF-like syntax:

$$F := \textbf{NIL} \mid mod\ f \mid F;\ F \mid \textbf{IF}\ C\ \textbf{THEN}\ F \mid\ \textbf{AFTER}\ f\ \textbf{THEN}\ F$$

where:

- **NIL** is the null policy;
- *mod* ranges over {**CAN**, **MUST**, **CANNOT**} and models different type of policies and respectively, *authorization*, *obligation*, and *prohibition* policies;
- *mod* f is the atomic authorization/obligation/prohibition fragment that expresses that f $(= \langle s, a, o \rangle)$ is allowed/obliged/denied. Its informal meaning is that subject s can/must/cannot perform action a on object o.
- $F; F$ is a list of composite fragments.
- **IF** C **THEN** F expresses the logical implication between a composite context C (see later) and a composite fragment: if C holds, then F is permitted.
- **AFTER** f **THEN** F is a temporal sequence of fragments. Informally, after f has happened, then the composite fragment F is permitted.

Fragments are evaluated within a *context*. In CNL4DSA, a context c is evaluated as a boolean value (true/-false) and it asserts properties of subjects and objects, in terms, *e.g.*, of users' roles, data categories, time, and geographical location. Simple examples of contexts are 'subject hasRole Facebook_admin', or 'object hasCategory user_post'. The constructs linking subjects and objects to their values, like hasRole and hasCategory in the above examples, are called *predicates*. To describe complex policies, contexts can be combined using the boolean connectors *and*, *or*, and *not*. Specifically, *composite contexts* are defined as follows:

$$C\ := c \mid C\ \textbf{AND}\ C \mid C\ \textbf{OR}\ C \mid \textbf{NOT}\ c$$

6.4.3.1 CNL4DSA-based toolkit

A textual rule, either written in CNL4DSA or in natural language, is managed by a CNL4DSA-based toolkit, originally proposed in [312] and successively renewed. Initially comprising a CNL4DSA Authoring Tool, a CNL4DSA Policy Analyser, and a CNL4DSA Mapper Tool, the toolkit has recently been enriched with a translator from natural language rules to CNL4DSA rules, the NL2CNL translation tool [347].

- NL2CNL Translator: a user with no expertise of CNLs can edit rules in natural language (e.g., in English); with a minimal user's effort, the translator outputs the rules in CNL4DSA;
- CNL4DSA Authoring Tool: an author with expertise in CNLs can edit rules directly in CNL4DSA. The rules are constrained by CNL4DSA constructs and the terms in the rules come from specific vocabularies (ontologies);
- CNL4DSA Analyser: it analyses a set of CNL4DSA rules, detecting potential conflicts among them. In case a conflict is detected, a conflict solver strategy based on prioritisation of rules is put in place to correctly enforce the right rules;

- CNL4DSA Mapper: it translates the CNL4DSA rules into an enforceable language. The mapping process takes as input the analysed CNL4DSA rules, translates them in a XACML-like language [269], and combines all the rules in line with the predefined conflict solver strategies. The outcome of this tool is an enforceable policy. Such policy will be evaluated at each request to use the objects specified in the policy itself.

6.4.4 Privacy Policy Enforcement via Encryption Schemes

The most elementary way to enforce access control policy is to use standard encryption schemes, particularly public key encryption schemes. Suppose everybody in a system has a unique public/private key pair, and the public keys are properly managed by a PKI (i.e. public key infrastructure). Suppose a data controller wishes to permit a user with the public key pk to access data m. Then the data controller can encrypt m with pk and store the ciphertext on a public repository. Based on the security of the encryption scheme, only the user with the private key sk can decrypt the ciphertext and obtain m. Therefore, the policy is automatically enforced by the encryption scheme, so that there is no need for PEP in this case. It is clear that public key encryption schemes can enforce simple access control models such as access control list, but it not suitable for more complex ones such as role-based or attributed based models.

A more powerful encryption primitive is Identity Based Encryption (IBE) [60]. This primitive assumes a trusted authority, which will issue private keys for different identities. In practice, an identity can be any string, e.g. ”Female doctor at Hospital X, whose age is above 50.” Suppose Hospital X acts as the trusted third party and a data controller wish to send data to the above identity. Then the data controller can encrypt the data with this identity. Later on, if a doctor wishes to access the data, she needs to demonstrate she satisfies the conditions in the identity specification. If so, Hospital X can issue her a private key which allows her to recover the data. With this primitive, it is possible for the data controller to encrypt data for the future, e.g. it can specify a condition that the data can only be recovered after Jan 1, 2020 (this policy can be enforced by the Hospital X to only issue the key at this date). In comparison to the standard encryption schemes, IBE can enforce more complex access control policies, e.g. role-based policies.

An even more powerful encryption primitive is Attribute Based Encryption (ABE) schemes [167, 316]. This primitive can have two flavors, namely Key-Policy ABE and Ciphertext-Policy ABE. In KP-ABE, attributes are used to annotate the ciphertexts and predicates over these attributes are ascribed to users' secret keys, while in CP-ABE attributes are used to describe the users credentials and the predicate over these credentials are attached to the ciphertext by the encrypting party. In both cases, the policies can be very flexible and match those in reality. This primitive can well enforce Attribute Based Access Control (ABAC) policies.

In addition to these encryption schemes, another type of encryption primitive, namely homomorphic encryption schemes [161], will also be very useful in many scenarios. With this primitive, a data controller can keep the sensitive data in encrypted form, and send the ciphertext to the potential recipients. The recipients can perform the desired operations on the data and request the data controller to recover the computed result. Access control policies can be enforced by the data controller when it needs to decrypt the result for the recipient.

Chapter 7 Integration Roadmap

7.1 Integration challenges and review of the technologies

The final goal of the HAII-T Program is to develop an integrated orchestration framework and a toolkit that support the security-by-design approach for the II. Such a goal is far beyond the purpose of the individual roadmaps defined in Chapters 2–6. In Chapter 4 we identified the lifecycle of a generic II and we discussed how the Infrastructure as Code (IaC) paradigm can support it through all the stages from the initial design to the production and monitoring phase. For this reason, we plan to take advantage of the IaC approach in two ways. First we map each technology developed in the context of the HAII-T Program to one of the five stages of the lifecycle discussed above (see Figure 4.1). A preliminary mapping is summarized in Table 7.1 at Task level.

Integration approaches vary according to each phase of the lifecycle. Below, we sketch the integration process for each of them.

Design In this phase the integration is supported by the design language used to develop the blueprint. Following the IaC approach, we plan to adopt an extensible language. The language extension will include support for the technologies under integration. Furthermore, we will create a catalog of objects an their properties. For instance, the catalog will contain devices and software that were formally verified or tested.

Validation The extended design language will include specifications and models that support the validation process. In this case, the integration will amount to developing a toolkit to be added to the orchestration framework via a plugin technique.

Deployment This phase converts a blueprint into an actual infrastructure. At this stage, the integration will be ensured by the presence of objects that implement the elements appearing and validated in the blueprint.

Testing Similarly to validation, the testing phase will be implemented as a toolkit/test suite to be launched against the deployed infrastructure.

Runtime In this phase the integration will be granted by the presence of the monitoring technologies that have been include in the infrastructure. Data collected in this phase will be available though a shared format.

Task	Technology	Stage
T6.1	New IoT crypto primitives	Design
	Low-power, secure networks	Design
	Secure OS software supply chain	Runtime
T6.2	Defenses against data-oriented attacks though virtualization	Deployment
	Defenses against code-oriented attacks though hardware-based monitoring	Deployment/Runtime
	Defenses against fault injection attacks for present and future devices	Design
T6.3	Security orchestration framework	All
	Formal verification of protocols	Validation
	Formal evidence language	Design/Validation
	Security orchestration for Fog computing	All
	II key components classification	Design
T6.4	Intrusion detection	Runtime
	Fault and intrusion tolerance	Testing/Runtime
T6.5	Privacy-preserving management and regulation	Design/Testing
	Privacy-enhancing technologies	Design/Validation
	Privacy evaluation techniques and methods	Design/Validation

Table 7.1**:** Table of technologies investigated within HAII-T Program.

7.2 Use cases

7.2.1 Smart building

A Building Automation System (BAS) is a II responsible for the automation of *smart building*. A BAS consists of several integrated and interconnected subsystems, e.g., heating, lightening and video surveillance. Examples of smart buildings include automated production plants as well as smart homes.

A generic network schema for a smart building may resemble the one depicted in Figure 7.1. It consists of a segmented network where each segment hosts services and devices related to a specific task: *(i) Server* contains the internal services (i.e., not meant to be publicly accessible), *(ii) DMZ* contains the public services (i.e., exposed to the outside world), *(iii) IoT* connects field devices (i.e., sensors, actuators and controllers), and *(iv) Edge* is an instance of a segment under the control of a Fog node. These four networks lay behind a *firewall* protecting the perimeter of the smart building. In general, the firewall is intentionally left open toward the DMZ to allow remote connections. The infrastructure is connected to the public Internet through the backbone of the Internet service provider. Remote entities from the cloud interact with the applications running in the smart building. For instance, a remote client may access sensors and actuators, e.g., to control the heating system, and data generated by medical devices inside the smart building may feed a remote healthcare service.

7.2.2 xMP

As long as the orchestration framework is limited in what can be deployed on remote (or virtual) nodes, it will not be possible to deploy all of the presented approaches for hardening legacy components. Yet, assuming the framework becomes able to deploy hypervisors (and the necessary images for virtual machines) on remote nodes, the framework could also deploy systems that are hardened against data-oriented attacks (Section 2.2.1.1). This would enable strong protection against data-oriented attacks on a variety of Intel-based legacy systems with virtualization support. One application scenario is in the health, automotive, and avionic sector, in which the end nodes (e.g., the on-board computer in automotive) must conform to the highest standards. As such, xMP would help to protect selected critical data structures, such as the process' credentials and page tables that are otherwise prone to attacks.

Similarly, if the nodes had in their dispositions reconfigurable devices that worked in close cooperation with the processors, there would be possible to harden them also against code-oriented attacks (Section 2.2.2). Such a protection is of crucial importance in all application fields, but especially in the embedded domain, and in particular, in addition to those already mentioned, also in the field of automatic controls for industrial systems or Cyber-Physical Systems (CPS). Here, more than to protect critical data structures, defenses against code-oriented attacks can mitigate or block significant attacks to take control over the functionalities of the nodes and turn them to their own advantage or for damaging the system's owner. The FPGA-based technique presented

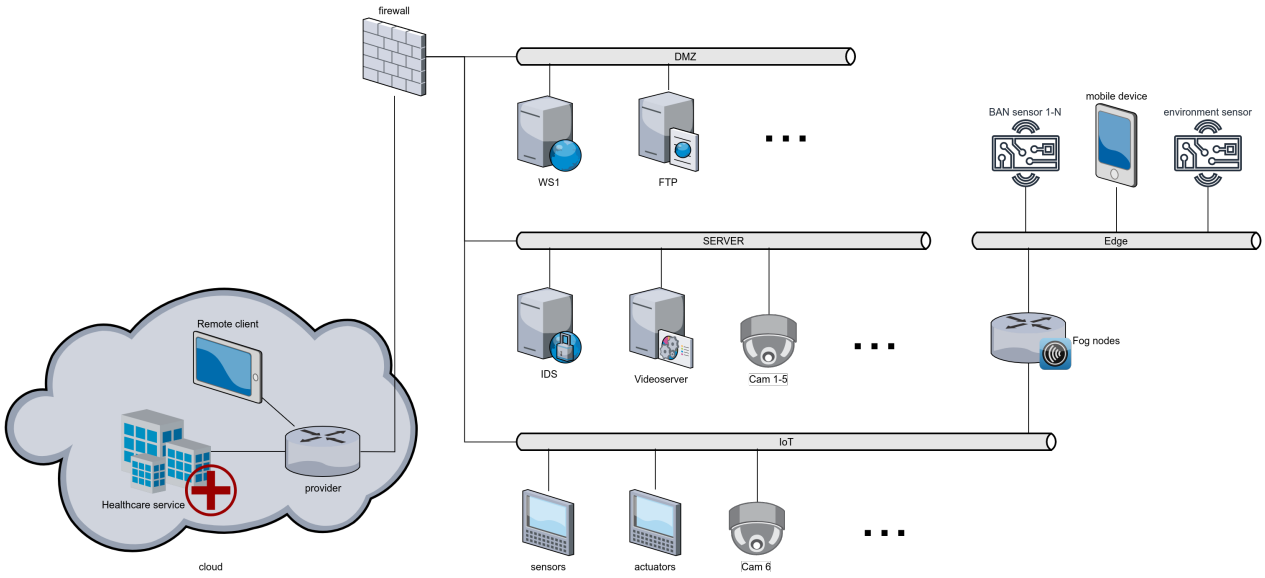


Figure 7.1: Smart building network scheme.

is well suitable to microcontrollers, and aims to protect them both when they are already in the field (*legacy*) and when they are still to be put into activity. In the first case, in fact, the external parallel interface they are provided with can be exploited to connect them to external reconfigurable devices; in the second case, instead, the monitor can be directly implemented on the FPGA placed inside the microcontroller, without entering straight leg in the design process to impose given security features within the core architecture.

7.2.3 Privacy-Preserving Internet of Vehicles

Internet of Vehicles (IoV) as the applications of the Internet of Things in intelligent transportation systems aim to improve many aspects in the transportation field from road safety, decreasing traffic jams, avoiding accidents, improving driving experiences and save fuel and travel times by finding optimal routes. IoV can be represented by a dynamic mobile communication system that communicates between vehicles and public networks using V2V (vehicle-to-vehicle), V2I (vehicle-to-infrastructure), V2H (vehicle-to-human) and V2S (vehicle-to-sensor). Thus, IoV as complex heterogeneous networks use various kinds of data resources such as vehicles, inter-vehicle sensors, road infrastructure sensors, and humans. Nevertheless, human data and driver behavior can cause several privacy issues that have to be addressed in a complex IoV environment. Therefore, IoV services have to deal with various privacy and security challenges and face various types of attacks. In the following list, we define main research subtopics related to privacy-preserving IoV:

- Conducting a general specification and analysis of privacy requirements for IoV services.
- Defining security risk management for vehicular networks.
- Exploring the impact of privacy-enhancing technologies in vehicular networks.
- Research in privacy-enhancing wireless communication for IoV.
- Research in general privacy-preserving communications for V2I, V2V, V2H and V2S (intra-vehicular).
- Research and development of privacy-enhancing access control systems based on attribute-based credentials for sharing vehicles, services, and for entering vehicles into city zones, low emission zones, parking lots/areas, etc.

7.3 Integration milestones, risks and mitigation strategy

In this section we present the main milestones of the integration process and we identify the risks that may compromise it. For each of them, we propose countermeasures to mitigate the risk of failure.

7.3.1 Milestones

We expect the following three milestones for the integration process.

List of technologies (M18) for this milestone we will accomplish a list of tools and techniques that will take part in the integration process and appear in the demonstration.

First demonstration (M24) At this stage, we plan to present a demonstration of the elements appearing in the list. The demonstration will present the technologies applied individually to a selected number of use cases.

Final demonstration (M36) The final milestone will consist of the integrated orchestration framework and toolkit applied to a selected use case. The use case will be based on one of the pilots of the project.

7.3.2 Risks and mitigation

There are several risks that might negatively impact on the integration process described above. In the first place, incompatibility issues could arise during the integration. Although this is unlikely to happen in theory, in practice existing tools may rely on technologies that are not designed to interact. In the worst case, this event can require, for instance, a re-engineering of existing tools. To prevent this possibility, we plan to identify suitable interfaces and interaction protocols at the first milestone (M18). Thus, by promptly standardizing the input/output interfaces, we will provide each partner with enough time check and ensure the compatibility of their technologies.

Another risk is related to the scalability of the presented techniques. In some cases, there is no actual evidence that purely theoretical methods can cope with real, large scale II. For this reason we will start with individual demonstrators to be applied to a limited number of shared use cases. Then, we will scale up the complexity of the use cases to test the feasibility of the techniques in an incremental way. In case a limitation is discovered for one of them, we will react by investigating alternative solutions and by studying the reasons behind the phenomenon.

Chapter 8 Conclusions

Implementing the security-by-design approach requires support to all the phases of the lifecycle of II. This document defined the roadmaps of the HAII-T Program. In particular, we discussed the state-of-the-art of the defense mechanisms for the legacy components as well as the security of the operating systems and software for the IoT and field devices. Then, we presented the technologies for the orchestration of complex, secure-by-design infrastructures. Moreover, we discussed the aspects related to the *resilience-by-design* and *privacy-by-design* approaches. The integration roadmap was also presented together with the use cases of the Program as well as the integration challenges and milestones. Below we report some concluding remarks specific to each chapter of this document.

Chapter 2

The presented preliminary results both challenged and increased security of legacy components and paved the way for future architectures. In fact, our research allows us to harden firmware and OSes on legacy systems against code-reuse and data-oriented attacks, respectively. By further focusing on fault injection attacks that leverage EM perturbations and laser against legacy hardware, we have identified ways that allow us to bypass security mechanisms. Consequently, this raised our attention towards software countermeasures against fault attacks, as well future directions towards general hardware-assisted software defenses that apply to legacy and open source architectures (i.e., RISC-V ISA).

Chapter 3

Better security for IoT software becomes more and more critical. In this context, it is to be expected that low-power IoT software will more and more mimic software elsewhere on the Internet, i.e. not only naturally network-oriented, but also assembled from a multitude of components of various origin and in large parts open-source. Securing low-power IoT operating system software in practice consists of combining and hardening a number of specific mechanisms at work at various levels of the system, namely (i) offline validation techniques, (ii) secure low-power networking mechanisms, and (iii) deeply embedded system software primitives.

Offline techniques and processes are obviously needed to validate the software a priori before it is deployed.

- Formal verification of critical parts of the IoT software can provide some guarantees that it is flawless (in some predetermined aspects). In this area, we plan to work on the secure compilation of IoT software and to focus more precisely on side-channel aware compilers, targeting crypto primitives.
- Security assessment of predetermined high-level policies can be carried out on the IoT software, which can provide some assurance that the device will not misbehave. In this area, we plan to work on verification mechanisms which can be applied on software update binary bundles, and which can detect security policy violations.

Nevertheless, however powerful the offline techniques may be, new bugs and new vulnerabilities are inevitably discovered down the line, which require patching the software a posteriori.

Low-power network mechanisms are thus needed to enable the secure deployment of embedded software updates on IoT devices in the field, often over low-power networks – because this is the only way to access the deployed devices.

- Secure networking protocols applicable to low-power communication media are needed to transfer data to/from the IoT devices. In this area, we plan to work on securing low-power communications at the transport and application layers, in close conjunction with IETF standardization in progress, which we plan to participate in.
- A framework for secure software update over-the-network is needed. This framework must be able to operate within the harsh constraints of low-end IoT in terms of embedded system resources and network throughput. In this area, we plan to work on top of the embedded operating system RIOT, to provide a general-purpose, evolutive open-source prototype.

Secure communication and secure software updates rely on a variety of lower-level primitives which provide some guarantees in terms of data confidentiality, integrity, or authenticity.

Embedded system primitives able to run efficiently on low-power IoT devices are thus needed within the embedded software itself.

- Low-level cryptographic primitives are needed to perform adequate cryptographic operations: to encrypt, to authenticate and check the integrity of software, and to provide strong guarantees. In this area, we

plan to develop public-key cryptosystems with smaller resource footprints, and to identipost-quantum candidate algorithms applicable in the IoT space

- Sandboxing primitives are necessary to isolate software components originating from, and maintained by, different entities. In this area, we plan to work on software-only approaches, which enable an equivalent of user space, which can work on low-end hardware that is inherently single address-space.

Chapter 4

Orchestrating an intelligent infrastracture is an extremely complex and multifaceted problem. Our review of the literature outlined that this is an active field of research and several proposals independently address some open issues of the II life cycle management process. However, a unified orchestration framework is still beyond the state of the art. The main reasons are (i) the heterogeneous nature of the technologies contributing to the II and (ii) the fact that existing techniques tackle some specific problems in isolation. To deal with these issues we sketched a roadmap for the definition of a unified orchestration framework for the II. The goal is to design and implement an orchestration framework that ensures the integration of existing as well as future techniques. This can be achieved through the adoption of the IaC paradigm that was already successfully applied to the problem of orchestrating cloud infrastructures.

Chapter 5

In the context of Task 6.4 ("resilience-by-design of intelligent infrastructures (II)"), in SPARTA, we first presented an overview of the main concepts and design principles relevant to Intrusion Tolerant (IT) architectures. The activities on intrusion detection conducted by the different partners focus on the analysis of the resulting flow that will be carried out on a single machine. Then, we provided a resilience techniques which are based on fault and intrusion tolerance techniques. The challenges put by looking at faults under the perspective of "malicious intelligence" have brought to the agenda hard issues such as uncertainty, adaptivity, incomplete knowledge, interference, and so forth. We believe that fault tolerance will witness an extraordinary evolution, which will have applicability in all fields and not only security-related ones.

Chapter 6

Intelligent Infrastructures integrating Internet of Things pose many challenges for the protection of privacy and the protection of users' personal data. Four basic aspects for privacy-by-design are overviewed in Chapter 6: privacy threats and attacks, privacy-preserving management and regulations, privacy-enhancing techniques, and privacy evaluation methods. Firstly, we overview the technical privacy threats, threats to private data publishing, privacy-based leakages, and social aspects. We mainly discuss threats to private data publishing and present the privacy leakages due to the poor design through the access control and GDPR perspectives. Based on the legal developments, this report also exposes a reasoning for privacy management and a method for managing GDPR compliance in business processes. The method helps explaining why and where organisational and technical countermeasures should be installed to mitigate private information leakage. Thirdly, we categorize 15 PETs and discuss their maturity and readiness for IoT. Several technologies, such as attribute-based credentials and group signatures, seem appropriate for constrained and heterogeneous environment such as IoT. Finally, we illustrate state-of-art techniques for privacy policy enforcement, and introduce a methodology and tool for Data Protection Impact Assessment.

As future work, task 6.5 will focus on selected use cases, e.g., Internet of Vehicles that contains many privacy challenges and privacy-requiring scenarios. Several promising technologies such as ABC will be further implemented into suitable scenarios in order to enhance user privacy. Other future challenges are mainly the detection of security risks, the definition of the personal data, the adoption of a user-centric approach in selected use cases, and the identification of other regulations that apply to IoT and IoV, and that influence the obligation of privacy-by-design.

Chapter 7

Although several, state-of-the-the techniques can deal with certain specific aspects of the security of the Intelligent Infrastructures, an integrated framework is yet to come. As a result of our preliminary study we identified the enabling technologies and we defined a roadmap for the integration of the methodologies developed in SPARTA. The key factor is a common infrastructure modeling language that takes into account all the phases of the life-cycle of an II and, at the same time, supports modularity though the definition of interfaces for the components to be integrated. Among the existing proposals, we identified TOSCA as a promising candidate. Finally, we presented the case studies that will serve for the evaluation of the integration process.

Chapter 9 List of Abbreviations

Abbreviation	Translation
IaC	Infrastructure as Code
BAS	Building Automation System
II	Intelligent Infrastructure
CPS	Cyber-Physical System
ICS	Industrial Control System
IoT	Internet of Things
IoV	Internet of Vehicles
CFI	Control-Flow Integrity
COTS	Commercial Of-The-Shelf
EM	Electromagnetic
EPT	Extended Page Table
FA	Fault Attack
FM	Fault Model
SLAT	Second Layer Address Translation
HMAC	Keyed-Hash Message Authentication Code

Chapter 10 Bibliography

[1] Aurum: A framework for information security risk management. In *Proceedings of the 42Nd Hawaii International Conference on System Sciences*, HICSS '09, pages 1–10, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 978-0-7695-3450-3. doi: 10.1109/HICSS.2009.82. URL http://dx.doi.org/10.1109/HICSS.2009.82.

[2] Top 10 Technologies That Will Drive the Future of Infrastructure and Operations. https://www.gartner.com/en/documents/3970841/top-10-technologies-that-will-drive-the-future-of-infras, 2019. [Online; accessed 06-December-2019].

[3] M. Abadi, M. Budiu, Ú. Erlingsson, and J. Ligatti. Control-flow integrity. In *Proceedings of the 12th ACM conference on Computer and communications security*, pages 340–353. ACM, 2005.

[4] A. Abbasi. GDPR Implementation in an Airline Contact Center. Master's thesis, University of Tartu, 2018.

[5] M. Abomhara and Koien. Security and Privacy in the Internet of Things: Current Status and Open Issues. In *International Conference on Privacy and Security in Mobile Systems (PRISMS)*, 2014.

[6] Francisco Javier Acosta Padilla, Emmanuel Baccelli, Thomas Eichinger, and Schleiser K. The Future of IoT Software Must be Updated. *IAB Workshop on Internet of Things Software Update (IoTSU)*, 2016.

[7] O. A-a. Affia, R. Matulevičius, and A. Nolte. Security risk management in cooperative intelligent transportation systems: A systematic literature review. In *Proceedings of CoopIS 2019*. Springer, 2019.

[8] Mehdi-Laurent Akkar, Louis Goubin, and Olivier Ly. Automatic integration of counter-measures against fault injection attacks. In *In Proceedings of E-Smart'2003*, 2003.

[9] F. A. Alabaa, M. Othma, I. Abaker, I. A. T. Hashem, and F. Alotaibib. Internet of Things security: A survey. *Journal of Network and Computer Applications*, 88(15):10–28, 2017.

[10] M. Alam, D. Roy, S. Bhattacharya, V. Govindan, R.S. Chakraborty, and D. Mukhopadhyay. Smashclean: A hardware level mitigation to stack smashing attacks in openrisc. In *2016 ACM/IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE)*, pages 1–4. IEEE, 2016.

[11] Réka Albert, Hawoong Jeong, and Albert-László Barabási. Error and attack tolerance of complex networks. *Nature*, 406(6794):378, 2000.

[12] Martin Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi, Jeffrey Hoffstein, Kim Laine, Kristin Lauter, Satya Lokam, Daniele Micciancio, Dustin Moody, Travis Morrison, Amit Sahai, and Vinod Vaikuntanathan. Homomorphic encryption security standard. Technical report, HomomorphicEncryption.org, Toronto, Canada, November 2018.

[13] S. Alexander. Defeating compiler-level buffer overflow protection. *The USENIX Magazine; login*, 2005.

[14] Andreea B. Alexandru and George J. Pappas. Secure Multi-party Computation for Cloud-based Control. *arXiv e-prints*, art. arXiv:1906.09652, Jun 2019.

[15] Muhammad Irfan Ali, B. Davvaz, and Muhammad Shabir. Some properties of generalized rough sets. *Information Sciences*, 224:170–179, mar 2013. doi: 10.1016/j.ins.2012.10.026. URL https://doi.org/10.1016%2Fj.ins.2012.10.026.

[16] Haitham Bou Ammar, Rasul Tutunov, and Eric Eaton. Safe policy search for lifelong reinforcement learning with sublinear regret. In *International Conference on Machine Learning*, pages 2361–2369, 2015.

[17] M. Ammar, G. Russello, and Crispo B. Internet of Things: A survey on the security of IoT frameworks. *Journal of Information Security and Applications*, 38:8–27, 2018.

[18] Tomasz Andrysiak, Łukasz Saganowski, Michał Choraś, and Rafał Kozik. Network traffic prediction and anomaly detection based on ARFIMA model. In *Advances in Intelligent Systems and Computing*, pages 545–554. Springer International Publishing, 2014. doi: 10.1007/978-3-319-07995-0_54. URL https://doi.org/10.1007%2F978-3-319-07995-0_54.

[19] D. W Archer, D. Bogdanov, Y. Lindell, L. Kamm, K. Nielsen, J. I. Pagter, N. P Smart, and R. N Wright. From Keys to Databases—Real-World Applications of Secure Multi-Party Computation. *The Computer Journal*, 61(12):1749–1771, 09 2018. doi: 10.1093/comjnl/bxy090.

[20] ARIA TOSCA. Apache ARIA TOSCA orchestration engine. http://ariatosca.incubator.apache.org/, 2019. (Accessed on October 2019).

[21] Alessandro Armando, David Basin, Yohan Boichut, Yannick Chevalier, Luca Compagna, Jorge Cuéllar, P Hankes Drielsma, Pierre-Cyrille Héam, Olga Kouchnarenko, Jacopo Mantovani, et al. The avispa tool for the automated validation of internet security protocols and applications. In *International conference on computer aided verification*, pages 281–285. Springer, 2005.

[22] Robert C Armstrong, Ratish J Punnoose, Matthew H Wong, and Jackson R Mayo. Survey of existing tools for formal verification. *SANDIA REPORT SAND2014-20533*, 2014.

[23] Matej Artac, Tadej Borovsak, Elisabetta Di Nitto, Michele Guerriero, and Damian Andrew Tamburri. Devops: Introducing infrastructure-as-code. *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, pages 497–498, 2017.

[24] Ali Assaf, Guillaume Burel, Raphaël Cauderlier, David Delahaye, Gilles Dowek, Catherine Dubois, Frédéric Gilbert, Pierre Halmagrand, Olivier Hermant, and Ronan Saillard. Dedukti : a logical framework based on the $\lambda\pi$-calculus modulo theory. 2016.

[25] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The Internet of Things: A Survey. *Computer networks*, 54(15):2787–2805, 2010.

[26] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, et al. *Fundamental concepts of dependability*. University of Newcastle upon Tyne, Computing Science, 2001.

[27] Josep Balasch, Benedikt Gierlichs, and Ingrid Verbauwhede. An in-depth and black-box characterization of the effects of clock glitches on 8-bit MCUs. In *Fault Diagnosis and Tolerance in Cryptography*, 2011.

[28] Maria-Florina Balcan, Avrim Blum, and Santosh Vempala. Efficient representations for lifelong learning and autoencoding. In *Conference on Learning Theory*, pages 191–210, 2015.

[29] M. Banerjee, J. Lee, and K.-K. R. Choo. A blockchain future to Internet of Things security: A position paper. *Digital Communications and Networks*, 2018.

[30] H. Bar-El, H. Choukri, D. Naccache, Michael Tunstall, and C. Whelan. The sorcerer's apprentice guide to fault attacks. *Proceedings of the IEEE*, 94(2):370–382, Feb 2006.

[31] Michel Barbeau and Joaquin Garcia-Alfaro. Faking and discriminating the navigation data of a micro aerial vehicle using quantum generative adversarial networks. In *IEEE GLOBECOM 2019 Workshop on Quantum Communications and Information Technology 2019 (5th QCIT workshop of the Emerging Technical Committee on Quantum Communications and Information Technology)*, 2019.

[32] Alessandro Barenghi, Luca Breveglieri, Israel Koren, Gerardo Pelosi, and Francesco Regazzoni. Countermeasures against fault attacks on software implemented aes: Effectiveness and cost. In *Proceedings of the 5th Workshop on Embedded Systems Security*, WESS '10, pages 7:1–7:10, New York, NY, USA, 2010. ACM.

[33] Alessandro Barenghi, Luca Breveglieri, Israel Koren, and David Naccache. Fault injection attacks on cryptographic devices: Theory, practice, and countermeasures. *Proceedings of the IEEE*, 100:3056 – 3076, 2012.

[34] A. Bargiela and W. Pedrycz. The roots of granular computing. In *2006 IEEE International Conference on Granular Computing*. IEEE, 2006. doi: 10.1109/grc.2006.1635922. URL https://doi.org/10.1109%2Fgrc.2006.1635922.

[35] Thierno Barry, Damien Couroussé, and Bruno Robisson. Compilation of a countermeasure against instruction-skip fault attacks. In *Proceedings of the Third Workshop on Cryptography and Security in Computing Systems*, CS2 '16, pages 1–6, New York, NY, USA, 2016. ACM.

[36] Gilles Barthe, Lennart Beringer, Pierre Crégut, Benjamin Grégoire, Martin Hofmann, Peter Müller, Erik Poll, Germán Puebla, Ian Stark, and Eric Vétillard. Mobius: Mobility, ubiquity, security. In Ugo Montanari, Donald Sannella, and Roberto Bruni, editors, *Trustworthy Global Computing*, pages 10–29, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[37] Gilles Barthe, Benjamin Grégoire, and Vincent Laporte. Secure compilation of side-channel countermeasures: The case of cryptographic "constant-time". In *CSF*, pages 328–343. IEEE Computer Society, 2018.

[38] David Basin, Cas Cremers, Jannik Dreier, and Ralf Sasse. Symbolically analyzing security protocols using tamarin. *ACM SIGLOG News*, 2017.

[39] David Basin, Cas Cremers, and Catherine Meadows. Model checking security protocols. In *Handbook of Model Checking*, pages 727–762. Springer, 2018.

[40] J. Baxter. A model of inductive bias learning. *Journal of Artificial Intelligence Research*, 12:149–198, mar 2000. doi: 10.1613/jair.731. URL https://doi.org/10.1613%2Fjair.731.

[41] Anja Bechmann. Internet profiling: the economy of data intraoperability on Facebook and Google. *Journal of media and communication research*, 29(55):19, 2013. ISSN 1901-9726.

[42] Arthur Beckers, Josep Balasch, Benedikt Gierlichs, Ingrid Verbauwhede, Saki Osuka, Masahiro Kinugawa, Daisuke Fujimoto, and Yuichi Hayashi. Characterization of EM faults on ATmega328P. In *International Symposium on Electromagnetic Compatibility*. IEEE, 2019.

[43] Gerd Behrmann, Alexandre David, and Kim G Larsen. A tutorial on uppaal 4.0. *Department of computer science, Aalborg university*, 2006.

[44] Johan Bengtsson, Kim Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. Uppaal—a tool suite for automatic verification of real-time systems. In *International hybrid systems workshop*, pages 232–243. Springer, 1995.

[45] Daniel J. Bernstein, Tanja Lange, and Peter Schwabe. The security impact of a new cryptographic library. In *LATINCRYPT 2012*, volume 7533 of *LNCS*, pages 159–176. Springer, 2012.

[46] Ivan Beschastnikh, Yuriy Brun, Michael D Ernst, Arvind Krishnamurthy, and Thomas E Anderson. Mining temporal invariants from partially ordered logs. *ACM SIGOPS Operating Systems Review*, 45(3):39–46, 2012.

[47] Ivan Beschastnikh, Yuriy Brun, Michael D Ernst, and Arvind Krishnamurthy. Inferring models of concurrent systems from logs of their behavior with CSight. In *Int. Conf. on Software Engineering*, pages 468–479, 2014.

[48] Frédéric Besson, Alexandre Dang, and Thomas P. Jensen. Information-flow preservation in compiler optimisations. In *CSF*, pages 230–242. IEEE, 2019.

[49] J. Bethencourt, A. Sahai, and B. Waters. Ciphertext-policy attribute-based encryption. In *2007 IEEE Symposium on Security and Privacy (SP '07)*, pages 321–334, May 2007.

[50] S. Bhatkar, D. DuVarney C, and R. Sekar. Address obfuscation: An efficient approach to combat a broad range of memory error exploits. In *USENIX Security Symposium*, volume 12, pages 291–301, 2003.

[51] Bruno Blanchet. Automatic verification of security protocols in the symbolic model: The verifier proverif. In *Foundations of Security Analysis and Design VII*, pages 54–87. Springer, 2013.

[52] Bruno Blanchet, Ben Smyth, Vincent Cheval, and Marc Sylvestre. Proverif 2.00: Automatic cryptographic protocol verifier, user manual and tutorial, 2018.

[53] Bruno Blanchet et al. An efficient cryptographic protocol verifier based on prolog rules. In *csfw*, volume 1, pages 82–96, 2001.

[54] Bruno Blanchet et al. Modeling and verifying security protocols with the applied pi calculus and proverif. *Foundations and Trends® in Privacy and Security*, 1(1-2):1–135, 2016.

[55] Olivier Blazy, Philippe Gaborit, Julien Schrek, and Nicolas Sendrier. A code-based blind signature. In *2017 IEEE International Symposium on Information Theory (ISIT)*, pages 2718–2722. IEEE, 2017.

[56] T. Bletsch, X. Jiang, and V. Freeh. Mitigating code-reuse attacks with control-flow locking. In *Proceedings of the 27th Annual Computer Security Applications Conference*, pages 353–362. ACM, 2011.

[57] T. Bletsch, X. Jiang, V. W. Freeh, and Z. Liang. Jump-oriented programming: a new class of code-reuse attack. In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, pages 30–40. ACM, 2011.

[58] Rakesh B Bobba, Katherine M Rogers Qiyan Wang, Himanshu Khurana, Klara Nahtstedt, and Thomas J Overbye. Detecting False Data Injection Attacks on DC State Estimation. In *1st Workshop on Secure Control Systems (CPSWEEK)*, pages 1–9, April 2010.

[59] Barbara Bobowska, Michal Choras, and Michal Wozniak. Advanced analysis of data streams for critical infrastructures protection and cybersecurity. *J. UCS*, 24(5):622–633, 2018.

[60] Dan Boneh and Matthew Franklin. Identity-based encryption from the weil pairing. *SIAM J. Comput.*, 32 (3):586–615, March 2003.

[61] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pages 13–16. ACM, 2012.

[62] Samarjeet Borah, Ranjit Panigrahi, and Anindita Chakraborty. An enhanced intrusion detection system based on clustering. In *Advances in Intelligent Systems and Computing*, pages 37–45. Springer Singapore, dec 2017. doi: 10.1007/978-981-10-6875-1_5. URL https://doi.org/10.1007%2F978-981-10-6875-1_5.

[63] Carsten Bormann, Mehmet Ersue, and Ari Keränen. *Terminology for Constrained-Node Networks*. Number 7228 in Request for Comments. RFC Editor, May 2014. doi: 10.17487/RFC7228. URL https://rfc-editor.org/rfc/rfc7228.txt. Published: RFC 7228.

[64] Yazan Boshmaf, Ildar Muslukhov, Konstantin Beznosov, and Matei Ripeanu. The socialbot network: when bots socialize for fame and money. In *Proceedings of the 27th Annual Computer Security Applications Conference (ACSAC'11)*, pages 93–102. ACM, 2011.

[65] Jakub Breier and Dirmanto Jap. Testing feasibility of back-side laser fault injection on a microcontroller. In *Proceedings of the WESS'15: Workshop on Embedded Systems Security*, New York, NY, USA, 2015.

[66] Jakub Breier, Dirmanto Jap, and Chien-Ning Chen. Laser profiling for the back-side fault attacks: With a practical laser skip instruction attack on AES. In *Proceedings of the 1st ACM Workshop on Cyber-Physical System Security*, New York, NY, USA, 2015.

[67] C. Bresch, A. Michelet, L. Amato, T. Meyer, and D. Hely. A red team blue team approach towards a secure processor design with hardware shadow stack. In *2017 IEEE 2nd International Verification and Security Workshop (IVSW)*, pages 57–62, July 2017. doi: 10.1109/IVSW.2017.8031545.

[68] C. Bresch, D. Hély, A. Papadimitriou, A. Michelet-Gignoux, L. Amato, and T. Meyer. Stack redundancy to thwart return oriented programming in embedded systems. *IEEE Embedded Systems Letters*, 10(3): 87–90, Sep. 2018. ISSN 1943-0663. doi: 10.1109/LES.2018.2819983.

[69] M. S. de Brito, S. Hoque, T. Magedanz, R. Steinke, A. Willner, D. Nehls, O. Keils, and F. Schreiner. A service orchestration architecture for Fog-enabled infrastructures. In *2017 Second International Conference on Fog and Mobile Edge Computing (FMEC)*, pages 127–132, May 2017. doi: 10.1109/FMEC.2017.7946419.

[70] Antonio Brogi and Jacopo Soldani. Reusing cloud-based services with tosca. In E. Plödereder, L. Grunske, E. Schneider, and D. Ull, editors, *Informatik 2014*, pages 235–246, Bonn, 2014. Gesellschaft für Informatik e.V.

[71] Sean Brooks, Sean Brooks, Michael Garcia, Naomi Lefkovitz, Suzanne Lightman, and Ellen Nadeau. *An introduction to privacy engineering and risk management in federal systems*. US Department of Commerce, National Institute of Standards and Technology, 2017.

[72] B Brumback and M Srinath. A chi-square test for fault-detection in Kalman filters. *IEEE Transactions on Automatic Control*, 32(6):552–554, Jun 1987. ISSN 0018-9286. doi: 10.1109/TAC.1987.1104658.

[73] David Brumley and Dan Boneh. Remote timing attacks are practical. In *USENIX Security Symposium*. USENIX Association, 2003.

[74] Andrea Burattin, Giuseppe Cascavilla, and Mauro Conti. Socialspy: Browsing (supposedly) hidden information in online social networks. *CoRR*, abs/1406.3216, 2014.

[75] Jan Camenisch, Manu Drijvers, and Jan Hajny. Scalable revocation scheme for anonymous credentials based on n-times unlinkable proofs. In *Proceedings of the 2016 ACM on Workshop on Privacy in the Electronic Society*, pages 123–133. ACM, 2016.

[76] Jan Camenisch, Manu Drijvers, Petr Dzurenda, and Jan Hajny. Fast keyed-verification anonymous credentials on standard smart cards. In *IFIP International Conference on ICT Systems Security and Privacy Protection*, pages 286–298. Springer, 2019.

[77] Jan L Camenisch, Jean-Marc Piveteau, and Markus A Stadler. Blind signatures based on the discrete logarithm problem. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 428–432. Springer, 1994.

[78] A. Campan, T. Marius Truta, and N. Cooper. P-sensitive k-anonymity with generalization constraints. *Trans. Data Privacy*, 3(2):65–89, 2010.

[79] Andrew T. Campbell, Irene Katzela, Kazuho Miki, and John Vicente. Open signaling for atm, internet and mobile networks (opensig'98). *SIGCOMM Comput. Commun. Rev.*, 29(1):97–108, January 1999. ISSN 0146-4833. doi: 10.1145/505754.505762. URL http://doi.acm.org/10.1145/505754.505762.

[80] Alvaro Cardenas, Saurabh Amin, Zong-Syun Lin, Yu-Lun Huang, Chi-Yen Huang, and Shankar Sastry. Attacks Against Process Control Systems: Risk Assessment, Detection, and Response. In *6th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '11, pages 355–366, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0564-8. doi: 10.1145/1966913.1966959.

[81] Alvaro A. Cardenas, Saurabh Amin, and Shankar Sastry. Secure control: Towards survivable cyber-physical systems. In *2008 The 28th International Conference on Distributed Computing Systems Workshops*. IEEE, jun 2008. doi: 10.1109/icdcs.workshops.2008.40. URL https://doi.org/10.1109%2Ficdcs.workshops.2008.40.

[82] N. Carlini and D. Wagner. Rop is still dangerous: Breaking modern defenses. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 385–399, 2014.

[83] Giuseppe Cascavilla, Mauro Conti, and Inbal Yahav Davide G. Schwartz. Revealing Censored Information Through Comments and Commenters in Online Social Networks. In *IEEE/ACM Advances in Social Networks Analysis and Mining (ASONAM 2015)*, 2015.

[84] Claude Castelluccia. Behavioural Tracking on the Internet: a Technical Perspective. In *European Data Protection: In Good Health?*, pages 21–33. Springer, Dordrecht, 2012. ISBN 978-94-007-2902-5.

[85] Claude Castelluccia, Mohamed-Ali Kaafar, and Minh-Dung Tran. Betrayed by Your Ads!: Reconstructing User Profiles from Targeted Ads. In *Proceedings of Privacy Enhancing Technologies*, pages 1–17. Springer-Verlag, 2012. ISBN 978-3-642-31679-1.

[86] Seyit Ahmet Çamtepe and Bülent Yener. Modeling and detection of complex attacks. In *Proc. of the 3rd Int. Conf. on Security and Privacy in Communications Networks*, pages 234–243, 2007. doi: 10.1109/SECCOM.2007.4550338.

[87] I. Çelebi. Privacy Enhanced Secure Tropos: A Privacy Modeling Language for GDPR Compliance. Master's thesis, University of Tartu, 2018.

[88] A. Chaudhari and J. A. Abraham. Effective control flow integrity checks for intrusion detection. In *2018 IEEE 24th International Symposium on On-Line Testing And Robust System Design (IOLTS)*, pages 1–6, July 2018. doi: 10.1109/IOLTS.2018.8474130.

[89] David Chaum, Debajyoti Das, Farid Javani, Aniket Kate, Anna Krasnova, Joeri De Ruiter, and Alan T Sherman. cmix: Mixing with minimal real-time asymmetric cryptographic operations. In *International Conference on Applied Cryptography and Network Security*, pages 557–578. Springer, 2017.

[90] P. Chen, H. Xiao, X. Shen, X. Yin, B. Mao, and L. Xie. Drop: Detecting return-oriented programming malicious code. In A. Prakash and I. Sen Gupta, editors, *Information Systems Security*, pages 163–177, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. ISBN 978-3-642-10772-6.

[91] P. Chen, X. Xing, B. Mao, L. Xie, X. Shen, and X. Yin. Automatic construction of jump-oriented programming shellcode (on the x86). In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, pages 20–29. ACM, 2011.

[92] Po-Yu Chen, Shusen Yang, Julie A McCann, Jie Lin, and Xinyu Yang. Detection of false data injection attacks in smart-grid systems. *IEEE Communications Magazine*, 53(2):206–213, 2015.

[93] Quan Chen, Ahmed M. Azab, Guruprasad Ganesh, and Peng Ning. PrivWatcher: Non-bypassable Monitoring and Protection of Process Credentials from Memory Corruption Attacks. In *asiaccs*, 2017.

[94] S. Chen, J. Xu, N. Nakka, Z. Kalbarczyk, and R. K. Iyer. Defeating memory corruption attacks via pointer taintedness detection. In *2005 International Conference on Dependable Systems and Networks (DSN'05)*, pages 378–387, June 2005. doi: 10.1109/DSN.2005.36.

[95] Yuan Chen, Soummya Kar, and Jose M. F. Moura. Dynamic Attack Detection in Cyber-Physical Systems with Side Initial State Information. *IEEE Transactions on Automatic Control*, PP(99):1–1, 2016. ISSN 0018-9286. doi: 10.1109/TAC.2016.2626267.

[96] Z Chen and B Liu. Lifelong machine learning in the big data era. In *24th International Joint Conference on Artificial Intelligence*. IJCAI, 2015.

[97] Zhiyuan Chen and Bing Liu. Topic modeling using topics from many domains, lifelong learning and big data. In *International Conference on Machine Learning*, pages 703–711, 2014.

[98] Y. Cheng, Z. Zhou, Y. Miao, X. Ding, and R. H. Deng. ROPecker: A generic and practical approach for defending against ROP attackw. 2014.

[99] Michał Choraś and Rafał Kozik. Network event correlation and semantic reasoning for federated networks protection system. In *Communications in Computer and Information Science*, pages 48–54. Springer Berlin Heidelberg, 2011. doi: 10.1007/978-3-642-27245-5_8. URL https://doi.org/10.1007%2F978-3-642-27245-5_8.

[100] Michał Choraś, Rafał Kozik, Rafał Piotrowski, Juliusz Brzostek, and Witold Hołubowicz. Network events correlation for federated networks protection system. In *Towards a Service-Based Internet*, pages 100–111. Springer Berlin Heidelberg, 2011. doi: 10.1007/978-3-642-24755-2_9. URL https://doi.org/10.1007%2F978-3-642-24755-2_9.

[101] Michał Choraś, Rafał Kozik, Damian Puchalski, and Witold Hołubowicz. Correlation approach for SQL injection attacks detection. In *Advances in Intelligent Systems and Computing*, pages 177–185. Springer Berlin Heidelberg, 2013. doi: 10.1007/978-3-642-33018-6_18. URL https://doi.org/10.1007%2F978-3-642-33018-6_18.

[102] N. Christoulakis, G. Christou, E. Athanasopoulos, and S. Ioannidis. Hcfi: Hardware-enforced control-flow integrity. In *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy*, pages 38–49. ACM, 2016.

[103] Zi Chu, Steven Gianvecchio, Haining Wang, and Sushil Jajodia. Detecting automation of Twitter accounts: Are you a human, bot, or cyborg? *IEEE Transactions on Dependable and Secure Computing*, 9 (6):811–824, 2012.

[104] C. Clifton and T. Tassa. On syntactic anonymity and differential privacy. *Trans. Data Privacy*, 6(2): 161–183, 2013.

[105] Brice Colombier, Alexandre Menu, Jean-Max Dutertre, Pierre-Alain Moëllic, Jean-Baptiste Rigaud, and Jean-Luc Danger. Laser-induced single-bit faults in flash memory: Instructions corruption on a 32-bit microcontroller. In *Hardware-Oriented Security and Trust*, 2019.

[106] Mauro Conti, Vittoria Cozza, Marinella Petrocchi, and Angelo Spognardi. TRAP: using targeted ads to unveil google personal profiles. In *2015 IEEE International Workshop on Information Forensics and Security, WIFS 2015, Roma, Italy, November 16-19, 2015*, pages 1–6, 2015. doi: 10.1109/WIFS.2015.7368607. URL https://doi.org/10.1109/WIFS.2015.7368607.

[107] Ştefan Conţiu and Adrian Groza. Improving remote sensing crop classification by argumentation-based conflict resolution in ensemble learning. *Expert Systems with Applications*, 64:269–286, dec 2016. doi: 10.1016/j.eswa.2016.07.037. URL https://doi.org/10.1016%2Fj.eswa.2016.07.037.

[108] Geoff Coulson, Gordon S Blair, Antônio Tadeu Gomes, Ackbar Joolia, Kevin Lee, Jo Ueyama, and Invin Ye. Reflective middleware-based programmable networking. In *The 2nd International Workshop on Reflective and Adaptive Middleware*, pages 115–119, 2003.

[109] C. Cowan, C. Pu, D. Maier, J. Walpole, P. Bakke, S. Beattie, A. Grier, P. Wagle, Q. Zhang, , and H. Hinton. Stackguard: Automatic adaptive detection and prevention of buffer-overflow attacks. 98:5–5, 01 1998.

[110] Vittoria Cozza, Van Tien Hoang, Marinella Petrocchi, and Angelo Spognardi. Experimental measures of news personalization in google news. In Sven Casteleyn, Peter Dolog, and Cesare Pautasso, editors, *Current Trends in Web Engineering*, Cham, 2016. Springer International Publishing. ISBN 978-3-319-46963-8.

[111] Cas Cremers. Scyther user manual. *Department of Computer Science, University of Oxford: Oxford, UK*, 32, 2014.

[112] Cas J. F. Cremers. The scyther tool: Verification, falsification, and analysis of security protocols. In Aarti Gupta and Sharad Malik, editors, *Computer Aided Verification*, pages 414–418, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. ISBN 978-3-540-70545-1.

[113] Stefano Cresci, Roberto Di Pietro, Marinella Petrocchi, Angelo Spognardi, and Maurizio Tesconi. Fame for sale: Efficient detection of fake Twitter followers. *Decision Support Systems*, 80:56–71, 2015.

[114] Mathieu Cunche. I know your MAC address: targeted tracking of individual using Wi-Fi. *Journal of Computer Virology and Hacking Techniques, 10(4): 219–227*, December 2013. doi: 10.1007/s11416-013-0196-1.

[115] Mathieu Cunche, Mohamed Ali Kaafar, and Roksana Boreli. Linking wireless devices using information contained in Wi-Fi probe requests. *Pervasive and Mobile Computing*, pages 56–69, 2013. doi: 10.1016/j.pmcj.2013.04.001.

[116] D. Hardt. The OAuth 2.0 Authorization Framework. RFC 6749, 2012. URL https://tools.ietf.org/html/rfc6749.

[117] György Dán and Henrik Sandberg. Stealth Attacks and Protection Schemes for State Estimators in Power Systems. In *Smart Grid Communications (SmartGridComm), 2010 First IEEE International Conference on*, pages 214–219, Oct 2010. doi: 10.1109/SMARTGRID.2010.5622046.

[118] George Danezis, Josep Domingo-Ferrer, Marit Hansen, Jaap-Henk Hoepman, Daniel Le Metayer, Rodica Tirtea, and Stefan Schiffner. Privacy and data protection by design-from policy to engineering. *arXiv preprint arXiv:1501.03726*, 2015.

[119] J. Danger, A. Facon, S. Guilley, K. Heydemann, U. Kühne, A. Si Merabet, and M. Timbert. Ccfi-cache: A transparent and flexible hardware protection for code and control-flow integrity. In *2018 21st Euromicro Conference on Digital System Design (DSD)*, pages 529–536, Aug 2018. doi: 10.1109/DSD.2018.00093.

[120] L. Davi, A. Sadeghi, D. Lehmann, and F. Monrose. Stitching the gadgets: On the ineffectiveness of coarse-grained control-flow integrity protection. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 401–416, 2014.

[121] L. Davi, M. Hanreich, D. Paul, A.R. Sadeghi, P. Koeberl, D. Sullivan, O. Arias, and Y. Jin. Hafix: hardware-assisted flow integrity extension. In *Proceedings of the 52nd Annual Design Automation Conference*, page 74. ACM, 2015.

[122] Lucas Davi, David Gens, Christopher Liebchen, and Ahmad-Reza Sadeghi. PT-Rand: Practical Mitigation of Data-only Attacks against Page Tables. In *ndss*, 2017.

[123] A. De, A. Basu, S. Ghosh, and T. Jaeger. Fixer: Flow integrity extensions for embedded risc-v. In *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 348–353, March 2019. doi: 10.23919/DATE.2019.8714980.

[124] Cécile De Terwangne and Karen Rosier. *Le Règlement Général sur la Protection des Données (RGPD/GDPR): analyse approfondie*. Number 44 in Collection du CRIDS. Larcier, 2018.

[125] Ashmita Debnath, Pradheepkumar Singaravelu, and Shekhar Verma. Privacy in wireless sensor networks using ring signature. *Journal of King Saud University-Computer and Information Sciences*, 26(2): 228–236, 2014.

[126] Mina Deng, Kim Wuyts, Riccardo Scandariato, Bart Preneel, and Wouter Joosen. A privacy threat analysis framework: supporting the elicitation and fulfillment of privacy requirements. *Requirements Engineering*, 16(1):3–32, Mar 2011. ISSN 1432-010X. doi: 10.1007/s00766-010-0115-7. URL https://doi.org/10.1007/s00766-010-0115-7.

[127] G. Dessouky, S. Zeitouni, T. Nyman, A. Paverd, L. Davi, P. Koeberl, N. Asokan, and A. Sadeghi. Lo-fat: Low-overhead control flow attestation in hardware. In *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, June 2017. doi: 10.1145/3061639.3062276.

[128] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. Technical report, Naval Research Lab Washington DC, 2004.

[129] Youcef Djenouri, Asma Belhadi, Jerry Chun-Wei Lin, and Alberto Cano. Adapted k-nearest neighbors for detecting anomalies on spatio–temporal traffic flow. *IEEE Access*, 7:10015–10027, 2019. doi: 10.1109/access.2019.2891933. URL https://doi.org/10.1109%2Faccess.2019.2891933.

[130] John E Dobson and Brian Randell. Building reliable secure computing systems out of unreliable insecure components. In *Seventeenth Annual Computer Security Applications Conference*, pages 164–173. IEEE, 2001.

[131] Krishna Doddapaneni, Ravi Lakkundi, Suhas Rao, Sujay Kulkarni, and Bhargav Bhat. Secure fota object for iot. In *2017 IEEE 42nd Conference on Local Computer Networks Workshops (LCN Workshops)*, 2017. doi: 10.1109/LCN.Workshops.2017.78.

[132] J. Domingo-Ferrer and V. Torra. A critique of k-anonymity and some of its enhancements. In *2008 Third International Conference on Availability, Reliability and Security*, pages 990–993, 2008.

[133] J. Domingo-Ferrer, S. Ricci, and J. Soria-Comas. A methodology to compare anonymization methods regarding their risk-utility trade-off. In *Modeling Decisions for Artificial Intelligence - 14th International Conference*, pages 132–143, 2017.

[134] Josep Domingo-Ferrer, Sara Ricci, and Carles Domingo-Enrich. Outsourcing scalar products and matrix products on privacy-protected unencrypted data stored in untrusted clouds. *Information Sciences*, 436: 320–342, 2018.

[135] Vijay D'Silva, Mathias Payer, and Dawn Xiaodong Song. The correctness-security gap in compiler optimization. In *SPW 2015*, pages 73–87. IEEE Computer Society, 2015.

[136] W. Du and M. J. Atallah. Secure Multi-party Computation Problems and Their Applications: A Review and Open Problems. In *Proceedings of the 2001 Workshop on New Security Paradigms*, NSPW '01, pages 13–22, 2001.

[137] Louis Dureuil, Guillaume Petiot, Marie-Laure Potet, Thanh-Ha Le, Aude Crohen, and Philippe de Choudens. FISSC: A fault injection and simulation secure collection. In *International Conference on Computer Safety, Reliability, and Security*, 2016.

[138] Jean-Max Dutertre, Timothé Riom, Olivier Potin, and Jean-Baptiste Rigaud. Experimental analysis of the laser-induced instruction skip fault model. In Aslan Askarov, René Rydhof Hansen, and Willard Rafnsson, editors, *The 24th Nordic Conference on Secure IT Systems, Nordsec 2019*, pages 221–237, Cham, 2019. Springer International Publishing.

[139] Ashutosh Dhar Dwivedi, Gautam Srivastava, Shalini Dhar, and Rajani Singh. A decentralized privacy-preserving healthcare blockchain for iot. *Sensors*, 19(2):326, 2019.

[140] C. Dwork and A. Roth. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9:211–407, 2014.

[141] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography Conference*. Springer, 2006.

[142] Catherine Dwyer. Behavioral Targeting: A Case Study of Consumer Tracking on Levis.com. In *AMCIS*, page 460, 2009.

[143] Sing E. Meta-Model Driven Method for Establishing Business Process Compliance to GDPR. Master's thesis, University of Tartu, 2018.

[144] E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, 2018. URL https://tools.ietf.org/html/rfc8446.

[145] E. Rescorla and N. Modadugu. Datagram Transport Layer Security Version 1.2. RFC 6347, 2012. URL https://tools.ietf.org/html/rfc6347.

[146] Enns, R and Bjorklund, M. and Schoenwaelder, J. and Bierman, A. Network configuration protocol (NETCONF) - Internet Engineering Task Force, RFC 6241. , June 2011.

[147] Sungwook Eom and Jun-Ho Huh. Group signature with restrictive linkability: minimizing privacy exposure in ubiquitous environment. *Journal of Ambient Intelligence and Humanized Computing*, pages 1–11, 2018.

[148] Christian Esposito, Aniello Castiglione, Francesco Palmieri, and Alfredo De Santis. Integrity for an event notification within the industrial internet of things by using group signatures. *IEEE Transactions on Industrial Informatics*, 14(8):3669–3678, 2018.

[149] European Parliament and of the Council. Regulation (EU) 2016/679 of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation). *Official Journal of the European Communities*, OJ L 119/1:1–88, apr 2016. URL http://data.europa.eu/eli/reg/2016/679/oj.

[150] Simone Fischer-Hbner and Stefan Berthold. Privacy-enhancing technologies. In *Computer and Information Security Handbook*, pages 759–778. Elsevier, 2017.

[151] International Organization for Standardization. Iso/iec cd 20009-3 information security — anonymous entity authentication — part 3: Mechanisms based on blind signatures. stage 30.60. Now under development.

[152] International Organization for Standardization. Iso/iec 20008-2: Information technology - security techniques - anonymous digital signatures - part 2: Mechanisms using a group public key. stage 60.60, 2013.

[153] International Organization for Standardization. Iso/iec 20009-2:2013 information technology — security techniques — anonymous entity authentication — part 2: Mechanisms based on signatures using a group public key. stage 90.93, 2013.

[154] Joni Fraga and David Powell. A fault-and intrusion-tolerant file system. In *Proceedings of the 3rd international conference on computer security*, volume 203, pages 2–s2, 1985.

[155] Aurélien Francillon, Daniele Perito, and Claude Castelluccia. Defending embedded systems against control flow attacks. In *Proceedings of the first ACM workshop on Secure execution of untrusted code*, pages 19–26. ACM, 2009.

[156] O. Friel, R. Barnes, M. Pritikin, H. Tschofenig, and M. Baugher. Application-layer tls. https://tools.ietf.org/html/draft-friel-tls-atls-03, 2019. URL https://tools.ietf.org/html/draft-friel-tls-atls-03.

[157] B. C. M. Fung, K. Wang, R. Chen, and P. S. Yu. Privacy-Preserving Data Publishing: A Survey of Recent Developments. *ACM Comput. Surv.*, 42(4):14:1–14:53, 2010.

[158] G. Selander and J. Mattsson and F. Palombini and L. Seitz. Object Security for Constrained RESTful Environments (OSCORE). RFC 8613, 2019. URL https://tools.ietf.org/html/rfc8613.

[159] Vijay K Garg. *Principles of Distributed Systems*. Kluwer Academic Publishers, 1996.

[160] Vijay K Garg. Lattice completion algorithms for distributed computations. In *Int. Conf. On Principles Of Distributed Systems*, pages 166–180, 2012.

[161] C. Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*, pages 169–178, 2009.

[162] E. Godefroy, E. Totel, M. Hurfin, and F. Majorczyk. Automatic generation of correlation rules to detect complex attack scenarios. In *10th International Conference on Information Assurance and Security*, pages 23–28, Nov 2014. doi: 10.1109/ISIAS.2014.7064615.

[163] E. Göktaş, E. Athanasopoulos, M. Polychronakis, H. Bos, and G. Portokalidis. Size does matter: Why using gadget-chain length to prevent code-reuse attacks is hard. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 417–432, 2014.

[164] Ian Goldberg. Privacy-enhancing technologies for the internet iii: ten years later. In *Digital Privacy*, pages 25–40. Auerbach Publications, 2007.

[165] Faming Gong, Ming-Wen Shao, and Guofang Qiu. Concept granular computing systems and their approximation operators. *International Journal of Machine Learning and Cybernetics*, 8(2):627–640, nov 2015. doi: 10.1007/s13042-015-0457-z. URL https://doi.org/10.1007%2Fs13042-015-0457-z.

[166] V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, CCS '06, pages 89–98, 2006.

[167] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM Conference on Computer and Com-*

munications Security, CCS 2006, Alexandria, VA, USA, Ioctober 30 - November 3, 2006, pages 89–98, 2006.

[168] Ben Greenstein, Ramakrishna Gummadi, Jeffrey Pang, Mike Y. Chen, Tadayoshi Kohno, Srinivasan Seshan, and David Wetherall. Can ferris bueller still have his day off? protecting privacy in the wireless era. In *11th USENIX Workshop on Hot Topics in Operating Systems (HOTOS'07)*. USENIX Association, 2007.

[169] Marco Gruteser and Dirk Grunwald. Enhancing location privacy in wireless lan through disposable interface identifiers: a quantitative analysis. *Mobile Networks and Applications*, 10(3):315–325, 2005.

[170] Paolo Guarda, Silvio Ranise, and Hari Siswantoro. Security analysis and legal compliance checking for the design of privacy-friendly information systems. In *SACMAT*, pages 247–254. ACM, 2017.

[171] E. Göktas, E. Athanasopoulos, H. Bos, and G. Portokalidis. Out of control: Overcoming control-flow integrity. In *2014 IEEE Symposium on Security and Privacy*, pages 575–589, May 2014. doi: 10.1109/SP.2014.43.

[172] Nabil Hachem, Hervé Debar, and Joaquin Garcia-Alfaro. HADEGA: A novel MPLS-based mitigation solution to handle network attacks. In *31st IEEE International Performance Computing and Communications Conference, IPCCC 2012, Austin, TX, USA, December 1-3, 2012*, pages 171–180, 2012. doi: 10.1109/PCCC.2012.6407750. URL https://doi.org/10.1109/PCCC.2012.6407750.

[173] Oliver Hahm, Emmanuel Baccelli, Hauke Petersen, and Nicolas Tsiftes. Operating Systems for Low-End Devices in the Internet of Things: A Survey. *IEEE Internet of Things Journal*.

[174] F. Han, J. Qin, and J. Hu. "secure searches in the cloud: A survey". *Future Generation Computer Systems*, 62:66 – 75, 2016.

[175] H. Hellaoui, M. Koudil, and A. Bouabdallah. Energy-efficient mechanisms in security of the internet of things: A survey. *Computer Networks*, 127:173–189, 2017.

[176] Noomene Ben Henda, Karl Norrman, and Katharina Pfeffer. Formal verification of the security for dual connectivity in lte. In *Proceedings of the Third FME Workshop on Formal Methods in Software Engineering*, pages 13–19. IEEE Press, 2015.

[177] Thomas Hérault, Richard Lassaigne, Frédéric Magniette, and Sylvain Peyronnet. Approximate probabilistic model checking. In *International Workshop on Verification, Model Checking, and Abstract Interpretation*, pages 73–84. Springer, 2004.

[178] Johannes Heurix, Peter Zimmermann, Thomas Neubauer, and Stefan Fenz. A taxonomy for privacy enhancing technologies. *Computers & Security*, 53:1–17, 2015.

[179] Jens Hiller, Jan Pennekamp, Markus Dahlmanns, Martin Henze, Andriy Panchenko, and Klaus Wehrle. Tailoring onion routing to the internet of things: Security and privacy in untrusted environments. In *IEEE ICNP*, 2019.

[180] Raymond C. Borges Hink, Justin M. Beaver, Mark A. Buckner, Tommy Morris, Uttam Adhikari, and Shengyi Pan. Machine learning for power system disturbance and cyber-attack discrimination. In *2014 7th International Symposium on Resilient Control Systems (ISRCS)*. IEEE, aug 2014. doi: 10.1109/isrcs.2014.6900095. URL https://doi.org/10.1109%2Fisrcs.2014.6900095.

[181] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Comput.*, 18(7):1527–1554, July 2006. ISSN 0899-7667. doi: 10.1162/neco.2006.18.7.1527. URL http://dx.doi.org/10.1162/neco.2006.18.7.1527.

[182] A. Hoehn and P. Zhang. Detection of covert attacks and zero dynamics attacks in cyber-physical systems. In *2016 American Control Conference (ACC)*, pages 302–307, July 2016. doi: 10.1109/ACC.2016.7524932.

[183] Andreas Hoehn and Ping Zhang. Detection of covert attacks and zero dynamics attacks in cyber-physical systems. In *2016 American Control Conference (ACC)*, pages 302–307, July 2016. doi: 10.1109/ACC.2016.7524932.

[184] K. Hofer-Schmitz and B. Stojanovic. Towards formal methods of iot application layer protocols. In *Proceedings of the 12th CMI conference - Cybersecurity and Privacy (CMI 2019)*, 2019.

[185] K. Hofer-Schmitz and B. Stojanovic. Formal verication for protocols in smart home: A review. *Under review*, 2019.

[186] Hong Hu, Shweta Shinde, Sendroiu Adrian, Zheng Leong Chua, Prateek Saxena, and Zhenkai Liang. Data-Oriented Programming: On the Expressiveness of Non-control Data Attacks. In *oakland*, 2016.

[187] Bing Huang, Yu liang Zhuang, and Hua xiong Li. Information granulation and uncertainty measures in interval-valued intuitionistic fuzzy information systems. *European Journal of Operational Research*,

231(1):162–170, nov 2013. doi: 10.1016/j.ejor.2013.05.006. URL https://doi.org/10.1016%2Fj.ejor.2013.05.006.

[188] IETF. Website of : Ietf suit draft architecture. https://tools.ietf.org/html/draft-ietf-suit-architecture, 2019. [Online; accessed August 7, 2019].

[189] IETF. Website of : Firmware updates for internet of things devices - an information model for manifests draft-ietf-suit-information-model-03. https://tools.ietf.org/html/draft-ietf-suit-information-model-03, 2019. [Online; accessed September 11, 2019].

[190] Intel. The intel iot platform - architecture specification white paper. https://www.intel.it/content/www/it/it/internet-of-things/white-papers/iot-platform-reference-architecture-paper.htm, 2015.

[191] JT IOT. The 5 types of iot platforms. https://blog.jtiotsims.com/the-5-types-of-iot-platforms, 01 2019.

[192] Kyriakos K Ispoglou, Bader AlBassam, Trent Jaeger, and Mathias Payer. Block Oriented Programming: Automating Data-Only Attacks. In *ccs*, 2018.

[193] J. Schaad. CBOR Object Signing and Encryption (COSE). RFC 8152, 2017. URL https://tools.ietf.org/html/rfc8152.

[194] Sushil Jajodia, Anup K Ghosh, Vipin Swarup, Cliff Wang, and X Sean Wang. *Moving target defense: creating asymmetric uncertainty for cyber threats*, volume 54. Springer Science & Business Media, 2011.

[195] Sushil Jajodia, Anup K Ghosh, VS Subrahmanian, Vipin Swarup, Cliff Wang, and X Sean Wang. *Moving Target Defense II: Application of Game Theory and Adversarial Modeling*, volume 100. Springer, 2012.

[196] A. R. Jakhale. Design of anomaly packet detection framework by data mining algorithm for network flow. In *2017 International Conference on Computational Intelligence in Data Science(ICCIDS)*. IEEE, jun 2017. doi: 10.1109/iccids.2017.8272665. URL https://doi.org/10.1109%2Ficcids.2017.8272665.

[197] Yuxuan Jiang, Zhe Huang, and Danny HK Tsang. Challenges and solutions in fog computing orchestration. *IEEE Network*, 32(3):122–129, 2017. URL https://ieeexplore.ieee.org/abstract/document/8121864.

[198] Y. Jin, C. Tian, H. He, and F. Wang. A secure and lightweight data access control scheme for mobile cloud computing. In *2015 IEEE Fifth International Conference on Big Data and Cloud Computing*, pages 172–179, Aug 2015.

[199] Ackbar Joolia, Geoff Coulson, Gordon Blair, Antonio Tadeu Gomes, Kevin Lee, and Jo Ueyama. Flexible programmable networking: A reflective, component-based approach, 2003.

[200] Nerijus Jusas. *Feature model-based development of Internet of Things applications*. PhD Thesis, Kaunas University of Technology, 2017.

[201] Kala K. Refinement of the General Data Protection Regulation (GDPR) Model: Administrative Fines Perspective. Master's thesis, University of Tartu, 2019.

[202] Mark Kattenbelt, Marta Kwiatkowska, Gethin Norman, and David Parker. A game-based abstraction-refinement framework for markov decision processes. *Formal Methods in System Design*, 36(3):246–280, 2010.

[203] K Keerthi, Indrani Roy, Aritra Hazra, and Chester Rebeiro. Formal verification for security in iot devices. In *Security and Fault Tolerance in Internet of Things*, pages 179–200. Springer, 2019.

[204] Ammara Anjum Khan, Mehran Abolhasan, and Wei Ni. 5g next generation VANETs using SDN and fog computing framework. In *2018 15th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, pages 1–6. IEEE, 2018.

[205] Joe Kilian and Erez Petrank. Identity escrow. In *Annual International Cryptology Conference*, pages 169–185. Springer, 1998.

[206] Paul C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *CRYPTO '96*, volume 1109 of *LNCS*, pages 104–113. Springer, 1996.

[207] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *CRYPTO '99*, volume 1666 of *LNCS*, pages 388–397. Springer, 1999.

[208] A. Kolehmainen. Secure firmware updates for iot: A survey. In *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, 2018.

[209] Koen Koning, Xi Chen, Herbert Bos, Cristiano Giuffrida, and Elias Athanasopoulos. No Need to Hide: Protecting Safe Regions on Commodity Hardware. In *eurosys*, 2017.

[210] Israel Koren and C Mani Krishna. *Fault-tolerant systems*. Elsevier, 2010.

[211] A. Korolova. Privacy Violations Using Microtargeted Ads: a Case Study. In *Data Mining Workshops (ICDMW)*, pages 474–482, 2010. doi: 10.1109/ICDMW.2010.137.

[212] Diego Kreutz, Fernando M. V. Ramos, Paulo E. Verissimo, Chistian E. Rothenberg, Siamak Azodolmolky, and Steve Uhlig. Software-Defined Networking: A Comprehensive Survey. *Proceedings of the IEEE*, 103(1):14–76, Jan 2015. ISSN 0018-9219. doi: 10.1109/JPROC.2014.2371999.

[213] David Kuipers and Mark Fabro. Control systems cyber security: Defense in depth strategies. Technical report, Idaho National Laboratory (INL), 2006.

[214] Dilip Kumar, Arthur Beckers, Josep Balasch, Benedikt Gierlichs, and Ingrid Verbauwhede. An in-depth and black-box characterization of the effects of laser pulses on ATmega328P. In *Smart Card Research and Advanced Applications*, 2018.

[215] Trishank Karthik Kuppusamy, Lois Anne DeLong, and Justin Cappos. Uptane: Security and Customizability of Software Updates for Vehicles. *IEEE Vehicular Technology Magazine*, 2018. ISSN 15566072. doi: 10.1109/MVT.2017.2778751.

[216] Marta Kwiatkowska, Gethin Norman, and David Parker. Prism: Probabilistic symbolic model checker. In *International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, pages 200–204. Springer, 2002.

[217] Marta Kwiatkowska, Gethin Norman, and David Parker. Prism 4.0: Verification of probabilistic real-time systems. In *International conference on computer aided verification*, pages 585–591. Springer, 2011.

[218] Jean-François Lalande, Karine Heydemann, and Pascal Berthomé. Software countermeasures for control flow integrity of smart card C codes. In Miroslaw Kutylowski and Jaideep Vaidya, editors, *ESORICS - 19th European Symposium on Research in Computer Security*, volume 8713 of *Lecture Notes in Computer Science*, pages 200–218, Wroclaw, Poland, 2014. Springer International Publishing.

[219] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978.

[220] D. Lanoe, M. Hurfin, and E. Totel. A scalable and efficient correlation engine to detect multi-step attacks in distributed systems. In *IEEE 37th Symposium on Reliable Distributed Systems (SRDS)*, pages 31–40, Oct 2018. doi: 10.1109/SRDS.2018.00014.

[221] David Lanoë, Michel Hurfin, Eric Totel, and Carlos Maziero. An efficient and scalable intrusion detection system on logs of distributed applications. In *ICT Systems Security and Privacy Protection*, pages 49–63, 2019.

[222] Y. Lee and G. Lee. Detecting code reuse attacks with branch prediction. *Computer*, 51(4):40–47, April 2018. doi: 10.1109/MC.2018.2141035.

[223] Y. Lee and G. Lee. Hw-cdi: Hard-wired control data integrity. *IEEE Access*, 7:10811–10822, 2019. doi: 10.1109/ACCESS.2019.2891762.

[224] Yang Lei. Network anomaly traffic detection algorithm based on SVM. In *2017 International Conference on Robots & Intelligent System (ICRIS)*. IEEE, oct 2017. doi: 10.1109/icris.2017.61. URL https://doi.org/10.1109%2Ficris.2017.61.

[225] H. Li and Zhou X. Study on Security Architecture for Internet of Things. In *ICAIC 2011, Part I*, volume CCIS 224, pages 404–411, 2011.

[226] Lan Li. Study on Security Architecture in the Internet of Things. In *Proceedings of 2012 International Conference on Measurement, Information and Control*, volume 1, pages 374–377. IEEE, 2012.

[227] Lichun Li, Rongxing Lu, Kim-Kwang Raymond Choo, Anwitaman Datta, and Jun Shao. Privacy-preserving-outsourced association rule mining on vertically partitioned databases. *IEEE Transactions on Information Forensics and Security*, 11(8):1847–1861, 2016.

[228] N. Li, W. H. Qardaji, and D. Su. On sampling, anonymization, and differential privacy or, *k*-anonymization meets differential privacy. In *7th ACM Symposium on Information, Compuer and Communications Security*, pages 32–33, 2012.

[229] Y. Li, Z. Dai, and J. Li. A control flow integrity checking technique based on hardware support. In *2018 IEEE 3rd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, pages 2617–2621, Oct 2018. doi: 10.1109/IAEAC.2018.8577547.

[230] D. Liberzon and A. S. Morse. Basic problems in stability and design of switched systems. *IEEE Control Systems Magazine*, 19(5):59–70, October 1999. ISSN 1066-033X. doi: 10.1109/37.793443.

[231] Fang Liu, Jin Tong, Jian Mao, Robert Bohn, John Messina, Lee Badger, and Dawn Leaf. NIST cloud computing reference architecture: Recommendations of the National Institute of Standards and Technology (Special Publication 500-292). 2012.

[232] J. Liu, T. Zhao, S. Zhou, Y. Cheng, and Z. Niu. CONCERT: a cloud-based architecture for next-generation cellular systems. *IEEE Wireless Communications*, 21(6):14–22, December 2014. doi: 10.1109/MWC.2014.7000967.

[233] Yao Liu, Peng Ning, and Michael K. Reiter. False Data Injection Attacks Against State Estimation in Electric Power Grids. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, pages 21–32, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-894-0. doi: 10.1145/1653662.1653666.

[234] Yao Liu, Peng Ning, and Michael K Reiter. False data injection attacks against state estimation in electric power grids. *ACM Transactions on Information and System Security (TISSEC)*, 14(1):13, 2011.

[235] Yutao Liu, Tianyu Zhou, Kexin Chen, Haibo Chen, and Yubin Xia. Thwarting Memory Disclosure with Efficient Hypervisor-Enforced Intra-Domain Isolation. In *ccs*, 2015.

[236] David Lo, Leonardo Mariani, and Mauro Pezzè. Automatic steering of behavioral model inference. In *European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 345–354, 2009.

[237] David Lo, Leonardo Mariani, and Mauro Santoro. Learning extended FSA from software: An empirical assessment. *Journal of Systems and Software*, 85(9):2063–2076, 2012.

[238] Hans-Wolfgang Loidl, Kenneth MacKenzie, Steffen Jost, and Lennart Beringer. A proof-carrying-code infrastructure for resources. *2009 Fourth Latin-American Symposium on Dependable Computing*, pages 127–134, 2009.

[239] J. Long, K. Zhang, X. Wang, and H.-N. Dai. Lightweight Distributed Attribute Based Keyword Search System for Internet of Things. In *Security, Privacy, and Anonymity in Computation, Communication, and Storage*, pages 253–264, 2019.

[240] Davide Lorenzoli, Leonardo Mariani, and Mauro Pezzè. Inferring state-based behavior models. In *Int. workshop on Dynamic systems analysis*, pages 25–32, 2006.

[241] Davide Lorenzoli, Leonardo Mariani, and Mauro Pezzè. Automatic generation of software behavioral models. In *Int. Conf. on Software Engineering*, pages 501–510, 2008.

[242] H. Ma, R. Zhang, S. Sun, Z. Song, and G. Tan. Server-aided fine-grained access control mechanism with robust revocation in cloud computing. *IEEE Transactions on Services Computing*, pages 1–1, 2019.

[243] Lukas Malina, Gautam Srivastava, Petr Dzurenda, Jan Hajny, and Sara Ricci. A privacy-enhancing framework for internet of things services. In *International Conference on Network and System Security*, pages 77–97. Springer, 2019.

[244] B. A. Martin, F. Michaud, D. Banks, A. Mosenia, R. Zolfonoon, S. Irwan, S. Schrecker, and J. K. Zao. OpenFog security requirements and approaches. In *2017 IEEE Fog World Congress (FWC)*, pages 1–6, October 2017. doi: 10.1109/FWC.2017.8368537.

[245] R. Matulevičius, J. Tom, K. Kala, and E. Sing. A Method for Managing GDPR Compliance in Business Processes. In *Submitted for review and publication*, 2019.

[246] N. Maunero, P. Prinetto, and G. Roascio. Cfi: Control flow integrity or control flow interruption? In *2019 IEEE East-West Design Test Symposium (EWDTS)*, pages 1–6, Sep. 2019. doi: 10.1109/EWDTS.2019.8884464.

[247] Andreas Maurer. Algorithmic stability and meta-learning. *Journal of Machine Learning Research*, 6(Jun): 967–994, 2005.

[248] Simon Meier, Benedikt Schmidt, Cas Cremers, and David Basin. The tamarin prover for the symbolic analysis of security protocols. In *International Conference on Computer Aided Verification*, pages 696–701. Springer, 2013.

[249] Peter M. Mell and Timothy Grance. SP 800-145. The NIST Definition of Cloud Computing. Technical report, Gaithersburg, MD, United States, 2011. URL https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf.

[250] Wei Meng, Xinyu Xing, Anmol Sheth, Udi Weinsberg, and Wenke Lee. Your Online Interests: Pwned! A Pollution Attack Against Targeted Advertising. In *Computer and Communications Security*, pages 129–140. ACM, 2014. ISBN 978-1-4503-2957-6.

[251] Alexandre Menu, Shivam Bhasin, Jean-Max Dutertre, Jean-Baptiste Rigaud, and Jean-Luc Danger. Precise spatio-temporal electromagnetic fault injections on data transfers. In *2019 Workshop on Fault*

Diagnosis and Tolerance in Cryptography, FDTC 2019, Atlanta, GA, USA, August 24, 2019, pages 1–8, 2019.

[252] Cédric Michel and Ludovic Mé. Adele: an attack description language for knowledge-based intrusion detection. In *Proceedings of the 16th International Conference on Information Security (IFIP/SEC 2001)*, pages 353–365, June 2001.

[253] Jakub Mikians, László Gyarmati, Vijay Erramilli, and Nikolaos Laoutaris. Detecting price and search discrimination on the Internet. In *Hot Topics in Networks*. ACM, 2012.

[254] Daniele Miorandi, Sabrina Sicari, Francesco De Pellegrini, and Imrich Chlamtac. Internet of things: Vision, applications and research challenges. *Ad Hoc Networks*, 10(7):1497 – 1516, 2012. ISSN 1570-8705. doi: https://doi.org/10.1016/j.adhoc.2012.02.016. URL http://www.sciencedirect.com/science/article/pii/S1570870512000674.

[255] Yilin Mo and Bruno Sinopoli. Secure control against replay attacks. In *Communication, Control, and Computing. 47th Annual Allerton Conference on*, pages 911–918. IEEE, Sept 2009. doi: 10.1109/ALLERTON.2009.5394956.

[256] Yilin Mo, Emanuele Garone, Alessandro Casavola, and Bruno Sinopoli. False data injection attacks against state estimation in wireless sensor networks. In *49th IEEE Conference on Decision and Control (CDC)*, pages 5967–5972, Dec 2010. doi: 10.1109/CDC.2010.5718158.

[257] Yilin Mo, Rohan Chabukswar, and Bruno Sinopoli. Detecting integrity attacks on SCADA systems. *IEEE Transactions on Control Systems Technology*, 22(4):1396–1407, July 2014. ISSN 1063-6536. doi: 10.1109/TCST.2013.2280899.

[258] Yilin Mo, Sean Weerakkody, and Bruno Sinopoli. Physical Authentication of Control Systems: Designing Watermarked Control Inputs to Detect Counterfeit Sensor Outputs. *IEEE Control Systems*, 35(1):93–109, February 2015. ISSN 1066-033X. doi: 10.1109/MCS.2014.2364724.

[259] Benjamin Morin, Ludovic Mé, Hervé Debar, and Mireille Duccassé. M4d4: a logical framework to support alert correlation in intrusion detection. *Information Fusion*, 10(4):285–299, 2009.

[260] N. Moro, K. Heydemann, E. Encrenaz, and B. Robisson. Formal verification of a software countermeasure against instruction skip attacks. *Journal of Cryptographic Engineering*, 4(3):145–156, 2014.

[261] Nicolas Moro, Karine Heydemann, Amine Dehbaoui, Bruno Robisson, and Emmanuelle Encrenaz. Experimental evaluation of two software countermeasures against fault attacks. In *Hardware-Oriented Security and Trust*, 2014.

[262] Bruno Mozzaquatro, Carlos Agostinho, Diogo Goncalves, João Martins, and Ricardo Jardim-Goncalves. An ontology-based cybersecurity framework for the internet of things. *Sensors*, 18:3053, 09 2018. doi: 10.3390/s18093053.

[263] Madhavan Mukund, K Narayan Kumar, and Milind Sohoni. Synthesizing distributed finite-state systems from MSCs. In *Int. Conf. on Concurrency Theory*, pages 521–535, 2000.

[264] M. A. Mustafa, S. Cleemput, A. Aly, and A. Abidin. A Secure and Privacy-preserving Protocol for Smart Metering Operational Data Collection. *IEEE Transactions on Smart Grid*, pages 1–1, 2019. doi: 10.1109/TSG.2019.2906016.

[265] Gonzalo Nápoles, Isel Grau, Rafael Falcon, Rafael Bello, and Koen Vanhoof. A granular intrusion detection system using rough cognitive networks. In *Recent Advances in Computational Intelligence in Defense and Security*, pages 169–191. Springer International Publishing, dec 2015. doi: 10.1007/978-3-319-26450-9_7. URL https://doi.org/10.1007%2F978-3-319-26450-9_7.

[266] HG Natke. System identification: Torsten Söderström and Petre Stoica. *Automatica*, 28(5):1069–1071, 1992.

[267] Jiaojiao Niu, Chenchen Huang, Jinhai Li, and Min Fan. Parallel computing techniques for concept-cognitive learning based on granular computing. *International Journal of Machine Learning and Cybernetics*, 9(11):1785–1805, feb 2018. doi: 10.1007/s13042-018-0783-z. URL https://doi.org/10.1007%2Fs13042-018-0783-z.

[268] OASIS. OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA) TC. https://www.oasis-open.org/committees/tosca/, 2019. (Accessed on October 2019).

[269] OASIS XACML Technical Committee. eXtensible Access Control Markup Language (XACML) Version 3.0, 2013.

[270] Internet of Things Questions and Anwsers. What is an iot platform? http://www.iot.qa/2017/11/what-is-iot-platform.htm, 11 2017. URL http://www.iot.qa/2017/11/what-is-iot-platform.html.

[271] R. Oksvort. A Prototype for Learning Privacy-Preserving Data Publishing. Master's thesis, University of Tartu, 2017.

[272] OpenFog Consortium Architecture Working Group. OpenFog reference architecture for fog computing. *OPFRA001*, 20817:162, 2017.

[273] N. Oualha and K. T. Nguyen. Lightweight Attribute-Based Encryption for the Internet of Things. In *2016 25th International Conference on Computer Communication and Networks (ICCCN)*, pages 1–6, Aug 2016.

[274] V. Pappas, M. Polychronakis, and A. D. Keromytis. Transparent rop exploit mitigation using indirect branch tracing. In *Presented as part of the 22nd USENIX Security Symposium (USENIX Security 13)*, pages 447–462, 2013.

[275] Eli Pariser. *The Filter Bubble: What the Internet Is Hiding from You*. The Penguin Group, 2011. ISBN 1594203008, 9781594203008.

[276] Fabio Pasqualetti, Florian Dorfler, and Francesco Bullo. Control-Theoretic Methods for Cyberphysical Security: Geometric Principles for Optimal Cross-Layer Resilient Control Systems. *IEEE Control Systems*, 35(1):110–127, Feb 2015. ISSN 1066-033X. doi: 10.1109/MCS.2014.2364725.

[277] Witold Pedrycz and Wladyslaw Homenda. Building the fundamentals of granular computing: A principle of justifiable granularity. *Applied Soft Computing*, 13(10):4209–4218, oct 2013. doi: 10.1016/j.asoc.2013.06.017. URL https://doi.org/10.1016%2Fj.asoc.2013.06.017.

[278] Anastasia Pentina and Christoph Lampert. A pac-bayesian bound for lifelong learning. In *International Conference on Machine Learning*, pages 991–999, 2014.

[279] Anastasia Pentina and Christoph H Lampert. Lifelong learning with non-iid tasks. In *Advances in Neural Information Processing Systems*, pages 1540–1548, 2015.

[280] Asier Perallos, Unai Hernandez-Jayo, Ignacio Julio García Zuazola, and Enrique Onieva. *Intelligent Transport Systems: Technologies and Applications*. John Wiley & Sons, 2015.

[281] Katharina Pfeffer. Formal verification of a lte security protocol for dual-connectivity: An evaluation of automatic model checking tools, 2014.

[282] Andrés F Murillo Piedrahita, Vikram Gaur, Jairo Giraldo, Alvaro A Cardenas, and Sandra Julieta Rueda. Virtual incident response functions in control systems. *Computer Networks*, 135:147–159, 2018.

[283] Evaggelia Pitoura, Panayiotis Tsaparas, Giorgos Flouris, Irini Fundulaki, Panagiotis Papadakos, Serge Abiteboul, and Gerhard Weikum. On measuring bias in online information. *SIGMOD Rec.*, 46(4):16–21, February 2018. ISSN 0163-5808. doi: 10.1145/3186549.3186553. URL http://doi.acm.org/10.1145/3186549.3186553.

[284] Mohsen Pourpouneh and Rasoul Ramezanian. A short introduction to two approaches in formal verification of security protocols: model checking and theorem proving. *The ISC International Journal of Information Security*, 8(1):3–24, 2016.

[285] David Powell. *Delta-4: a generic architecture for dependable distributed computing*, volume 1. Springer Science & Business Media, 2012.

[286] Ramjee Prasad and Marina Ruggieri. Special issue on "intelligent infrastructure". *Wireless Personal Communications*, 76(2):121–124, 05 2014. ISSN 1572-834X. doi: 10.1007/s11277-014-1681-7. URL https://doi.org/10.1007/s11277-014-1681-7.

[287] M. Pritikin, M. Richardson, T. Eckert, M. Behringer, and K. Watsen. Bootstrapping remote secure key infrastructure (brski). https://tools.ietf.org/html/draft-ietf-anima-bootstrapping-keyinfra-30, 2019. URL https://tools.ietf.org/html/draft-ietf-anima-bootstrapping-keyinfra-30.

[288] Sergej Proskurin, Marius Momeu, Seyedhamed Ghavamnia, Vasileios P. Kemerlis, and Michalis Polychronakis. xMP: Selective Memory Protection for Kernel and User Space. In *oakland*, 2020.

[289] Julien Proy, Karine Heydemann, Alexandre Berzati, and Albert Cohen. Compiler-assisted loop hardening against fault attacks. *ACM Transactions on Architecture and Code Optimization*, 14:1–25, 12 2017. doi: 10.1145/3141234.

[290] D. Puthal, S. P. Mohanty, S. A. Bhavake, G. Morgan, and R. Ranjan. Fog Computing Security Challenges and Future Directions [Energy and Security]. *IEEE Consumer Electronics Magazine*, 8(3):92–96, May 2019. doi: 10.1109/MCE.2019.2893674.

[291] Yuhua Qian, Jiye Liang, and Chuangyin Dang. Incomplete multigranulation rough set. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 40(2):420–431, mar 2010. doi: 10.1109/tsmca.2009.2035436. URL https://doi.org/10.1109%2Ftsmca.2009.2035436.

[292] P. Qiu, Y. Lyu, J. Zhang, D. Wang, and G. Qu. Control flow integrity based on lightweight encryption architecture. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(7): 1358–1369, July 2018. doi: 10.1109/TCAD.2017.2748000.

[293] Qiang Qiu and Guillermo Sapiro. Learning transformations for clustering and classification. *The Journal of Machine Learning Research*, 16(1):187–225, 2015.

[294] Jean-Pierre Quemard, Jan Schallabok, Irene Kamara, and Matthias Pocs. *Guidance and gap analysis for European standardisation: Privacy standards in the information security context*. ENISA, 1 edition, 3 2019.

[295] Panagiotis I Radoglou-Grammatikis and Panagiotis G Sarigiannidis. Securing the smart grid: A comprehensive compilation of intrusion detection and prevention systems. *IEEE Access*, 7:46595–46620, 2019.

[296] Inez Raguenet and Carlos Maziero. A fuzzy model for the composition of intrusion detectors. In *Proceedings of The Ifip Tc 11 23rd International Information Security Conference*, pages 237–251, 2008.

[297] Silvio Ranise and Hari Siswantoro. Automated legal compliance checking by security policy analysis. In *SAFECOMP Workshops*, volume 10489 of *Lecture Notes in Computer Science*, pages 361–372. Springer, 2017.

[298] Joel Reardon and Ian Goldberg. Improving tor using a tcp-over-dtls tunnel. In *Proceedings of the 18th conference on USENIX security symposium*, pages 119–134. USENIX Association, 2009.

[299] E. Rescorla, R. Barnes, and H. Tschofenig. Compact tls 1.3. https://tools.ietf.org/html/draft-rescorla-tls-ctls-03, 2019. URL https://tools.ietf.org/html/draft-rescorla-tls-ctls-03.

[300] Matthieu Rivain, Emmanuel Prouff, and Julien Doget. Higher-order masking and shuffling for software implementations of block ciphers. In *CHES 2009*, volume 5747 of *LNCS*, pages 171–188. Springer, 2009.

[301] Lionel Rivière, Zakaria Najm, Pablo Rauzy, Jean-Luc Danger, and Julien Bringer. High precision fault attacks on the instruction cache of ARMv7-M architectures. In *2015 IEEE International Symposium on Hardware Oriented Security and Trust*, 2015.

[302] N. Roessler and A. DeHon. Protecting the stack with metadata policies and tagged hardware. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 478–495, May 2018. doi: 10.1109/SP.2018.00066.

[303] Rodrigo Roman, Jianying Zhou, and Javier Lopez. On the features and challenges of security and privacy in distributed internet of things. *Computer Networks*, 57(10):2266–2279, 2013.

[304] Rodrigo Roman, Javier Lopez, and Masahiro Mambo. Mobile edge computing, fog et al.: A survey and analysis of security threats and challenges. *Future Generation Computer Systems*, 78:680–698, 2018.

[305] J.M. Rubio-Hernan. *Detection of Attacks against Cyber-Physical Industrial Systems*. PhD thesis, Télécom SudParis, 2017.

[306] Jose Rubio-Hernan, Luca De Cicco, and Joaquin Garcia-Alfaro. Revisiting a watermark-based detection scheme to handle cyber-physical attacks. In *Availability, Reliability and Security (ARES), 2016 11th International Conference on*, pages 21–28. IEEE, August 2016. doi: 10.1109/ARES.2016.2. URL http://dx.doi.org/10.1109/ARES.2016.2.

[307] Jose Rubio-Hernan, Luca De Cicco, and Joaquin Garcia-Alfaro. Event-triggered watermarking control to handle cyber-physical integrity attacks. In *21st Nordic Conference on Secure IT Systems (NordSec 2016)*, pages 3–19. Springer, November 2016. doi: 10.1007/978-3-319-47560-8_1. URL http://dx.doi.org/10.1007/978-3-319-47560-8_1.

[308] Jose Rubio-Hernan, Juan Rodolfo-Mejias, and Joaquin Garcia-Alfaro. Security of cyber-physical systems. In *Conference on Security of Industrial-Control-and Cyber-Physical Systems*, pages 3–18. Springer, 2016.

[309] Jose Rubio-Hernan, Luca De Cicco, and Joaquin Garcia-Alfaro. On the use of watermark-based schemes to detect cyber-physical attacks. *EURASIP Journal on Information Security*, 2017(1):8, 2017. doi: 10.1186/s13635-017-0060-9. URL http://dx.doi.org/10.1186/s13635-017-0060-9.

[310] Jose Rubio-Hernan, Luca De Cicco, and Joaquin Garcia-Alfaro. Adaptive control-theoretic detection of integrity attacks against cyber-physical industrial systems. *Transactions on Emerging Telecommunications Technologies*, 29(7):3209–3225, 2018. doi: 10.1002/ett.3209. URL http://dx.doi.org/10.1002/ett.3209.

[311] Jose Rubio-Hernan, Rishikesh Sahay, Luca De Cicco, and Joaquin Garcia-Alfaro. Cyber-physical architecture assisted by programmable networking. *Internet Technology Letters*, page e44, 2018.

[312] Jose Fran. Ruiz, Marinella Petrocchi, Ilaria Matteucci, Gianpiero Costantino, Carmela Gambardella, Mirko Manea, and Anil Ozdeniz. A lifecycle for data sharing agreements: How it works out. In *Privacy Technologies and Policy - 4th Annual Privacy Forum, APF 2016, Frankfurt/Main, Germany, September 7-8, 2016, Proceedings*, pages 3–20, 2016. doi: 10.1007/978-3-319-44760-5_1.

[313] Matt Rutkowski, Luc Boutier, and Chris Lauwers. TOSCA Simple Profile in YAML Version 1.2. Technical report, OASIS, January 2019. URL https://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.2/os/TOSCA-Simple-Profile-YAML-v1.2-os.pdf.

[314] AliAkbar Sadeghi, Salman Niksefat, and Maryam Rostamipour. Pure-call oriented programming (pcop): chaining the gadgets using call instructions. *Journal of Computer Virology and Hacking Techniques*, 14 (2):139–156, May 2018. ISSN 2263-8733. doi: 10.1007/s11416-017-0299-1. URL https://doi.org/10.1007/s11416-017-0299-1.

[315] Łukasz Saganowski, Marcin Goncerzewicz, and Tomasz Andrysiak. Anomaly detection preprocessor for SNORT IDS system. In *Advances in Intelligent Systems and Computing*, pages 225–232. Springer Berlin Heidelberg, 2013. doi: 10.1007/978-3-642-32384-3_28. URL https://doi.org/10.1007%2F978-3-642-32384-3_28.

[316] Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In *Proceedings of the 24th Annual International Conference on Theory and Applications of Cryptographic Techniques*, pages 457–473, 2005.

[317] Rishikesh Sahay, Gregory Blanc, Zonghua Zhang, and Hervé Debar. Towards autonomic DDoS mitigation using Software Defined Networking. In *SENT 2015 : NDSS Workshop on Security of Emerging Networking Technologies*, page ., San Diego, Ca, United States, February 2015. Internet society. doi: 10.14722/sent.2015.23004. URL https://hal.archives-ouvertes.fr/hal-01257899.

[318] Umair Sarfraz, Masoom Alam, Sherali Zeadally, and Abid Khan. Privacy aware iota ledger: Decentralized mixing and unlinkable iota transactions. *Computer Networks*, 148:361–372, 2019.

[319] Christian Schellenberger and Ping Zhang. Detection of covert attacks on cyber-physical systems by extending the system dynamics with an auxiliary system. In *56th IEEE Annual Conference on Decision and Control (CDC)*, pages 1374–1379, December 2017. doi: 10.1109/CDC.2017.8263846.

[320] Padraig Scully. Understanding iot security – part 1 of 3: Iot security architecture on the device and communication layers. https://iot-analytics.com/understanding-iot-security-part-1-iot-security-architecture/, 2016.

[321] Padraig Scully. Iiot platforms for manufacturing 2019-2024. http://iot-analytics.com/wp/wp-content/uploads/2019/04/IIoT-Platforms-for-Manufacturing-2019-2024-vSample.pdf, 04 2019.

[322] Padraig Scully and Knud Lasse Lueth. Guite to iot solution development. https://iot-analytics.com/wp/wp-content/uploads/2016/09/White-paper-Guide-to-IoT-Solution-Development-September-2016-vf.pdf, 09 2016.

[323] Noam Segev, Maayan Harel, Shie Mannor, Koby Crammer, and Ran El-Yaniv. Learn on source, refine on target: A model transfer learning framework with random forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(9):1811–1824, sep 2017. doi: 10.1109/tpami.2016.2618118. URL https://doi.org/10.1109%2Ftpami.2016.2618118.

[324] Mariana Segovia, Ana Cavalli, Nora Cuppens, and Joaquin Garcia-Alfaro. A Study on Mitigation Techniques for SCADA-driven Cyber-Physical Systems. In *Foundations and Practice of Security. FPS 2018. Lecture Notes in Computer Science, vol 11358*. Springer, November 2018.

[325] Mariana Segovia, Ana Cavalli, Nora Cuppens, Jose Rubio-Hernan, and Joaquin Garcia-Alfaro. Reflective Attenuation of Cyber-Physical Attacks. In *5th CyberICPS Workshop, 24th European Symposium on Research in Computer Security (ESORICS 2019). Lecture Notes in Computer Science, volume 11980*. Springer, September 2019.

[326] L. Seitz, G. Selander, E. Wahlstroem, S. Erdtman, and H. Tschofenig. Authentication and authorization for constrained environments (ace) using the oauth 2.0 framework (ace-oauth). https://tools.ietf.org/html/draft-ietf-ace-oauth-authz-27, 2019. URL https://tools.ietf.org/html/draft-ietf-ace-oauth-authz-27.

[327] G. Selander, J. Mattsson, and F. Palombini. Ephemeral diffie-hellman over cose (edhoc). https://tools.ietf.org/html/draft-selander-lake-edhoc-00, 2019. URL https://tools.ietf.org/html/draft-selander-lake-edhoc-00.

[328] H. Shacham et al. The geometry of innocent flesh on the bone: return-into-libc without function calls (on the x86). In *ACM conference on Computer and communications security*, pages 552–561. New York,, 2007.

[329] Wenli Shang, Junrong Cui, Chunhe Song, Jianming Zhao, and Peng Zeng. Research on industrial control anomaly detection based on FCM and SVM. In *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. IEEE, aug 2018. doi: 10.1109/trustcom/bigdatase.2018.00042. URL https://doi.org/10.1109%2Ftrustcom%2Fbigdatase.2018.00042.

[330] Yun Shen and Siani Pearson. Privacy enhancing technologies: A review. *HP Laboratories*, 2739:1–30, 2011.

[331] Y. Shi and G. Lee. Augmenting branch predictor to secure program execution. In *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*, pages 10–19, June 2007. doi: 10.1109/DSN.2007.19.

[332] Kevin Sim, Emma Hart, and Ben Paechter. A lifelong learning hyper-heuristic method for bin packing. *Evolutionary Computation*, 23(1):37–67, mar 2015. doi: 10.1162/evco_a_00121. URL https://doi.org/10.1162%2Fevco_a_00121.

[333] Mauw Sjouke and Oostdijk Martijn. Foundations of attack trees. In *Information Security and Cryptology - ICISC 2005*, pages 186–198, 2006. doi: 10.1007/11734727_17.

[334] Roy Smith. A decoupled feedback structure for covertly appropriating networked control systems. *IFAC Proceedings Volumes*, 44(1):90 – 95, 2011. ISSN 1474-6670. 18th IFAC World Congress.

[335] Roy Smith. Covert Misappropriation of Networked Control Systems: Presenting a Feedback Structure. *IEEE Control Systems*, 35(1):82–92, Feb 2015. ISSN 1066-033X. doi: 10.1109/MCS.2014.2364723.

[336] Mingli Song, Wenqian Shang, Lidong Wang, and Witold Pedrycz. Analysis of spatiotemporal data relationship using information granules. *International Journal of Machine Learning and Cybernetics*, 8(5):1439–1446, jun 2015. doi: 10.1007/s13042-015-0386-x. URL https://doi.org/10.1007%2Fs13042-015-0386-x.

[337] Markus Stadler, Jean-Marc Piveteau, and Jan Camenisch. Fair blind signatures. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 209–219. Springer, 1995.

[338] Ivan Stojmenovic. Fog computing: A cloud to the ground support for smart things and machine-to-machine networks. In *2014 Australasian Telecommunication Networks and Applications Conference (ATNAC)*, pages 117–122. IEEE, 2014.

[339] Gianluca Stringhini, Christopher Kruegel, and Giovanni Vigna. Detecting spammers on social networks. In *Proceedings of the 26th Annual Computer Security Applications Conference (ACSAC'10)*, pages 1–9. ACM, 2010.

[340] Gianluca Stringhini, Gang Wang, Manuel Egele, Christopher Kruegel, Giovanni Vigna, Haitao Zheng, and Ben Y Zhao. Follow the green: growth and dynamics in Twitter follower markets. In *Proceedings of the 13th Internet Measurement Conference (IMC'13)*, pages 163–176. ACM, 2013.

[341] D. Sullivan, O. Arias, L. Davi, P. Larsen, A. Sadeghi, and Y. Jin. Strategy without tactics: Policy-agnostic hardware-enhanced control-flow integrity. In *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, June 2016. doi: 10.1145/2897937.2898098.

[342] Bing Sun, Chong Xu, and Stanley Chong. The Power of Data-Oriented Attacks. *Black Hat, Asia*, 2017.

[343] Microsoft Support. A detailed description of the Data Execution Prevention (DEP). https://support.microsoft.com/en-us/help/875352/a-detailed-description-of-the-data-execution-prevention-dep-feature-in. [Online; accessed 18-June-2019].

[344] Zareen Syed, Ankur Padia, Timothy W. Finin, M. Lisa Mathews, and Anupam Joshi. Uco: A unified cybersecurity ontology. In *AAAI Workshop: Artificial Intelligence for Cyber Security*, 2016.

[345] L. Szekeres, M. Payer, T. Wei, and D. Song. Sok: Eternal war in memory. In *2013 IEEE Symposium on Security and Privacy*, pages 48–62, May 2013. doi: 10.1109/SP.2013.13.

[346] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, 2008. URL https://tools.ietf.org/html/rfc5246.

[347] Irfan Khan Tanoli, Marinella Petrocchi, and Rocco De Nicola. Towards automatic translation of social network policies into controlled natural language. In *12th International Conference on Research Challenges in Information Science, RCIS 2018, Nantes, France, May 29-31, 2018*, pages 1–12, 2018. doi: 10.1109/RCIS.2018.8406683.

[348] Ming Tao, Jinglong Zuo, Zhusong Liu, Aniello Castiglione, and Francesco Palmieri. Multi-layer cloud architectural model and ontology-based security service framework for iot-based smart homes. *Future Generation Comp. Syst.*, 78:1040–1051, 2018.

[349] PaX Team. PaX Non-Executable Pages Design and Implementation. https://pax.grsecurity.net/docs/noexec.txt, 2003. [Online; accessed 17-June-2019].

[350] TA Team et al. Avispa v1. 1 user manual. *Information Society Technologies Programme (June 2006), http://avispa-project. org*, 2006.

[351] KaaIoT Technologies. Iot 101: What is an iot platform. https://www.kaaproject.org/what-is-iot-platform/, 2019.

[352] André Teixeira, Daniel Pérez, Henrik Sandberg, and Karl Henrik Johansson. Attack Models and Scenarios for Networked Control Systems. In *Proceedings of the 1st International Conference on High Confidence Networked Systems*, HiCoNS '12, pages 55–64, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1263-9. doi: 10.1145/2185505.2185515.

[353] André Teixeira, Iman Shames, Henrik Sandberg, and Karl Henrik Johansson. A secure control framework for resource-limited adversaries. *Automatica*, 51:135–148, 2015. ISSN 0005-1098. doi: http://dx.doi.org/10.1016/j.automatica.2014.10.067.

[354] D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall, and G. J. Minden. A survey of active network research. *Comm. Mag.*, 35(1):80–86, January 1997. ISSN 0163-6804. doi: 10.1109/35.568214. URL https://doi.org/10.1109/35.568214.

[355] Aakanksha Tewari and BB Gupta. Security, privacy and trust of different layers in Internet-of-Things (IoTs) framework. *Future Generation Computer Systems*, 2018.

[356] J. Tom, E. Sing, and R. Matuleviŋius. Conceptual Representation of the GDPR: Model and Application Directions. In J. Zdravkovic, J. Grabis, S. Nurcan, and J. Stirna, editors, *Perspectives in Business Informatics Research. BIR 2018*, volume 330. Springer, 2018.

[357] Eric Totel, Bernard Vivinis, and Ludovic Mé. A Language Driven Intrusion Detection System for Event and Alert Correlation. In *Proc. ot the 19th IFIP Int. Information Security Conf.*, pages 209–224. Kluwer Academic, 2004.

[358] Eric Totel, Mouna Hkimi, Michel Hurfin, Mourad Leslous, and Yvan Labiche. Inferring a distributed application behavior model for anomaly based intrusion detection. In *European Dependable Computing Conference (EDCC)*, pages 53–64, 2016.

[359] E. Trichina and R. Korkikyan. Multi fault laser attacks on protected CRT-RSA. In *Fault Diagnosis and Tolerance in Cryptography*, 2010.

[360] Tzi-Cker Chiueh and Fu-Hau Hsu. Rad: a compile-time solution to buffer overflow attacks. In *Proceedings 21st International Conference on Distributed Computing Systems*, pages 409–417, April 2001. doi: 10.1109/ICDSC.2001.918971.

[361] Fredrik Valeur. *Real-Time Intrusion Detection Alert Correlation*. PhD thesis, UNIVERSITY OF CALIFORNIA, 2006.

[362] Nathan Vance, Daniel Yue Zhang, Yang Zhang, and Dong Wang. Privacy-aware edge computing in social sensing applications using ring signatures. In *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*, pages 755–762. IEEE, 2018.

[363] A. Vasselle, H. Thiebeauld, Q. Maouhoub, A. Morisset, and S. Ermeneux. Laser-induced fault injection on smartphone bypassing the secure boot. In *2017 Workshop on Fault Diagnosis and Tolerance in Cryptography*, 2017.

[364] K. Velasquez, D. P. Abreu, D. Gonçalves, L. Bittencourt, M. Curado, E. Monteiro, and E. Madeira. Service Orchestration in Fog Environments. In *2017 IEEE 5th International Conference on Future Internet of Things and Cloud (FiCloud)*, pages 329–336, August 2017. doi: 10.1109/FiCloud.2017.49.

[365] Karima Velasquez, David Perez Abreu, Marcio R. M. Assis, Carlos Senna, Diego F. Aranha, Luiz F. Bittencourt, Nuno Laranjeiro, Marilia Curado, Marco Vieira, Edmundo Monteiro, and Edmundo Madeira. Fog orchestration for the Internet of Everything: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, 9(1):14, July 2018. ISSN 1869-0238. doi: 10.1186/s13174-018-0086-3. URL https://doi.org/10.1186/s13174-018-0086-3.

[366] Algimantas Venckauskas, Vytautas Stuikys, Jevgenijus Toldinas, and Nerijus Jusas. A Model-Driven Framework to Develop Personalized Health Monitoring. *Symmetry-Basel*, 8(7):65, 2016. ISSN 2073-8994. doi: 10.3390/sym8070065.

[367] Paulo Verissimo and Luis Rodrigues. *Distributed systems for system architects*, volume 1. Springer Science & Business Media, 2012.

[368] Paulo Verissimo, Antonio Casimiro, and Christof Fetzer. The timely computing base: Timely actions in the presence of uncertain timeliness. In *Proceeding International Conference on Dependable Systems and Networks. DSN 2000*, pages 533–542. IEEE, 2000.

[369] Luca Viganò. Automated security protocol analysis with the avispa tool. *Electronic Notes in Theoretical Computer Science*, 155:61–86, 2006.

[370] Jouni Viinikka, Hervé Debar, Ludovic Mé, Anssi Lehikoinen, and Mika Tarvainen. Processing intrusion detection alert aggregates with time series modeling. *Information Fusion*, 10(4):312 – 324, 2009. doi: https://doi.org/10.1016/j.inffus.2009.01.003. Special Issue on Information Fusion in Computer Security.

[371] Sabrina De Capitani Di Vimercati, Sara Foresti, Pierangela Samarati, and Sushil Jajodia. Access control policies and languages. *Int. J. Comput. Sci. Eng.*, 3(2):94–102, November 2007. ISSN 1742-7185. doi: 10.1504/IJCSE.2007.015739. URL http://dx.doi.org/10.1504/IJCSE.2007.015739.

[372] M. Vucinic, J. Simon, K. Pister, and M. Richardson. Constrained join protocol (cojp) for 6tisch. https://tools.ietf.org/html/draft-ietf-6tisch-minimal-security-15, 2019. URL https://tools.ietf.org/html/draft-ietf-6tisch-minimal-security-15.

[373] Christian Wagner, Simon Miller, Jonathan M. Garibaldi, Derek T. Anderson, and Timothy C. Havens. From interval-valued data to general type-2 fuzzy sets. *IEEE Transactions on Fuzzy Systems*, 23(2): 248–269, apr 2015. doi: 10.1109/tfuzz.2014.2310734. URL https://doi.org/10.1109%2Ftfuzz.2014.2310734.

[374] Isabel Wagner and David Eckhoff. Technical privacy metrics: a systematic survey. *ACM Computing Surveys (CSUR)*, 51(3):57, 2018.

[375] Eric Ke Wang, Yunming Ye, Xiaofei Xu, S. M. Yiu, L. C. K. Hui, and K. P. Chow. Security issues and challenges for cyber physical system. In *2010 IEEE/ACM Int Conference on Green Computing and Communications & Int Conference on Cyber, Physical and Social Computing*. IEEE, dec 2010. doi: 10.1109/greencom-cpscom.2010.36. URL https://doi.org/10.1109%2Fgreencom-cpscom.2010.36.

[376] Pin-Han Wang, I-En Liao, Kuo-Fong Kao, and Jyun-Yao Huang. An intrusion detection method based on log sequence clustering of honeypot for modbus tcp protocol. In *2018 IEEE International Conference on Applied System Invention (ICASI)*, pages 255–258. IEEE, 2018.

[377] W. Wang, M. Liu, P. Du, Z. Zhao, Y. Tian, Q. Hao, and X. Wang. An architectural-enhanced secure embedded system with a novel hybrid search scheme. In *2017 International Conference on Software Security and Assurance (ICSSA)*, pages 116–120, July 2017. doi: 10.1109/ICSSA.2017.14.

[378] Yang Wang. Privacy-enhancing technologies. In *Handbook of research on social and organizational liabilities in information security*, pages 203–227. IGI Global, 2009.

[379] Z. Wen, R. Yang, P. Garraghan, T. Lin, J. Xu, and M. Rovatsos. Fog Orchestration for Internet of Things Services. *IEEE Internet Computing*, 21(2):16–24, March 2017. doi: 10.1109/MIC.2017.36.

[380] Wenjian He, S. Das, W. Zhang, and Y. Liu. No-jump-into-basic-block: Enforce basic block cfi on the fly for real-world binaries. In *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, June 2017. doi: 10.1145/3061639.3062291.

[381] Gerhard Widmer and Miroslav Kubat. Learning in the presence of concept drift and hidden contexts. *Machine Learning*, 23(1):69–101, apr 1996. doi: 10.1007/bf00116900. URL https://doi.org/10.1007%2Fbf00116900.

[382] WolfSSL. WolfSSL - Embedded TLS Library. https://www.wolfssl.com/.

[383] Run Xie, Chanlian He, Chunxiang Xu, and Chongzhi Gao. Lattice-based dynamic group signature for anonymous authentication in iot. *Annals of Telecommunications*, pages 1–12, 2019.

[384] Maysam Yabandeh, Abhishek Anand, Marco Canini, and Dejan Kostic. Finding almost-invariants in distributed systems. In *Symp. on Reliable Distributed Systems (SRDS)*, 2011.

[385] Chao Yang, Robert Harkreader, and Guofei Gu. Empirical evaluation and new design for fighting evolving Twitter spammers. *IEEE Transactions on Information Forensics and Security*, 8(8):1280–1293, 2013.

[386] Hsiuhan Lexie Yang and Melba M. Crawford. Domain adaptation with preservation of manifold geometry for hyperspectral image classification. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 9(2):543–555, feb 2016. doi: 10.1109/jstars.2015.2449738. URL https://doi.org/10.1109%2Fjstars.2015.2449738.

[387] Hsiuhan Lexie Yang and Melba M. Crawford. Spectral and spatial proximity-based manifold alignment for multitemporal hyperspectral image classification. *IEEE Transactions on Geoscience and Remote Sensing*, 54(1):51–64, jan 2016. doi: 10.1109/tgrs.2015.2449736. URL https://doi.org/10.1109%2Ftgrs.2015.2449736.

[388] Ji-Jiang Yang, Jian-Qiang Li, and Yu Niu. A hybrid solution for privacy preserving medical data sharing in the cloud environment. *Future Generation Computer Systems*, 43:74–86, 2015.

[389] K. Yang and X. Jia. Expressive, efficient, and revocable data access control for multi-authority cloud storage. *IEEE Transactions on Parallel and Distributed Systems*, 25(7):1735–1744, July 2014.

[390] X. Yang, Z. Li, Z. Geng, and H. Zhang. A Multi-layer Security Model for Internet of Things. In *IOT Workshop 2012*, volume CCIS 312, pages 388–393, 2012.

[391] Xue Yang, Zhihua Li, Zhenmin Geng, and Haitao Zhang. A Multi-layer Security Model for Internet of Things. In Yongheng Wang and Xiaoming Zhang, editors, *Internet of Things*, pages 388–393, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[392] Zhi Yang, Christo Wilson, Xiao Wang, Tingting Gao, Ben Y Zhao, and Yafei Dai. Uncovering social network sybils in the wild. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 8(1):2:1–2:29, 2014.

[393] Zhihong Yang, Yingzhao Yue, Yu Yang, Yufeng Peng, Xiaobo Wang, and Wenji Liu. Study and Application on the Architecture and Key Technologies for IoT. In *2011 International Conference on Multimedia Technology*, pages 747–751. IEEE, 2011.

[394] X. Yao, Z. Chen, and Y. Tian. A lightweight attribute-based encryption scheme for the Internet of Things. *Future Generation Computer Systems*, 49:104 – 112, 2015.

[395] Yiyu Yao. A partition model of granular computing. In *Transactions on Rough Sets I*, pages 232–253. Springer Berlin Heidelberg, 2004. doi: 10.1007/978-3-540-27794-1_11. URL https://doi.org/10.1007%2F978-3-540-27794-1_11.

[396] Yiyu Yao and Ning Zhong. Granular computing. *Wiley Encyclopedia of Computer Science and Engineering*, 2007.

[397] Shanhe Yi, Zijiang Hao, Zhengrui Qin, and Qun Li. Fog computing: Platform and applications. In *2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb)*, pages 73–78. IEEE, 2015.

[398] Shanhe Yi, Cheng Li, and Qun Li. A survey of fog computing: concepts, applications and issues. In *Proceedings of the 2015 workshop on mobile big data*, pages 37–42. ACM, 2015.

[399] Hyunguk Yoo and Taeshik Shon. Novel approach for detecting network anomalies for substation automation based on iec 61850. *Multimedia Tools and Applications*, 74(1):303–318, 2015.

[400] Håkan LS Younes and Reid G Simmons. Probabilistic verification of discrete event systems using acceptance sampling. In *International Conference on Computer Aided Verification*, pages 223–235. Springer, 2002.

[401] Ashkan Yousefpour, Caleb Fung, Tam Nguyen, Krishna Kadiyala, Fatemeh Jalali, Amirreza Niakanlahiji, Jian Kong, and Jason P. Jue. All one needs to know about fog computing and related edge computing paradigms: A complete survey. *Journal of Systems Architecture*, 98:289 – 330, 2019. ISSN 1383-7621. doi: https://doi.org/10.1016/j.sysarc.2019.02.009. URL http://www.sciencedirect.com/science/article/pii/S1383762118306349.

[402] Omerah Yousuf and Roohie Naaz Mir. A Survey on the Internet of Things Security: State-of-Art, Architecture, Issues and Countermeasures. *Information & Computer Security*, 27(2):292–323, 2019.

[403] B. Yuce, N. F. Ghalaty, H. Santapuri, C. Deshpande, C. Patrick, and P. Schaumont. Software fault resistance is futile: Effective single-glitch attacks. In *2016 Workshop on Fault Diagnosis and Tolerance in Cryptography*, 2016.

[404] B. Yuce, C. Deshpande, M. Ghodrati, A. Bendre, L. Nazhandali, and P. Schaumont. A secure exception mode for fault-attack-resistant processing. *IEEE Transactions on Dependable and Secure Computing*, 16(3):388–401, May 2019. ISSN 2160-9209. doi: 10.1109/TDSC.2018.2823767.

[405] Z. Shelby and K. Hartke and C. Bormann. The Constrained Application Protocol (CoAP). RFC 7252, 2014. URL https://tools.ietf.org/html/rfc7252.

[406] Lotfi A. Zadeh. Toward a theory of fuzzy information granulation and its centrality in human reasoning and fuzzy logic. *Fuzzy Sets and Systems*, 90(2):111–127, sep 1997. doi: 10.1016/s0165-0114(97)00077-8. URL https://doi.org/10.1016%2Fs0165-0114%2897%2900077-8.

[407] Koen Zandberg, Kaspar Schleiser, Francisco Acosta, Hannes Tschofenig, and Emmanuel Baccelli. Secure Firmware Updates for Constrained IoT Devices Using Open Standards: A Reality Check. *IEEE Access*, 2019.

[408] Eva Zangerle and Günther Specht. Sorry, I was hacked: a classification of compromised Twitter accounts. In *Proceedings of the 29th Symposium on Applied Computing (SAC'14)*, pages 587–593. ACM, 2014.

[409] Z. Zhang, M. C. Y. Cho, C. Wang, C. Hsu, C. Chen, and S. Shieh. IoT Security: Ongoing Challenges and Research Opportunities. In *2014 IEEE 7th International Conference on Service-Oriented Computing and Applications*, pages 230–234, Nov 2014.

[410] Kai Zhao and Lina Ge. A Survey on the Internet of Things Security. In *2013 Ninth international conference on computational intelligence and security*, pages 663–667. IEEE, 2013.

[411] Hongfei Zhu, Yu-an Tan, Xiaosong Zhang, Liehuang Zhu, Changyou Zhang, and Jun Zheng. A round-optimal lattice-based blind signature scheme for cloud services. *Future Generation Computer Systems*, 73:106–114, 2017.