A CYBERATTACK IMPACT ANALYSIS APPROACH FOR INDUSTRIAL CONTROL SYSTEMS

BY

ALVI JAWAD

A thesis submitted to the Faculty of Graduate and Postdoctoral Affairs
in partial fulfillment of the requirements for the degree of

Master of Applied Science
in
Electrical and Computer Engineering

Carleton University
Ottawa, Ontario, Canada

To my younger brother

Abstract

Many of today's critical infrastructures, including Industrial Control Systems (ICSs), are evolving with the integration of numerous connected cyber components with legacy systems. This evolution has exposed ICS to a wide range of security vulnerabilities, requiring different cybersecurity considerations. Understanding how severely cyberattacks can exploit such system vulnerabilities to disrupt or delay system operations is paramount for developing targeted and effective defenses.

In this thesis, we present a four-stage impact analysis approach to observe and characterize the manifold impact of cyberattacks on ICS operations. Representative tampering and spoofing attacks demonstrated on a timed automata model of a manufacturing cell control system show how classical and statistical model checking, respectively, are effective at identifying and quantifying the severity of cyberattack impact on ICS mission objectives. Furthermore, the impact analysis results provide extensive insight into the varying impact caused by different attackers and how systems with defenses can mitigate such impacts.

Acknowledgements

This work began with a collection of fragmented ideas on viewing system security from a different perspective, which has since evolved into something much more. The work itself, and my growth during its progress, would not be possible without the sincere and unwavering support from my supervisor Professor Jason Jaskolka. His guidance, most times as a supervisor but at times as a guardian or even an elder brother, has helped me pull through the hardest of times. I am grateful beyond words.

I thank my colleagues for creating an earnest yet relaxing environment to blend in. A very special thank you goes to Joe Samuel for his overwhelmingly optimistic influence, and more importantly, for simply being a great friend.

I express my sincere gratitude to Dr. Mohammad T. Kawser, who, even during his unimaginably agonizing moments, guided me to explore the fundamentals of research. I thank Professor Abdur Rouf and Dr. Rifat Ahmed for being the two best teachers in my life and allowing me to clearly see where my interests and ultimate goals lie.

Lastly, I convey my heartfelt gratitude to my family for supporting me with everything they had throughout my journey so far.

Contents

List of Tables

List of Figures

List of Abbreviations

BNF Backus–Naur Form

CPS Cyber-physical System

DCS Distributed Control System

DoS Denial-of-Service

ICS Industrial Control System

ICS-CERT ICS Cyber Emergency Response Team

ICSs Industrial Control Systems

IoT Internet of Things

IT Information Technology

MCCS Manufacturing Cell Control System

OT Operational Technology

PCS Process Control System

PLC Programmable Logic Controller

RF Radio Frequency

SCADA Supervisory Control and Data Acquisition System

SMC Statistical Model Checking

TCTL Timed Computation Tree Logic

WAN Wide Area Network

Chapter 1

Introduction

Industrial Control Systems (ICSs), due to the integration of various cyber components with traditional legacy systems, require different considerations when it comes to security. In this chapter, we discuss how cyberattacks can cause various impacts on ICS operations and why a comprehensive approach to analyze such impacts is necessary. Section 1.1 introduces the threat landscape of ICS and how its security considerations are different. Section 1.2 details the manifold impact of cyberattacks on ICS and why their analysis is a necessity. Section 1.3 provides a brief survey of the existing literature and reveals the motivation behind this study. Section 1.4 states the problem that we are trying to confront. Section 1.5 summarizes the main contributions of our work. Section 1.6 provides a list of publications related to the work presented in this theses. Finally, Section 1.7 outlines the structure of the remainder of the thesis.

1.1 Why is ICS Security Different?

The goal to advance towards a safer, more secure, and resilient country involves the development of systems that are critical to this progress. In Canada's National Strategy, *critical infrastructure* refers to the processes, systems, facilities, technologies, networks, assets, and services essential to the health, safety, security, or economic well-being of the public and the effective functioning of government [Can09]. Critical infrastructure in Canada includes ten sectors, namely energy and utilities, finance, food, transportation, government, information and communication technology, health, water, safety, and manufacturing, among others. The continued existence and uninterrupted operation of these systems are not only imperative for progress, but more importantly, their failure may lead to catastrophic loss of human lives and the environment, serious damage to the economic infrastructure, and significant harm to the public confidence. As a result, systems that are involved in managing such critical infrastructure require special attention [HF+18].

Industrial Control System (ICS) refers to a broad category of monitoring and control systems critical to the automatic and intelligent operation of industrial processes, comprising many of today's critical infrastructures [MKW+16]. The collective term ICS can be used to describe different types of control systems and the associated instrumentation such as devices, components, and networks involved in the automated monitoring and control of system processes. Figure 1.1 shows the distinctions between different types of ICS that are sometimes used interchangeably, and thus incorrectly [BL04]. Depending on their area of control, the term ICS may mean (1) Supervisory Control and Data Acquisition System (SCADA), focused on supervisory

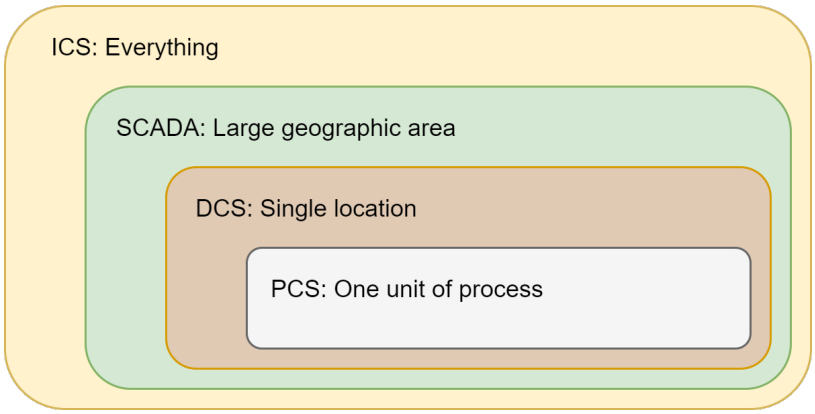


Figure 1.1: Different types of ICS

control and long-distance monitoring and control of field sites over a large geographic area, (2) Distributed Control System (DCS), used to control systems found in one location, and (3) Process Control System (PCS), used to control a single unit of process. Examples of ICS include manufacturing systems, wastewater processing systems, chemical reactors, transportation systems, and power management systems, among others.

In Figure 1.2, an example ICS is shown to the right connected via a group Wide Area Network (WAN) to a corporate IT network [Mica]. The overall ICS is divided into the supervisory network (SCADA) and the distributed production network (DCS). The SCADA system provides a centralized control system at the supervisory level, e.g., to acquire and transmit data from numerous process inputs and outputs. DCS, on the other hand, can use centralized supervisory control to manage multiple local controllers or devices. Overall, the SCADA system provides long-distance monitoring and control, whereas the DCS provides local control to manage the operation of a PCS. A PCS (e.g., a Programmable Logic Controller (PLC)), in turn, controls local

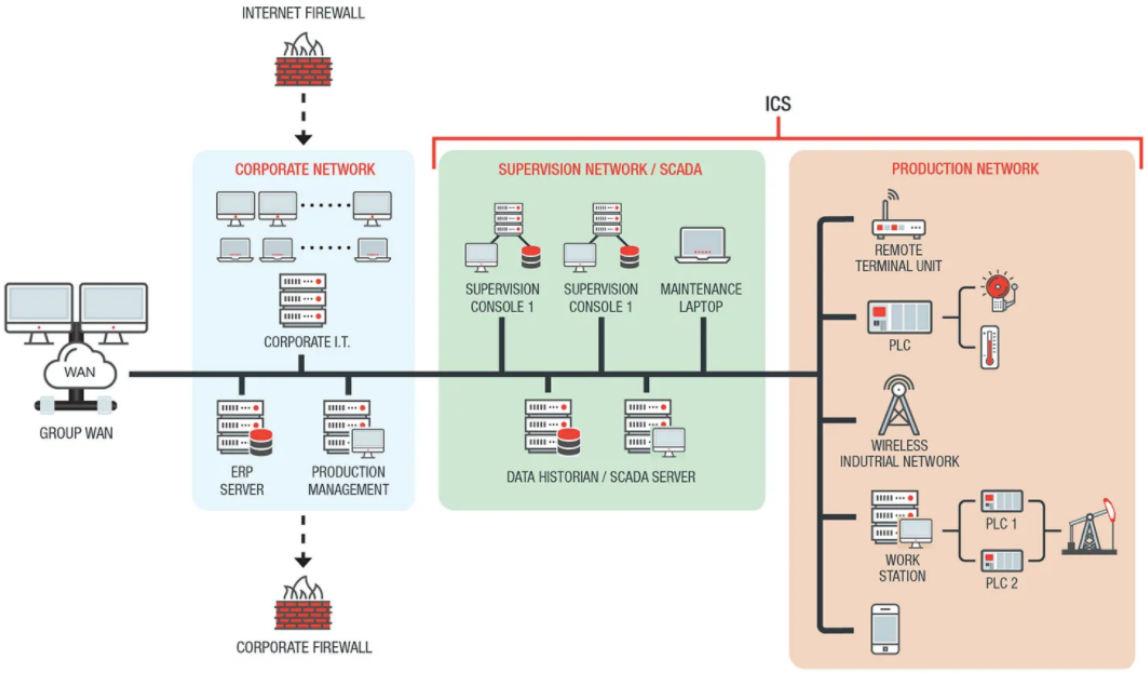


Figure 1.2: Various ICS components and their interactions [Mica]

operations such as opening and closing of valves and breakers, collecting sensor data, and monitoring the local environment for alarm conditions [Mica].

The long-standing myth that ICS are immune to cyberattacks has been proven wrong with the inclusion of open standards such as Ethernet, TCP/IP, and web technologies [BL04]. While the continuous integration of cyber components with legacy systems has made ICS highly interconnected and efficient, it has also made them prone to a wide range of new vulnerabilities and threats [RPK01]. The innate weaknesses of ICS and its underlying components was first revealed to the world by the discovery of Stuxnet [FMC11], a highly sophisticated malware that has driven numerous studies on ICS security since. Over time, attacks on ICS have

evolved to targeted attacks on the communication protocols as well as system operations [DSV15, GKA17, KG13, GTG+13, BL04, MKW+16], leading to potentially catastrophic disruptive events [Slo19].

The threats to ICS are accentuated by the fact that many ICS (e.g., manufacturing) are deeply intertwined with the entire product lifecycle and can be attacked anytime in the product lifecycle, as well as anywhere in the supply chain, to create a significant and sustained impact on the system [WCWW14]. The reliability of ICS systems is closely linked to the system design process and is not without its own issues [BK08]. In fact, the design process itself can be targeted by attacks to alter design files, process parameters, or the quality control systems [WCWW14]. These attacks can adversely affect a product's design intent, performance, or quality and lead to failed system deployment or reduced-quality end-products, posing a human safety risk to operators and consumers [WCWW14]. The traditional security requirements such as confidentiality, integrity, and availability also take a different meaning when considered for ICS. The definitions for these properties, translated for use in control systems, are adapted from [CAS08].

Availability in ICS: *Availability* is considered the most important property for ICS, and is interpreted as the capacity to maintain operations by preventing or surviving Denial-of-Service (DoS) attacks on the information collected by the sensors, commands given by the controllers, and physical actions taken by the actuators. *Timeliness* [ZJS11], the ability of the system to take timely actions, can be considered an availability issue due to the hard real-time requirements in the control domain.

Integrity in ICS: *Integrity* is considered to be the system's ability to prevent, detect and survive deception attacks. Deception in terms of ICS operations is the result of an authorized party (e.g., a controller) receiving false data and believing it to be true. The term *veracity* [Gol12], reflecting the trustworthiness of the assertion, translates into ensuring that sensors faithfully capture measurements of process state or controllers, or issue truthful commands as a result of control algorithm execution. Veracity also applies to process information storage as their trustworthiness or the historical dynamic of process data is used to make decisions by many advanced control algorithms and can be considered an integrity issue.

Confidentiality in ICS: Finally, *confidentiality* is interpreted as the ability to prevent an adversary from inferring the state of the physical system by eavesdropping on the communication channel between the sensors, controllers, and actuators. For ICS that are not distributed, the threat to confidentiality primarily comes from insiders (e.g., disgruntled employees) [WCWW14] and targeted attacks (e.g., ransomware). Otherwise, attacks on confidentiality may be considered less consequential when compared to availability and integrity.

ICS security is more than ensuring availability; modern ICS are a combination of legacy Operational Technology (OT) systems with Information Technology (IT) components for enhanced connectivity and automation of processes. IT systems refer to the entire spectrum of technologies for information processing, including software, hardware, communications technologies, and related services. OT systems, on the other hand, refer to the hardware and software that detects or causes a change through the direct monitoring and/or control of physical devices, processes, and events. IT

and OT have vastly different characteristics and priorities: in IT systems, protection of the information (confidentiality) that it stores, processes, or uses are the highest priority, whereas, in OT systems, the priority falls on ensuring the uninterrupted operation (availability) of the involved systems and components.

The cyberinfrastructure used to monitor and control the physical processes inside an ICS gives rise to an interesting dynamic where neither IT security nor OT security alone is enough to prevent attacks on ICS from occurring. Attacks targeting the cyberinfrastructure may end up affecting the control mechanism, causing physical damage to equipment, disrupting the system processes, and/or leading to cascading effects on the ICS operations [KG13]. In light of the fourth industrial revolution, the security of ICS must also consider the integration of various enabling technologies such as Internet of Things (IoT), cloud computing, and big data analytics [CRFAF17]. It means that on top of the information protection mandated by traditional IT security, ICS security must additionally safeguard the availability of system components and ensure the integrity of system operations.

All of the discussion above points to the fact that ICS security, due to its characteristic elements, must be considered differently. The same is true for the development of appropriate countermeasures against attacks on ICS; while the majority of the software, hardware, and communication protocol vulnerabilities are well understood, suggested security requirements and solutions to prevent and/or mitigate attacks on ICS are often economically infeasible and/or beyond the technical capabilities of the legacy systems involved [KG13]. To make matters worse, any countermeasure also comes with its own performance, cost, and usability tradeoffs. As a result, system

developers must have a way to prioritize system vulnerabilities and develop targeted defensive mechanisms for ICS, for which they will need to know where best to utilize the often limited available resources. And therefore, our investigation in this thesis is a work towards defining a solid criterion upon which prioritization of ICS vulnerabilities can be based so that our limited available resources are put to the best use possible.

1.2 What is Cyberattack Impact?

Two of the many reasons security is of the least concern among many system developers and operators are disbelief or unawareness that system security is a necessity. This is exemplified by the fact that about 20% of the system vulnerabilities, even when detected, are ignored for about a month by product vendors [GTG+13], or how insider negligence with a single infected USB drive can result in extended plant downtime of three weeks [Cyb12]. Even when the intention to secure systems is there, in practice, limitations imposed by time and resource scarcity often force developers to make compromises in what security mechanisms are implemented in the system. The negligence, along with the fact that system developers are constrained by what they can do with their limited available resources, leads to the development of unprotected systems, and consequently, a perfect hunting ground for attackers.

Even if some developers were to show prudence, the implementation of typical IT security countermeasures has its own limitations. Depending on the rigor of the countermeasure, it may have a negative effect on the system's performance [SDF17]

and prove to be inadequate, especially in the face of hard and/or impossible to detect attacks such as zero-day vulnerability exploits. A different way to contain cyberattacks is to reduce the impact of successful cyberattacks by improving the resilience of systems against such attacks [GD14]. After vulnerability identification, the next step can focus on determining the kind and severity of impact such vulnerabilities may expose if they were to be exploited to mount a cyber-attack.

Cyberattacks can severely affect the safe and reliable operation of an ICS and negatively impact the system's mission objectives. Of particular importance is the adverse effect on the system operations to disrupt or delay operations, damage equipment, and reduce customer trust, among others [JJ21b]. Can an attacker exploiting a particular vulnerability in a nuclear command and control system merely turn a light on and off on a control panel, or can they cause a complete nuclear meltdown? These are two vastly different possibilities and knowing which one is possible (and/or more likely) for a given system is a critical piece of information that can be used to assess where and how to spend valuable resources for mitigating the existence and particular vulnerabilities in the system.

In the last two decades, ICSs in various parts of the world have seen attempts at malfunctions using cyberattacks, many of which have succeeded in disastrous consequences [HF+18]:

1. **Maroochy Water:** In March of 2000, the Maroochy Shire Council in Queensland, Australia, experienced problems with its new wastewater system that led to incorrect operation of pumps and the alarms used to notify faults [HF+18]. On April 23, 2001, the police captured the perpetrator, an insider, who used

a laptop, a radio transmitter, and specialized SCADA equipment to interfere with the RF signals used in the wastewater system [Smi01]. Over a three-month period, the attacker controlled over 150 sewage pumping stations and released millions of gallons of untreated sewage into the waterways and local parks [SM07].

2. **Aurora:** In 2007, researchers from the Idaho National Laboratory demonstrated the "Aurora" attack on an electric grid [Mes07]. The attacker gained access to the control network of a diesel generator, ran a malicious program to rapidly open and close the generator circuit breakers out of phase from the rest of the grid, resulting in the diesel generator exploding [MKW+16]. The attack was a demonstration of how cyberattacks could damage and even destroy physical components of a system, alerting how severe the consequences could be if such attacks were to be done on a large scale.

3. **Turkey Pipeline Explosion:** In August 2008, a pipeline in Turkey was hit by an explosion causing the spilling of 30,000 barrels of oil in an area above a water aquifer [MKW+16]. The attack was caused by intrusion using vulnerabilities of the wireless camera communication software and then tampering with the alert units and the PLCs to increase valve pressure leading to the explosion. This attack is regarded as a physical attack rather than a cyber intrusion by some researchers [HF+18]. Nevertheless, the explosion ended up costing British Petroleum $5 million USD a day in transit tariffs.

4. **Stuxnet:** Stuxnet (2010) is unanimously considered to be the most sophisticated worm targeted towards ICS that infected PLCs in 14 industrial sites

in Iran, including a uranium enrichment plant [FMC11]. The attack included intrusion via infected USB drives and stealthy self-propagation through the network by exploiting four unpatched Microsoft vulnerabilities [HF+18]. Stuxnet, believed to have damaged one-fifth of the nuclear centrifuges in Iran, was a wake-up call for the security of critical infrastructure around the world.

Following the massive impact in Iran caused by Stuxnet, researchers all over the world focused on the capabilities of sophisticated malware and security of ICS in general. However, the attackers did not stay still, and attacks on ICSs continued to create substantial impacts even to this day. As an example, just in the year 2014, the ICS Cyber Emergency Response Team (ICS-CERT) had to respond to 245 cyber incidents [MKW+16]. Some of the more recent cyber incidents targeting ICS include:

1. **German Steel Mill:** In December 2014, a report from the government of Germany briefly described an attack on an unspecified German Mill through spear-phishing and social engineering tactics [HF+18]. The attackers were reported to have an advanced understanding of ICSs, including knowledge of the target steel plant process, that allowed them to cause multiple failures of individual control systems. All of these prevented a blast furnace from being able to shut down in a controlled manner, resulting in massive damage to the plant.

2. **Ukraine Power Grid:** In 2015, just two days before Christmas, the world saw the first successful cyberattack on the power grid [HF+18]. The attack is believed to be caused by the "BlackEnergy" malware, planted onto the company's network using spear-phishing emails, that exploited macros in Microsoft Excel documents. The attack shut down 30 substations and left almost 230,000

people without electricity for up to six hours. Furthermore, it also disrupted the use of SCADA equipment, meaning that the power restoration had to be done manually, leading to further delays in the restoration efforts [Zet16].

3. **US Colonial Pipeline:** More recently, a US Colonial Pipeline that carries 2.5 million barrels a day was attacked by ransomware in May of 2021 [Rus21]. A cybercriminal gang called "DarkSide" is believed to be behind the attack who infiltrated the pipeline network, stole almost 100 gigabytes of data, threatening to leak the data onto the internet unless a ransom of 4.4 million USD was paid. The attack halted the plant operation for six days, leading to a fuel crisis and increased prices along northern South Carolina to southern Virginia.

4. **JBS Foods:** Just a few weeks after the US Colonial Pipeline attacks, another ransomware attack followed in June 2021, this time on one of the world's largest meat producers JBS Foods [Mor21]. JBS Foods ultimately had to pay $11 million USD in bitcoins in order to make a deal to prevent further attacks. Overall, the attack resulted in a massive impact on the meat industry, forcing the temporary closure of all of its beef plants in the United States. Furthermore, impacts on an international scale ensued, affecting one Canadian plant and halting all beef and lamb production in Australia.

All of the incidents above portray how severely a successful exploit can negatively affect ICS mission objectives. The clearly visible impact of such attacks gives rise to an interesting line of thought: if we had a way to show system developers concrete evidence beforehand (e.g., through attacks on system models) that cyberattacks can severely impact the operations of the systems that they are building, the view that

"security considerations for ICS are unnecessary" may start to change. Additionally, if we could pinpoint which attacks were the most impactful, developers would be able to prioritize countermeasures based on cyberattack impact, and developing systems with defenses may become a norm. For example, mitigating attacks that can permanently disrupt the system activities can be prioritized over attacks that can minimally affect a single process. The *impact of attacks*, thus, may end up becoming the criterion that we were looking for to incentivize the development of targeted and effective security controls.

The impact of cyberattacks on standalone IT communication networks has been studied extensively in the literature [HQD+03, SHÇ+10]. In OT systems, however, the priority is drastically different, as availability is considered the most salient security property [SWW15]. This leads to two important observations. First, regardless of its maturity, impact assessment frameworks developed for IT networks will not be equally effective and/or applicable to OT systems [GWP19]. Second, in systems where IT and OT systems are equally important, such as ICS or cyber-physical systems (CPS), cyberattacks will not be limited to creating only cyber impacts, but will inevitably lead to physical impacts and/or cascading effects through system operations. In other words, the development of more secure and reliable ICS mandates the development of an impact analysis framework tailored to this IT-OT convergence.

1.3 The Challenges

Modern ICS and its highly coupled physical and cyber components are novel technologies, which are not readily available for security evaluation [GWP19]. Tests on live systems are challenging since these systems usually have very high availability requirements. On the other hand, the construction and maintenance of physical testbeds to mimic and study the behavior of large ICS are costly. Conducting safe and practical experiments is also not feasible due to the cost of testing and reproducing damaging manipulations [KG13]. A widely adopted alternative is to use models of system components to simulate cyber-physical processes, and thus, assessing the impact of attacks via simulation has been identified as a prominent research topic in the literature [CL07, HKVW15]. While simulations alone can be used to evaluate the impact (similar to contingency analysis for smart power grids), more meaningful results can be achieved by considering an integrated model of the cyber-physical components to identify the combined impact of attacks due to a successful exploit [SL14].

To get ourselves familiar with the various modeling and simulation efforts that explore cyberattack impact analysis, we examined the existing literature. Most relevant to our work was that of Li et al. [LXDW16], who presented false sequential logic attacks using improperly sequenced control commands and proposed an impact ranking model. Sultan et al. [SDF17] presented a methodology to assess the impact of attacks and countermeasures on a vulnerable system using model checking on various timed automata system model mutations. Other modeling and simulation studies included the use of reliability graphs [GWP19], impact dependency graphs [Jak11], attack graphs

with functional dependency attributes [AHT20], reachability graphs [CPE+17], attack trees [MAM+18], Bayesian networks [HZT+18, Mey20], Petri nets [CMM+14], influence Diagrams [KMKP16], geometrical modeling [GGGAD17], functional dependency network analysis [GD14] and business process modeling notation [MTT+11], among others. Other works put less emphasis on the modeling aspect and emphasized the use of simulations [TBG16, CBPK13, YLG11]. Lastly, a few works used available system testbeds [LFK14, SL14] to analyze the impact of attacks. A more detailed description of these works are presented in Chapter 2.

The exploration of the different modeling and simulation-based methods and metrics used to define cyberattack impact on ICS and CPS led to the revelation that the topic of impact analysis is still rather new and lacks consistency in the primary terms such as what *impact* really means. In the studies discussed above, impact took many different definitions depending on the type of system and what was considered to be the most salient properties for its mission objectives [JJ21b]. This included impact on the system availability (disrupted operation), additional cost (to repair or replace damaged physical components and systems), acceleration of natural system degradation, and even impact on other components and/or systems due to cascading events. The aspect of time became important when the duration of cyberattacks was taken into account to define resulting timed consequences (process delays, system downtime) on the system operations. In our study, **impact is defined as a violation of ICS mission objectives that may ultimately lead to disrupted, delayed, or halted ICS processes or damaged physical infrastructure**.

The study also revealed a few other concerning aspects about the existing impact analysis approaches:

Overlooking the broad picture: First, we noticed that the number of research articles on the smart grid as well as its SCADA systems was overwhelmingly large compared to that of other ICS and CPS. But we must bear in mind that the ICS and CPS that control systems related to our daily needs (e.g., manufacturing, oil, gas, telecommunication systems) as well as systems related to our general well being (wastewater treatment systems) are equally important. Therefore, research is needed to identify and analyze the impacts on such system operations, and perhaps, on human lives due to cyberattacks. These considerations also mean that the *development of impact analysis approaches that are generic and can be reused* for all ICS or CPS in general is a necessity.

Misunderstanding the problem landscape: A majority of the studies also only investigated the impact on the system availability and consequences due to that. In essence, these studies were treating ICS and CPS as traditional OT systems or legacy systems where the primary concern was about system availability. However, as discussed before, ICS security is more than just about availability, and therefore, *the notion impact on ICS must also be expanded to include other security objectives* such as confidentiality or integrity impacts on the cyberinfrastructure, or how attacks targeting such security objectives on the cyberinfrastructure can end up impacting the physical system processes.

Domain-specific impact metrics: Similar to the overwhelming number of studies focusing on smart grids and their SCADA systems, a majority of the studies look at

particular types of systems (e.g., only power systems or only manufacturing). As a result, the defined metrics that were used to identify and/or measure impact were also domain-specific. For example, loss of load expectancy, energy index of reliability, voltage stability index, or cost of losing each branch [TBG16, CBPK13, HWY17], are all metrics that are useful when we are only concerned about power systems but becomes useless as soon as we turn our attention towards other types of ICS. This gives rise to the questions of how such domain-specific metrics can be reused for ICS in general or whether the authors are even thinking about developing reusable metrics when they are coming up with such metrics. These events bring us to the conclusion that *metrics used to define impact on ICS must be domain-agnostics and reusable.*

Granularity of the perspective: The granularity considered by the discussed approaches tends to differ vastly as well. Some studies only examine whether the system is in a steady state or not, thus defining impact, whereas others investigate the impact on individual system components and/or systems composed of other systems. Studies that consider the impact on the overall system state tend to miss out on impacts at component-level processes. For example, some attacks may not cause noticeable impacts by forcing the system away from its steady-state but may cause smaller impacts such as concentration change in a tank of a chemical reactor system [LXDW16]. Such a small impact may, over time, lead to bigger consequences; the small concentration change may end up causing a disqualification of the final product and thus direct economic losses (a major impact if sustained). Therefore, we need to develop impact analysis approaches with *flexibility in granularity* so that the overall system operations, as well as salient system processes, are taken into account.

The aspect of time: Since ICS involve numerous real-time physical processes, a timed perspective to analyze ICS security can enable the analysis of impacted system operations (e.g., delays during operation) or the system's evolution during cascading attacks over time. While various studies look at impact analysis, many (e.g., [TBG16, KMKP16, HWY17, Mey20, MTT+11, GGGAD17, GD14]) do not consider the real-time system constraints. As a result, important considerations about the system's timeliness, such as whether the system mission objectives can be reached within time or how to identify and measure the delays in system operations due to cyberattacks, were often left unanswered. Therefore, to comprise a comprehensive impact analysis framework for ICS, *timed analysis approaches* illustrating how severely cyberattacks can impact system operations, over time, is necessary.

In summary, the majority of the existing impact analysis approaches discussed above are limited in their applicability, reusability, flexibility, and scope of investigation when analyzing the impact on time-critical ICS operations. These challenges became the inspiration behind us delving into the study of the impact and the development of an impact analysis approach tailored to ICS.

1.4 Problem Statement

Due to the IT-OT convergence, the security requirements for ICS are different from those of traditional IT and OT systems. The existence of the cyber control infrastructure makes ICS vulnerable to many forms of cyberattacks, which may affect the safe and reliable operations of the system, impacting the system's mission objectives. The

criterion of cyberattack impact can be used to prioritize ICS vulnerabilities so that targeted countermeasures can be built with our limited available resources. Therefore, the development of a systematic impact analysis approach tailored to ICS is paramount.

The limitations in the existing impact analysis studies discussed in Section 1.3 enabled us to determine a set of five research objectives for this study. To deal with the challenges, the overall objective became the development of an ICS-centric impact analysis approach that would:

RO1. be widely applicable to ICS in general,

RO2. consider the impact on different security objectives pertaining to ICS,

RO3. define impact metrics that are domain-agnostic and reusable,

RO4. have flexibility when modeling and analyzing systems with different granularity, and

RO5. take a timed perspective in analyzing impact.

In light of our research objectives, the work presented in this study is an effort to show, with quantified evidence, that cyberattacks (if successful) can create a sustained and measurable impact on ICS operations. More precisely, the study aims to satisfy the research objectives while answering four questions:

RQ1. Can we identify the impact when cyberattacks aimed at the systems operations of an ICS impact the achievement of its mission objectives?

RQ2. Can we quantify such impacts of different attacks to characterize them based on severity and inform appropriate defensive mechanisms?

RQ3. Can we characterize the change in impact when the same attack is done by different types of attackers?

RQ4. Can we identify how the addition of various system defenses affect the notion of impact and how they compare to defenseless systems?

1.5 Main Contributions

The main contributions of this work are as follows:

- **A survey of existing modeling and simulation-based impact analysis studies on ICS and CPS (Chapter 2)**. In particular, we evaluate the applicability and limitations of works that propose different ways to identify the physical impact of cyberattacks on systems where IT and OT systems converge, and to which extent this is done in a systematic and formal manner.

- **Development of a systematic ICS-centric impact analysis approach (Chapter 3–Chapter 7).** To study the behavior of an ICS under various attacks, the four-stage approach uses timed automata [AD94] to model the real-time behavior of ICS and potential attackers in Uppaal [BDL04]. We use classical model checking [BK08] to identify the existence of an impact on ICS mission objectives (i.e., answer **RQ1**.), and Statistical Model Checking (SMC) [BDL$^+$12] to quantify such impacts (i.e., answer **RQ2**.).

- **Analysis of the behavior of different types of attackers (Chapter 5–Chapter 7).** We develop generic and extendable attacker models to capture timed spoofing and tampering attacks on ICS operations. Our impact analysis with different attacker types enables us to observe the differences in their behavior and how they can create varying impacts on ICS mission objectives even with the same attack (i.e., answer **RQ3**.).

- **Examination of the effectiveness of different system defenses (Chapter 8).** We extend the system model with various defensive capabilities to compare and contrast the impact on protected and defenseless ICS. With changes to the system model, we show how only selected activities in the proposed approach need to be reevaluated to characterize changes in the notion of impact (i.e., answer **RQ4**.).

1.6 Related Publications

The following are a list of publications related to the work presented in this thesis.

- A. Jawad and J. Jaskolka, "Modeling and simulation approaches for cybersecurity impact analysis: State-of-the-art," in 2021 Annual Modeling and Simulation Conference (ANNSIM), pp. 1–12, IEEE, 2021.

- A. Jawad and J. Jaskolka, "Analyzing the impact of cyberattacks on industrial control systems using timed automata," in 2021 IEEE International Conference

on Software Quality, Reliability, and Security (QRS), pp. 1–12, IEEE, 2021. (To Appear).

1.7 Structure of the Thesis

The remainder of the thesis is organized as follows:

Chapter 2 presents a detailed survey of the existing modeling and simulation-based impact analysis studies and their limitations.

Chapter 3 provides an overview of the activities involved in the proposed impact analysis approach and the required prior information.

Chapter 4 describes how to model and verify the system as a network of timed automata in UPPAAL.

Chapter 5 introduces different types of attackers and details modeling the behavior and capabilities of such attackers.

Chapter 6 simulates the modeled system and attackers in parallel to observe the execution of different attack strategies on the example system and varying impacts on the system operations.

Chapter 7 shows how results from SMC queries can be analyzed to quantify the impact of various attacks performed by different attackers on the example system.

Chapter 8 extends the system model with various defensive capabilities and explores their resiliency against cyberattack impact compared to defenseless systems.

Chapter 9 assesses the strengths and limitations of various aspects of the proposed

impact analysis approach.

Chapter 10 provides concluding remarks and briefly discusses the directions for future research.

Chapter 2

Literature Review

The impact of cyberattacks has been defined, analyzed, and characterized using many different approaches in recent years. In this chapter, we survey the existing literature that used modeling and simulation-centric methods to analyze the impact on ICS. Section 2.1 discusses the diverse graph-based approaches used to model the system behavior and analyze the impact on the modeled system. Similarly, Sections 2.2 and Section 2.3 explore studies that use various automata-based techniques and attack tree-based approaches, respectively. Other model-based approaches that do not fall into these three broad categories are discussed in Section 2.4. Additionally, studies that do not involve a modeling stage but rely primarily on simulations are examined in Section 2.5. Pertinent observations from surveying the literature are summarized in Section 2.6. Section 2.7 outlines the key elements and outlines the objectives for our impact analysis approach. Finally, Section 2.8 concludes our discussion of the topic.

2.1 Graph-based Approaches

Graphs are mathematical structures that represent a pairwise relationship between a set of objects and are well suited to model the granular operations in CPS or ICS in a convenient and compact manner [KFL+10]. A graph G is defined as an ordered pair $G = (V, E)$, where V is a set of vertices (also called nodes) and E is a set of edges. A vertex $v \in V$ may be used to represent the states or the functions performed by a component and/or system, whereas an edge $e \in E$ may be used to connect vertices to show relationships or relate dependencies between them. Edges may or may not have directions, leading to directed or undirected classes of graphs, respectively. Furthermore, vertices and edges may be associated with additional metrics (e.g., operational capabilities, probabilities) to diversify the way graphs can be used, leading to powerful analysis approaches.

Kundur et al. [KFL+10] proposed a graph-theoretic dynamical systems paradigm to quantify the physical impact of tampering attacks on an electric grid. In their approach, each node of the graph is associated with state information governed by dynamical system equations to model the physics of physical electrical components or the functionality of the cyber elements of a smart grid. This provides a flexible framework to model the cause-effect relationships between the discrete-event cyber and the continuous-time electrical system state that can ultimately be linked to power delivery metrics to provide information about the degree of disruption enabled by a cyber attack. Through MATLAB and Simulink simulations, a case study involving a single generator system demonstrated how adding biases in the system may lead to incorrect shedding or serving of loads or how the generator frequency can be decreased

to force a complete system blackout. A larger case study [KFM+11] on an IEEE 13 node test feeder system using PSCAD simulations show how similar results can be achieved on a microgrid test system operating in islanding mode.

More recently, Giehl et al. [GWP19] proposed a framework where reliability graphs are used to assess the impact on manufacturing systems. Each edge of the reliability graph is assigned a probability that represents its reliability, and the overall system reliability is measured by the averaged sum over the availability of the system components. The availability of each component is determined by combining the mean-time-to-failure (MTTF) with the mean-time-to-repair (MTTR) for repairable systems. An attacker function is considered that can expedite system degradation over time by reducing the overall system availability below a certain threshold. Stealthy and non-stealthy attacks performed using a Delev-Yao attacker model [DY83] on specific components of the assembly lines of an experimental cellular manufacturing environment showed immediate and subtle degradation effects leading to reduced availability and eventually an unusable assembly line.

Alternatively, Cam et al. [CMM+14] presented time Petri net models, which are continuous-time Markov processes to integrate the impact and timing relationships with firewall, vulnerability scanner, and recovery operations. A Petri net is a formal modeling technique that is a directed bipartite graph containing two types of nodes, places, and transitions, connected via directed arcs from places to transitions or vice-versa to represent input-output relationships. The authors used time Petri nets to create an attack model to define the impact of attacks on distributed control systems using a time-dependent impact factor (TIF). TIF takes into consideration the timing

of an attack to measure the degree of compromise to an attacked asset. A directed graph is used to represent an influence model, where first-order linear differential equations are used to calculate the influences of assets at a certain node on other nodes at a given time to characterize the impact of influences.

Jakobson [Jak11] used impact dependency graphs as a mathematical abstraction of the domain semantics of assets, services, mission steps, missions, and all of their dependencies of a cyber terrain (CT). CT is a multi-level information structure containing three sub-terrains (hardware, software, and services) which could be considered a living environment for mission agents to live or be threatened when the CT is weakened or destroyed by cyber attacks. Assets (hardware, software, and services) were associated with operational capacities that may be reduced by an attacker or be reset by a human. Impact is calculated via an impact factor (IF), measured by $IF = VS/10$, where VS represents the vulnerability score calculated using common vulnerability scoring system (CVSS) metrics [MSR06]. The mission impact assessment is performed by doing a primary target impact assessment and then considering the impact propagation through the CT.

Huang et al. [HZT+18] proposed an approach where Bayesian networks (BNs) are used to model the attack propagation process on a boiling water power plant to infer the probabilities of sensors and actuators to be compromised. A BN is a directed acyclic graph where a node represents a variable whose state depends only on the state of its parent node, and directed arcs represent the conditional dependencies between the variables. The approach uses CVSS metrics to derive the conditional probability table (CPT) parameters that are fed into a stochastic hybrid system

(SHS) model to estimate the evolution of the controlled physical process. A new security metric, named mean-time-to-shut-down (MTTSD), is proposed to describe the expected amount of time needed by an attacker to drive the physical process to an undesirable state. MTTSD can be used to quantify the cyber-to-physical risk by evaluating the system availability with the SHS model. The method, however, does not work with CPS that cannot be handled through linearization and does not consider existing recovery mechanisms that may restore the steady-state of the system shortly after an attack.

Very recently, the Multi-Attribute Vulnerability-based Criticality Analysis (MAVCA) model was proposed by Ani et al. [AHT20]. The MAVCA model includes the estimation of a criticality index (CI) as its first stage. The CI is a function of two parameters: *vulnerability exploit potential* and *vulnerability impact potential*. The *vulnerability impact potential* is defined as the influence on an ICS when a vulnerability is successfully exploited. More specifically, the impact can be evaluated based on technical vulnerabilities in the ICS, the likelihood of exploiting the vulnerabilities, and a vulnerable component's functional dependency relative to other ICS components. The MAVCA model combines CVSS temporal and environmental attributes with attack graph probabilities and functional dependency attributes to assess the security state (e.g., severity and impact) of the system. The resulting analysis showed that a vulnerability may have varying exploitation and impact potentials in different network environments and may yield different criticality indices.

Other works such as [CPE+17] construct a reachability graph based on the number of existing security rules and the data access relationship between subsystems of a distributed wastewater control system. Impact is defined as the quantity of unprocessed load, attack duration, and material damages (i.e., pumps, sensors, area flooding), and impact scores are computed using CVSS metrics. Zhang et al. [ZWW+20] proposed an attack graph generation algorithm and performed a quantitative assessment based on attack graphs in order to find critical paths in ICS and show how the attacked state of upper nodes have a high impact on lower nodes. In [HWY17], the authors proposed a framework of electrical CPS where each bus in a power grid is equipped with a controller and a sensor to devise a highly intelligent smart grid. A case study on an IEEE 39-bus system modeled using graphs and simulated using MATLAB showed how attacks intended to trip a branch may cause several consequences, such as damage due to lost transmission lines or cascading failures. The impact is measured as the physical damage and the economic cost of losing each branch leading to the additional cost of adjusting generator outputs.

Graph-based approaches provide a simple and intuitive way to model systems with varying granularity. The complexity of processing in any such approach is dependent on the size of the graph (number of nodes), the connectivity (number of edges), and the particular dynamics (related to its order), if applicable, used to model the nodes [KFL+10]. Purely graph-based approaches, however, are not sufficient to model the state changes within the physical system [ES09]. Another problem arises when trying to model the physical portion of a cyber-physical system as graphs, by themselves, do not account for the unique characteristics of the system at various timescales [KFL+10].

2.2 Automata-based Approaches

For cyber-physical processes, control commands can be considered the trigger of an event for the physical process, and chains of commands can be regarded as discrete behavioral events [LXDW16]. These systems can therefore be modeled as discrete event systems such as finite state machines (FSMs), Petri nets, generalized semi-Markov processes, etc. Among these, FSMs, also known as finite state automata, are the most straightforward consisting of four main elements: 1) states to define behaviors, 2) transitions defining movements from one state to another, 3) transitions to allow movement once certain conditions are satisfied, and 4) external or internally generated events to trigger transitions [JHH12].

One such study that uses FSMs to study the impact of a novel attack is detailed in [LXDW16]. The study presented a new type of CPS attack on SCADA systems, named *false sequential logic attack*, that aims to disrupt the physical process of a system by using licit control commands in an improper sequence. Various sequences of attacks on a modeled three-tank system are simulated using MATLAB and Simulink, and the results suggested that the attack could cause varying impacts starting from simply causing economic losses to affecting the physical world, such as, equipment damage, casualties, and destruction of production facilities. To further categorize the impact, the authors propose an impact ranking model based on the evaluation of three elements. The elements are *effect duration*, *disturbance degree*, and *damage degree*, which are indicators that represent the duration of the effect, the degree of disturbance to the proper physical process, and the degree of potential damage to the physical system caused due to an attack, respectively. The final impact score is

given by an equation that is a function of all the three elements discussed above. The rankings are, however, decided for only a single observed variable and are relative to the attacks under discussion only. Additionally, no weights are assigned to the specific elements, rather damage is prioritized over disruption, which is prioritized further over the duration in case of a tie in the ranking.

Another study [SDF17] provided a detailed description of evaluating the impact of attacks and countermeasures on a vulnerable system using model checking on various timed automata system model mutations. Timed automata is a hybrid mathematical modeling formalism where a finite set of real-valued clocks is used to represent continuous time in a discrete-event system [AD94]. For systems involving various cyber-physical interactions, timed automata provides the advantage of modeling the cyber and physical portions of the system using discrete-event and continuous-time dynamics, respectively. The study is performed on a naval vessel where various models are constructed based on information from entities such as naval doctrine, missions, functional and physical architecture, etc., over the vessel life cycle. Timed automata are used to create a behavioral model of the system architecture, and deduced safety properties, representing the mission goal, are used to verify the correct behavior of the model through model checking. The system modeling phase is followed by performing a set of model mutations from descriptions of vulnerabilities, countermeasures, and attacks. Automata mutations are first performed to create mutant models for vulnerable and a patched systems. A new set of model checks are performed again to measure the capability of the vulnerable and patched systems to satisfy the safety properties. Further model mutations are led to compose both vulnerable and patched systems with attack automata followed by another set of model checks to identify the

impact of the attacks on the system mission. Finally, impact matrices are deduced from all the model verifications, and the differences allow to compute the impact of attacks as well as the impact and efficacy of various countermeasures to rank them accordingly. As with any state-based model, however, the state space explosion problem is evident, and the authors try to provide some recommendations to contain the resulting exploration time in the case of complex real system models.

2.3 Attack Tree-based Approaches

Attack trees are an emerging approach to present simple visualization of attacks against a system as a tree to enable analysis of attacker behaviors to mitigate threats against a system [MAM+18]. An attack tree is a hierarchical structure where the root node represents the primary goal of the attacker, and the leaf nodes represent different ways of reaching the primary goal. Any node can be decomposed into a set of easier to accomplish subgoals or atomic activities. Atomic activities represent independent exploits that require no further decomposition. The set of activities can be either disjunctions (OR nodes) or conjunctions (AND nodes), respectively.

Maciel et al. [MAM+18] present a way to evaluate the impact of Distributed Denial-of-Service (DDoS) attacks and malicious software on the availability of computer systems using hierarchical attack tree models. To determine the damage to the target system, they propose the concept of the pain factor (PF). PF is calculated by evaluating several factors such as the operational, financial, and reputation losses that the target system must bear for the successful execution of an attack step represented

by an attack tree node. The values for the losses must be obtained by adopting corresponding indicator profiles [Ing21]. Their approach also used impact values ranging from 1 (None) to 10 (High) based on minor to complete compromise of a system, and impact of exploited threats was extracted from available information security reports [ACBS17, Cat10].

Another work [Mey20] showed how attack trees can be effectively modeled using BNs to represent probabilistic attacker behavior against different LAN models. Nodes in the BN represented vulnerabilities and target conditions and edges represented the path between them. The probability of reaching each node depends on the conditional probability of its parent node. Each node was associated with the time to compromise each vulnerability and each edge was associated with the probability of successfully reaching a target condition by a successfully exploit. They defined the net impact on SCADA systems associated with power systems as a combined impact on the cyber-physical system. The physical impact is defined by performing dynamic simulations for each contingency and numerically evaluating the impact (maximum frequency and voltage deviation) on the set of compromised human-machine interfaces and intelligent electronic devices. The cyber impact is determined similarly by analyzing the physical impact, and the attack efficiency and risk on the SCADA system.

While graph and automata-based techniques are suitable for modeling the system dynamics, attack trees and the resulting Bayesian networks are particularly useful for modeling the sequence of attack steps or the propagation of attacks to obtain probabilistic measures. The reliability of attack tree-based approaches depends upon the accuracy of the values assigned to individual nodes, i.e., already available information

for an attack and/or impact on a system. This indicates that in cases of unavailable or are inadequate information, these methods become ineffective or inaccurate due to the missing and/or assumed values for nodes. The sequence of attacks is not typically included in attack tree modeling, making it less expressive than both graph- and automata-based approaches in representing system or component dependency. The concept of time is also not involved, as there is no way to model timing of system actions using attack trees.

2.4 Other Model-based Approaches

Various other methods have been used to model the operation of a system to study the impact of attacks. Such modeling techniques include Geometrical Modeling [GGGAD17], Influence Diagrams [KMKP16], functional dependency network synthesis [GD14], and Business Process Modeling Notation (BPMN) [MTT+11], among others.

Functional dependency network analysis (FDNA) has been used in [GD14] to identify critical systems and the critical links in System-of-systems (SoS). SoS represent a complex aggregate of systems where the constituent entities possess partial or full operational and managerial independence where the behavior of the overall SoS is determined by the interactions between the constituent systems. The concern for SoS is that the degradation occurring due to an attack in one part of the system may cause a critical loss of operability in other systems in the SoS leading to a cascading effect throughout the system. FDNA network synthesis allows SoS to be modeled

as a network where nodes represent the component systems and their capabilities, and edges represent the operational dependencies and the information flow. They defined the internal health (called Self-Effectiveness) and the properties (strength and criticality) of the dependencies for each node (system) to quantify the operability of that node using FDNA. The operability, ranging from 0 to 100, is defined as the percentage of effectiveness representing the level at which the desired operational capability of the system is currently being achieved. Experimental results on a Littoral combat warfare SoS showed how internal cyberattacks on a system can cause internal operability degradation as well as externally affecting all dependent systems. Additionally, attacks on communications links demonstrated how the operabilities of predecessor nodes are not affected but dependent systems can still be affected due to disrupted input. The method allows one to identify critical systems and critical links and rank nodes and links based on the criticality of the impact of attacks on the operability of the entire SoS.

The work of Gonzalez et al. [GGGAD17] takes a novel polytope-based approach to model the impact of attacks and countermeasures in a given system. In their approach, services, attacks, and countermeasures are represented in an n-dimensional coordinate system and the impact of each event (attacks, countermeasures) on a given service can be computed using geometrical operations (i.e., length, area, volume, hyper-volume). In the case of each event, the number of affected dimensions determines the size and impact of the event, i.e., the higher affected dimensions result in higher impact. For example, an attack represented as a volume (can affect three service dimensions such as resources, channels, and users) will have a higher impact (represented in $units^3$) and a higher priority of mitigation than attacks represented

as surfaces ($units^2$) or lines ($units$). The method has the advantage of measuring the portion of an attack left with no treatment (residual risk) or the portion affected not by attacks but deployed countermeasures (potential collateral damage). This allows one to identify priority areas in the system based on attacks with the highest attack volume or the intersection of multiple events. Obtained impact results are, however, in $units^n$, leading to the question of how to convert these values to more meaningful values such as cost or system downtime.

Musman et al. [MTT+11] showed how a set of six cyber effects (degradation, interruption, modification, interception, fabrication, unauthorized use) can represent the mission impacts of cyber incidents on military combat missions. BPMN is used to model the mission and cyber activities and supporting resources. Mission capability is estimated using stochastic simulation and Monte Carlo analysis on a mission model to link system capability to mission-oriented measures of effectiveness (MOE). Assuming that incident reports, a 4-tuple containing incident effect, affected cyber resource, start time, and anticipated duration, will be available, the information of detailed effects on IT resources can be used to modify the mission model to reflect changes in the system capabilities due to a cyberattack. The model can be rerun to produce new MOE outcomes, and the impact of the attack can be determined by comparing the two MOE values. The limitations of the method lie in the required manual intervention via a GUI to alter the mission model and the inadequacy of the BPMN modeling technique to describe the characteristics of mission-critical systems.

A more general discussion of impact on the availability of SCADA components is given in [KMKP16]. Influence diagrams [SEN09], an enhancement of Bayesian networks that use graphical representations of decision problems coupled with a probabilistic inference engine, are used to model DoS attacks on remote telemetry units of a SCADA system. Different DoS attacks on the network, transport, and application layer are simulated against the response time to identify the impact on the system availability. Impact is determined by the system unavailability, disrupted communication, improper functionality due to unreachable commands, and affected critical functionalities such as scheduling, state estimation, and islanding.

While these modeling approaches are quite different from what is used in most impact analysis studies, they enable a fresh perspective to model systems and enable a wide variety of divergent analyses that may follow.

2.5 Simulation-centric Approaches

Many works put less emphasis on the modeling aspect and emphasize the use of simulations to analyze the impact of attacks. Tatar et al. [TBG16] focused on reliability evaluation of power generation systems when under DoS or integrity attacks to compromise software vulnerabilities in RTUs or Master terminal units (MTUs) or the traffic between them. System adequacy data are collected from RBTS [L$^+$13], and Monte Carlo reliability analysis is used to calculate the mean time interval between failures and repairs. Interruptions and unreliability measurements are taken

to identify the cost of yearly interruptions for a system under the influence of cyber-attacks. The reliability indices used were the frequency of interruption, loss of load expectancy, loss of energy expectation, duration of the interruption, load curtailed per interruption, and energy index of reliability to measure the unreliability cost i.e., impact on the system availability.

Alternatively, Lemay et al. [LFK14] performed real-time integration of the actual SCADA cyber components emulated in an ICS sandbox with physical component behavior replicated by the PyPower electrical simulator. PyPower is a Python implementation of MATPOWER that has the built-in capability to calculate power flows and operation costs using optimal power flow algorithms and includes an IEEE reliability test system class with easy access to various electrical components. This integration platform allowed the estimation of the state of the power grid and the generation cost associated with that state in near-real-time, and the replication of physical terrorist attacks proposed in [SWB04] using disruptive cyber commands. The impact is calculated by tracking the physical impact of the damage leading to the increase in generation costs using simulations.

Stefanov and Liu [SL14] investigated the impact on the SCADA system and power grid conditions due to DoS, man-in-the-middle, and configuration modification attacks in the substations of an ICT network. A cyber-physical testbed is used to replicate an integrated model of a CPS where cyberattacks are conducted at the cyber system layer with appropriate security controls protecting the ICT infrastructure, whereas time-domain simulations are performed at the power system layer. The attacks led to various phenomena inside the power grid such as overload, under-voltage, and

frequency deviations and result in partial, and in case of cascading events, a complete blackout inside the power grid.

Chen et al. [CBPK13] studied the impact of modification attacks on system stability. A case study on the New England 39 bus system showed that attacks on the measurements of SVC or STATCOM system can deteriorate the stability margin or cause system instability when followed by a contingency. The system was modeled in the transient security assessment tool (TSAT) of the DSA *Tools*™ package, followed by time-domain simulations to analyze the impact using measurements of the angle and voltage stability index of the interconnected power system.

Yan et al. [YLG11] presented several attack scenarios on the power system of a wind farm SCADA system and investigated the impact of cyber attacks on power system dynamics using Digsilent Powerfactory simulations. Malicious control commands targeting the substations user interfaces have been shown to result in various physical impacts such as overspeed of the wind turbine, equipment damage, disconnection, and decrease in active power output, among others.

Many of the simulation-centric studies make use of an available testbed of a system, which, for many proprietary systems, may not be readily available. As mentioned before, testbeds are also expensive to construct and maintain [GWP19], and therefore, methodologies that employ some simulation platform are difficult to replicate or adopt to be used in other domains.

2.6 Observations

This section presents some of the key observations from the literature reviewed in this work. Table 2.1 summarizes the relevant information extracted from the studies discussed in this chapter.

Following the Department of Energy Infrastructure Assurance Outreach Program [Dag01] and its inclusions of *impact analysis of unauthorized access to cyber-infrastructure on physical system operations* in the last stage of their vulnerability assessment process, there have been many inspired studies that have looked into the impact of cyberattacks on the smart grid, associated SCADA systems, and its physical operations. An overwhelming majority of such studies use metrics that are very specific to the considered system domain to define impact. A considerable amount of work is needed to identify a generalized approach that can be leveraged to identify the impact of cyberattacks in systems in general. The model mutation approach taken by Sultan et al. [SDF17] or the Duration-Disturbance-Damage impact ranking model proposed by Li et al. [LXDW16] is a good step towards that direction.

Depending on the type of system and what we consider to be the most important properties for its mission objectives, impact may take many different definitions. Because of the partial OT properties of the discussed systems, impact on the availability of the physical system operation resulting in disrupted operation or system downtime has been regarded as the primary deciding factor by most studies. Another popular metric is the additional cost needed to repair or replace damaged physical components and/or systems. The impact on availability or any other security property may

Table 2.1: Summary of modeling and simulation-based impact analysis approaches

Category	Modeling Technique	Studied Domain	Case Study Systems	Analysis Techniques
Graph-based approaches	Graphs, Reliability Graphs, Dependency Graphs, Reachability Graphs, Bayesian Networks, Petri Nets, Graph-based Dynamical System Modeling	Smart Grid, SCADA, ICS, Manufacturing, Web-based ICS, CPS, Industrial CPS, Electrical CPS	Single generator system, IEEE 13 node test feeder system (Microgrid test example), IEEE 39-bus system, Cellular manufacturing environment, Boiling water power plant, Distributed wastewater system control application, Miniature ICS Testbed	MTTF, MTTR, MTTSD, C2P risk, Impact factor (IF), Time-dependent Impact factor (TIF), CVSS ratings, Simulations using MATLAB and Simulink, PSCAD, GNU Octave
Automata-based approaches	Finite state machines (FSM), Timed Automata	Naval system, SCADA, CPS	Rudder controller system, Simplified three-tank system	Model mutations, Impact matrices, MATLAB and Simulink simulations
Attack Trees	Attack Trees, Bayesian Attack Trees	Smart Grid, SCADA	IEEE 39-bus system with associated SCADA	CVSS ratings, MTTC, Attack efficiency, Attack Tree simulations
Other model-based approaches	Geometrical Modeling, Influence Diagrams, FDNA network synthesis, Business Process Modeling Notation (BPMN)	SCADA, ICS, Critical Infrastructures, System-of-Systems (SoS)	Military combat mission, Aerospace SoS, Littoral combat warfare SoS	Monte Carlo simulations, Functional dependency network analysis (FDNA), Geometrical measurements
Simulation-centric approaches	N/A	Smart Grid, SCADA, ICS, CPS	ICS sandbox, New England 39 bus system, IEEE 39-bus system, SCADA testbed, Wind turbine to infinite bus system	Monte Carlo Reliability Analysis, RBTS datasets, Time-domain simulations, Simulations in PyPower, Digsilent Powerfactory, TSAT

be measured as the degree of disruption or the percentage of the system operations becoming unusable or unstable due to an attack. Some studies define impact as the acceleration of natural system degradation, while others relate impact to the effect on other systems and/or components due to cascading events. Timed approaches, in particular, consider the duration of the attack and the resulting consequence on the system operations based on some simulated unit of time. A vast majority of the approaches, however, focus too much on only one objective (e.g., system availability or economic loss) and only marginally consider that sophisticated cyberattacks may violate multiple system objectives.

Graphs seem to be, by far, the most popular choice among all modeling techniques to model the cyber-physical interactions of systems. This is due to their simplicity and the intuitive way to model the control commands resulting in different state changes of the system. Petri nets and Automata-based techniques are a good alternative to this as they provide similar benefits while being more versatile than purely graph-based approaches in appropriate applications. These state-based modeling techniques, however, suffer from the state space explosion problem and see severe limitations once up against a real-world system with numerous complex interactions. To reduce the complexity in such cases, groups of salient components may be modeled as agents as suggested by Kundur et al. [KFL+10], or alternative evaluation and computation approaches may be adopted as proposed in [SDF17].

Due to how discrete control commands from the cyberinfrastructure controls the real-time operation of physical processes in ICS and CPS, we assert that modeling the timed operation of such systems is a critical aspect of performing meaningful impact

analysis. While purely graph-based techniques, Petri nets, or finite state machines do not provide ways to model the passage of time, timed variants of such modeling techniques are available, as illustrated in the works that use graph-based dynamical system modeling [KFL+10], Time Petri nets [CMM+14], and Timed Automata [SDF17], among others. These timed variants allow modeling the cyber segment of a system using discrete-event specifications and the physical segment of a system using continuous-time dynamics and allow resulting simulations to represent the system's typical and impacted operation from a real-time perspective.

The granularity that can be presented by the proposed approaches to model systems tends to differ vastly among the works considered. While many studies consider the steady and impacted state of the considered system only, other works look into the impact on individual components and even systems that are composed of other systems. Considering the impact on the state of the overall system does not represent the whole picture, as there may be impacts on system processes at the component-level that go unnoticed. The various attack scenarios presented in [LXDW16] is a prime example, as they show that even though some attacks may not force the physical system to an unsafe state, they may lead to other impactful consequences such as concentration change in a certain tank of a chemical reactor system. If the concentration is a crucial factor in subsequent operations, this may lead to disqualification of the final product, and thus, direct economic losses. We, therefore, suggest the analysis of the target system and its salient processes to determine the proper level of granularity before selecting the modeling technique or starting the impact analysis process.

All the impact analysis techniques reviewed in this work are quantitative, which is expected since simulations are a common factor among them. The majority of the works on smart grids use MATLAB and Simulink simulations for the analysis of the impact on systems and their operations. If the estimation of a metric is considered an important part of the impact analysis process, Monte Carlo simulations are widely used. Analysis approaches that use some existing form of data often make use of available datasets or use CVSS metrics to determine impact scores. For non-repairable systems, the mean-time-to-an-event, e.g., failure (MTTF), shut-down (MTTSD), and compromise (MTTC), and for repairable systems, repair (MTTR), have also been widely used. Finally, available testbeds and their integration with different simulation platforms have been used in various other studies. While the analysis approaches vary widely, the selection of such approaches is contingent on the type of system, the type of modeling technique used, and the type and number of metrics we need to determine for different impact analysis studies.

2.7 Directions for Research

The survey undertaken in this chapter led to the identification of a set of important guidelines that inspired the selection of appropriate modeling formalisms and tools for our impact analysis approach. In this section, we define our research directions based on the observations of Section 2.6 as follows:

1. While the majority of the works studied have focused on smart grid applications, attention should also be paid to *other critical infrastructure domains* in which exploitation of vulnerabilities could be just as catastrophic.

2. For more comprehensive impact analyses, we need to leverage the existing definitions to *expand the notion of impact* by including multiple system objectives.

3. We need to develop an impact analysis approach that is widely applicable for ICS in general. In that light, the development of *domain-agnostic and reusable metrics* to define impact is paramount.

4. To derive from, and be compatible with, many existing state-based studies, we identified that the use of *graph or automata-based modeling techniques* is suitable for their simplicity and versatility in modeling the cyber-physical interactions of the studied systems. Additionally, we must also prioritize identifying the *appropriate level of granularity* for a system before selecting suitable modeling and analysis techniques.

5. When selecting a suitable modeling technique, we need to select *timed variants* of such techniques so that the cyber and physical system segments, respectively, could be modeled with discrete-event and continuous-time dynamics to appropriately represent the impact on real-time physical processes.

These revelations, in turn, presented a set of five research objectives (see Section 1.4) that are the focus of this work. These challenges motivated us to develop a suitable impact analysis approach with *reusability*, *flexibility*, and *multiple security objectives* and *timeliness* of ICS operations in mind.

2.8 Conclusions

The development of a suitable impact analysis framework is an important step towards understanding the diverse impact of cyberattacks on critical infrastructures and performing comprehensive risk estimation analyses. In this chapter, we surveyed the existing works on critical infrastructures highlighting the different ways in which the impact of cyberattacks has been defined in the literature. More specifically, we looked at modeling and simulation-centric attempts at identifying and quantifying the impact on ICS and CPS. The resulting analysis enabled us to gain insight into how particular modeling techniques and critical infrastructure domains have taken precedence over the others and how impact may take diverse definitions based on the type of system and analysis approach.

At the end of this chapter, we are left with:

1. *A broad understanding of the existing modeling and simulation-based impact analysis approaches*
2. *A set of challenges leading to directions for future impact analysis studies*

Overall, the research directions identified in this chapter led to our research objectives outlined in Section 1.4, encouraging us with an ambitious goal of developing an impact analysis approach that could satisfy all of them.

Chapter 3

Approach

The impact analysis approach presented in this thesis entails performing specified activities in a four-stage process. The stages involve building timed automata models of ICS and attackers and analyzing results collected from the parallel execution of those models. In this chapter, we present an overview of the stages and introduce an example ICS that will be used as an illustrative running example for demonstration throughout the rest of this work. Section 3.1 provides an overview of the approach and the activities to be performed in each of the stages. Section 3.2 introduces a manufacturing cell control system along with its behavior specification. Section 3.3 discusses common attacks that threaten the system. Section 3.4 presents concluding remarks.

3.1 The Four-stage Process

The approach involves building a model of the target system at the early stages of the system development life cycle. The system model is then paired with an attacker model so that experimental results involving attacker actions can be used to analyze the impact on real-time system operations. In this section, we present an overview of the four-stage impact analysis process and the activities involved in each of those stages.

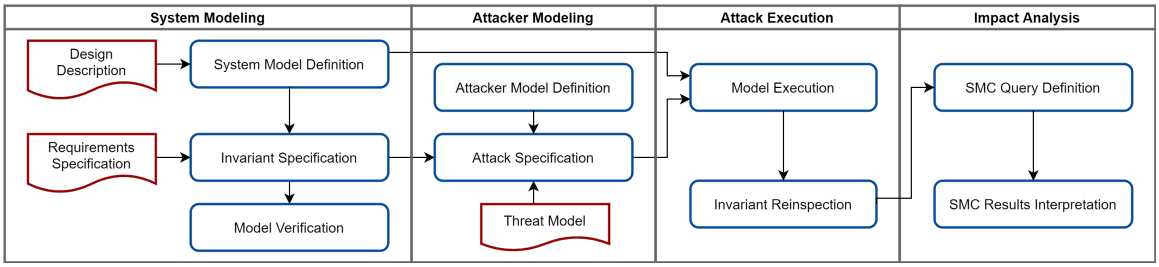


Figure 3.1: An overview of the proposed impact analysis approach

The approach is divided into four primary stages as shown in Figure 3.1.

System Modeling. The first stage involves the identification of a target system of analysis and gathering relevant documents from the early stages of the system design. The approach is applicable for systems in general that involve many cyber-physical interactions as we wish to see the impact of attacks on the cyber segment affecting the physical system processes. Next, work is required in three different steps to prepare and verify a formal system model for further analysis. These steps are named *System Model Definition*, *Invariant Specification*, and *Model Verification*, respectively.

The *System Model Definition* step focuses on building a timed automata [BDL04] model of the target system of analysis. We selected timed automata [AD94] as our

modeling formalism of choice due to its versatile nature, capability to model *timed behavior of ICS*, and ability to model systems with *any level of granularity*. Using timed automata as our basis, we leverage information from the system design description to model the intended behavior of the system components as a network of timed automata. In our work, we use the UPPAAL [BDL04] integrated tool environment that enables us to model, validate, and verify the behavior of real-time systems modeled as networks of timed automata.

Next, we consult the requirements specification documents to identify a set of system invariants and specify them using temporal logic in the *Invariant Specification* activity. These system invariants represent properties that the system must satisfy to operate in a safe manner. Lastly, in the *Model Verification* step, we use classical model checking to verify that the system model satisfies the specified system invariants and represents the expected behavior. The system model, thus built and verified formally, enable us to study the impact on the behavior of the system in the presence of an attacker. The System Modeling stage and the involved activities are discussed in detail in Chapter 4.

Attacker Modeling. The second stage focuses on developing an attacker model capable of performing various attacks on the system model. To this end, identification of the common attacks against the target system of analysis is necessary. Information from a threat model of the target system can help determine the most probable attacks against the system and formulate an attack strategy against the system. With access to a system threat model and/or information on common attacks on the system, two

different steps, *Attacker Model Definition* and *Attack Specification*, constitute this stage.

To enable the execution of a wide variety of attacks, in the *Attacker Model Definition* activity, we model the general steps that an attacker needs to exploit a vulnerability leading to an attack. These steps are presented in a generic manner and are extensible so that many different attack scenarios can be modeled. In the *Attack Specification* stage, we use system threat model information to define attack strategies determining what the attacker can and cannot do. Once determined, these attack strategies can be specified in the attacker model to enable corresponding attacking capabilities. The attacker model, with the specified attack strategies and capabilities, can now be used to exploit existing system vulnerabilities and negatively impact the system operations. The details of the Attacker Modeling stage can be found in Chapter 5.

Attack Execution. The third stage involves running the system model and the attacker model in parallel to execute attacks on the system. This parallel execution allows us to observe the system operations as the attacker keeps performing various attacks on the system. Consequently, we are able to see the impact on the system operations due to successful attacks and how these attacks affect the satisfaction of the system invariants. This stage involves two different activities, *Model Execution* and *Invariant Reinspection*.

In the *Model Execution* activity, the system model and the attacker model are executed in parallel, and simulation traces of the execution are closely monitored. Due to the timed nature of the selected modeling language, we can see the attempts made by the attacker as well as the negative impacts on the system operation due to successful

attempts. We can also reinspect the system invariants in the *Invariant Reinspection* activity to quickly identify any violation of the requirements due to the attacker's influence. Attacks that lead to a violation and consequently negatively impact the system's objectives are identified, allowing us to proceed to the final stage of the proposed approach. The Attack Execution stage is elucidated with example simulations in Chapter 6.

Impact Analysis. The final stage involves gathering statistical results from SMC queries and interpret the results to analyze the impact on the target system of analysis. For a more useful interpretation, defining the system's mission objectives and how the achievement of those objectives can be impeded, slowed, or even halted completely is necessary. This stage involves work in two different steps. The steps are named *SMC Query Definition* and *SMC Results Interpretation*, respectively.

Several SMC queries are defined and inspected to gather statistical data about the completion of system mission objectives in the *SMC Query Definition* activity. The queries chosen for inspection must relate to a metric that can be used to reason about an aspect (e.g., reachability/percentage/speed of completion, etc.) of the system objectives. The final step, *SMC Results Interpretation*, involves determining how the difference in the metrics between the normal and attacked system operations can be interpreted to define the impact on the system objectives. Thus statistical model checking results can be used to analyze the specified attacks and their severity to gain insight into the impact on the system objectives.

The approach used timed automata as the modeling language as it enables modeling the time-constrained behavior of system processes and concurrent attackers to perform a timed impact analysis. This choice also enables us to use Uppaal [BDL04] to support the modeling and verification activities of the proposed approach. More specifically, we use Uppaal 4.1.25-5 as it supports the SMC extension [DLL+15]. For a more detailed discussion of the syntax and semantics of timed automata and Uppaal notations, refer to Section 4.2.1 and Section A.1, respectively. The Impact Analysis stage along with the involved activities are detailed in Chapter 7.

A brief summary of the assumptions made throughout the approach is presented in Section 9.5.

Overall, the four-stage impact analysis approach provides a systematic framework to build system and attacker models to use such models in simulations and various model checking activities to analyze impact. Changes to the system (e.g., introducing a defensive mechanism) will require the system model to be extended in the *System Model Definition* activity, whereas activities *Model Verification*, *Model Execution*, *Invariant Reinspection*, and *SMC Results Interpretation* would have to be repeated to obtain new impact analysis results.

3.2 Illustrative Example: Manufacturing Cell Control System

To demonstrate our approach, we need a suitable model of an industrial control system. In this section, we introduce a Manufacturing Cell Control System (MCCS), adapted from [JV17b], which will be used as a running example throughout the rest of the work.

The MCCS consists of four primary system agents. The *Control Agent* C is the supervisory agent that coordinates the activities of other internal system agents. It manages the overall system phases and is responsible for starting and ending the MCCS activities depending on inputs from external systems within the production facility. The *Storage Agent* S keeps track of the empty/full status of the raw material inventory. The *Handling Agent* H moves materials from the storage to the *Processing Agent*, and the *Processing Agent* P performs the actual processing (e.g., drilling, cutting, etc.) of the material. Depending on the manufacturing facility, the type of function performed during processing can be a simple activity or a series of such actions in the chain of production. After a unit of material is manufactured, the production of the next unit begins. The MCCS continues operating until it has manufactured the required (M) units of material.

The expected operation of the MCCS is shown in Figure 3.2. The solid and dashed lines represent message-passing and shared-variable communications, respectively. Initially, the MCCS is in its idle phase, where all the system components are waiting for instructions. The manufacturing process begins when an *External System* ES

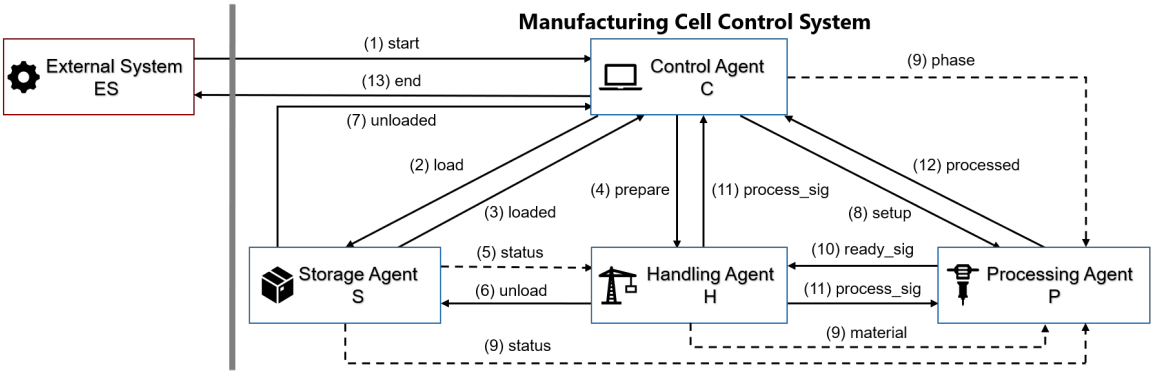


Figure 3.2: Collaboration diagram depicting the expected operation of the MCCS

sends a start message (1) to C that signifies the beginning of the manufacturing procedure. C responds by sending a load request (2) to S. S spends some time to load the material, and after the storage is full, it sends a loaded message (3) to C. C then begins the preparation phase by sending a prepare request (4) to H to prepare the material for processing.

In the preparation phase, H checks the material storage by checking the value of the shared variable status (5) for material availability. If the storage is empty, H waits until the storage is full before moving the material. When the storage is full, H sends a unload request (6) to S to unload the material storage within the specified time, which notifies C with an unloaded (7) message when the unloading is completed. C begins the initialization phase by sending a setup request (8) to P to start processing. P responds by confirming that the system is in its initialization phase, the material is available for processing, and the storage is empty by confirming the corresponding values of the shared variables phase, material, and status, respectively (9). If the conditions are not satisfied, P waits until success before moving to its setup behavior. After confirming the system's readiness to process the material, P sends a ready_sig

message (10) to H that starts moving the material to P for processing. After the moving is complete, H broadcasts a process_sig message (11) to both P and C. C then begins the processing phase for the system.

P begins the processing event by confirming the system's readiness for processing by evaluation of the value of the ready shared variable. Once the system is ready, P performs the processing according to the type of manufacturing cell. After any processing delays, P records the successful production of one unit of material and sends a processed message (12) to C. C then sends a final end message (13) to ES. The end signal prompts ES to immediately begin another production cycle. Once the MCCS successfully produces the required (M) units of material, the MCCS mission objectives are reached, and all system activities are halted, ending the manufacturing process.

3.3 Threats to the System

In an era where cybercrime thrives, we can no longer rely on the myth that ICSs are immune to cyberattacks. The high reliance on "security through obscurity" and the integration of commodity operating systems and standard network protocols has made ICS highly vulnerable to both malicious and coincidental infiltration [BL04]. The threat landscape of ICS is riddled with vulnerabilities exploitable by traditional computer viruses, malware, remote break-ins, and targeted attacks, among others [MKW+16]. Attacks may come from state-sponsored organizations as well as from competitors, malicious insiders, and even hacktivists [Micb].

Attacks on Integrity: Related to the security property of *integrity*, and in some cases, *veracity* (see Section 1.1) are tampering and spoofing attacks on the operations of an ICS. Almost every ICS involves internal communication, dependencies, and time-constrained actions analogous to the example MCCS [JV17a]. These interdependencies include the correct sequence of control messages and the proper assignment of shared variables to communicate information about individual component states. The fact that the shared-variable communications can be compromised by manipulating the external interface provided by a control system to corrupt or alter the shared variables makes data tampering attacks a common occurrence against ICS [MKW+16]. For example, for the MCCS, tampering with the shared variables `status` or `material` may misinform system agents about the availability or the readiness of the material for processing, respectively. Even worse, tampered central variables that are involved in maintaining the overall system stages, such as `phase`, may lead to all kinds of unexpected situations, and consequently, unintended operations of the MCCS. Real-world cyber incidents where tampering was used to modify the behavior of alert systems and PLCs include the Maroochy Water [SM07] and Turkey Pipeline Explosion [HF+18] (see Section 1.2).

An attacker may also intercept the messages sent and received by a compromised agent to send spoofed control commands that can force an ICS to deviate from its ideal behavior and/or result in an untimely action [CFN+18]. A more resourceful attacker (e.g., an insider) may place their own spoofed components or systematically manipulate an infected system component to perform functions that are either unexpected or undesired by the system. To illustrate, if a spoofed control message to process the material were to be sent before the material moving could finish, this

could lead to a wasted processing action for the MCCS. Furthermore, targeted spoofing attacks on one or more system agents may result in an unexpected sequence of events, leading to unpredictable outcomes. Such deviations from the normal behavior can lead to system malfunctions as well as damage to the production equipment or the final products [JJ21a].

Attacks on Availability: While attacks on *availability* may face some form of resistance due to the distributed nature of many of these systems, coordinated denial-of-service attacks can render various components of the product lifecycle to be disrupted at the same time [CRFAF17]. Attacks targeting the *timeliness* of operations, such as preventing an alarm from working when needed, may lead to explosions and equipment damage [HF+18]. Jamming attacks can interfere with the communication signals, leading to disrupted or unexpected operation of ICS components [HF+18]. For the MCCS, such attacks could prevent the system agents from recording or relaying information to the other system agents, essentially causing disruptions in system availability. The German Steel Mill incident in 2014 [HF+18] and the Ukraine Power Grid incident in 2015 have shown how the disruption of control process and/or SCADA equipment can lead to massive damage and delays in service restoration.

Attacks on Confidentiality: The first stage of an attack on ICS usually involves reconnaissance to survey and understand the target ICS environment and vulnerabilities [Micb]. ICS are plagued by malicious insiders (e.g., dissatisfied employees) who may engage in insidious espionage or even explicit modification of the system behavior [WCWW14]. Such reconnaissance may reveal information about the vulnerable components or sequence of control messages about the MCCS for example,

creating opportunities for other types of attacks (e.g., spoofing) to succeed. More sophisticated attacks on confidentiality include attacks by organizations that provide "Ransomware as a service" (e.g., *DarkSide* in the case of the recent US Colonial Pipeline attack [Rus21]), that may steal large amounts of data before asking for millions of USD in ransom to prevent information leakage.

Attacks on ICS are hard to categorize; an attack that tampers with the cyberinfrastructure (intent to attack integrity) may lead to disruptions or a halt in the physical system operations (intent to impact availability). It is, therefore, hard to tell whether compromising the integrity or the availability was the original objective of the attacker. This characteristics reiterates the need to study ICS impact since attacks to one security objective may end up impacting another, or in some cases, multiple security objectives.

In summary, attacks on ICS may vary based on their attack intent, impact intent, and the resources available to the attacker, among others. While the list of such attacks can go on, for demonstrative purposes, we will focus on the highly frequent tampering and spoofing attacks (i.e., attacks on *integrity*) on the MCCS and analyze their manifold impacts on the system mission objectives.

3.4 Conclusions

The objective of this chapter was to acquaint the readers with all the steps involved in the approach proposed in this thesis. To this end, we presented a four-stage impact analysis approach and provided a brief description of the activities involved in each

of the stages. The description included the information required to perform such activities as well as how the outcomes of each activity are used in the subsequent activities. Additionally, a Manufacturing Cell Control System, its behavior specification, and threats to the system are presented to highlight the information required to get started with the approach.

At the end of this chapter, we are left with:

1. *An overall understanding of the various activities involved in the proposed four-stage impact analysis approach*
2. *An understanding of the required information prior to starting the approach*

The MCCS will be used throughout the rest of this work as a running example to illustrate each activity in our impact analysis approach.

Chapter 4

System Modeling

The first stage in our impact analysis approach (see Chapter 3) involves building a suitable model of the target system of analysis. In this chapter, we describe the necessary activities in building such a system model leveraging information from the system design description and requirements specification documents. Section 4.1 provides an overview of the various system modeling processes. Section 4.2 introduces the selected modeling techniques and tools and illustrates the process to create a networked timed-automata model from the system design descriptions. Section 4.3 discusses the specification process of system invariants and Section 4.4 details the verification activities of the system model using the specified invariants. Lastly, Section 4.5 concludes the system modeling phase.

4.1 Overview

The System Modeling stage (see Figure 4.1) revolves around building and validating a suitable system model for the target system of analysis. First, in the *System Model Definition* activity, we model the selected system using a formal modeling technique. This activity leverages information from the system design description to identify the core system components and their primary functions. These functions are then modeled using a modeling formalism capable of representing the interactions between cyber commands and corresponding reactions in physical processes. The formal model, henceforth, will represent the intended behavior of the system under normal circumstances and provide the basis for our model-based analysis. While many such techniques may be suitable, our approach uses a timed formalism so that we can observe timed characteristics of system actions and the potential impact that cyberattacks can achieve on those actions over time.

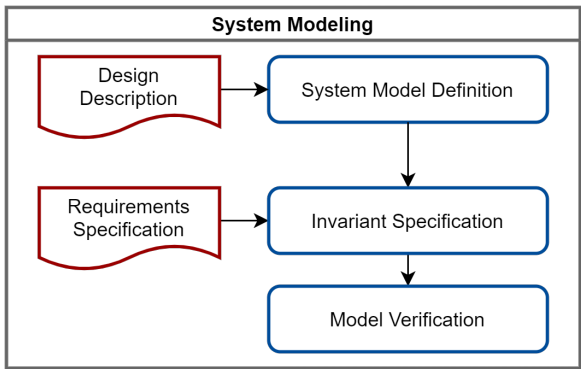


Figure 4.1: The First Stage: System Modeling

Next, in the *Invariant Specification* activity, we consult the requirements specification documents to identify a set of system invariants. These system invariants represent properties that the system must satisfy in order to operate safely. Each system

invariant must be specified using suitable logic so that we can formally verify each property. In our approach, we specify the system invariants using temporal logic, more specifically Timed Computation Tree Logic (TCTL), and verify each property separately in the *Model Verification* activity. This activity ensures that the system model satisfies the specified system invariants and represents the expected behavior of the model.

4.2 System Model Definition

Modeling an ICS in the *System Model Definition* activity entails a meticulous analysis of the system design description to identify the core system components and the various states they can be in during the system operation. Depending on the system, the number and type of states can vary considerably. The selected modeling formalism must have the capability to represent the behavior of the system in each of those states. If the system operations involve message-passing communications, we must identify the number and type of system messages required to trigger events. Additionally, for systems that require shared-variable communications, we must also identify the number of shared variables and the components responsible for keeping track of such variables. Timing of system operations is another crucial factor in many ICS where a single delayed action may endanger the entire system operation. If timing is involved, then we must also identify the timing constraints imposed upon particular actions and ensure that the selected modeling formalism supports such behavior.

To model the real-time behavior of the system process, in this thesis, we selected the timed automata [AD94] modeling formalism. As a suitable modeling environment, we use the UPPAAL [BDL04] modeling tool that provides several extensions of timed automata. A brief description of timed automata and UPPAAL is provided below.

4.2.1 Timed Automata

Timed automata is a hybrid mathematical modeling formalism in which a finite set of real-valued clocks are used to represent continuous time in a discrete-event system [AD94]. Any real-time system whose behavior involves a predetermined set of finite actions can be represented as a network of timed automata. The system can be decomposed into its constituent components, and each component can be modeled as a timed automaton, which is a finite-state machine extended with clock variables [BDL04]. Formally, a timed automaton is a tuple (L, l_0, C, A, E, I), where:

- L is a set of locations
- $l_0 \in L$ is the initial location
- C is the set of clocks
- A is a set of actions, co-actions, and the internal τ-action
- $E \subseteq L \times A \times B(C) \times 2^C \times L$ is a set of edges between locations with an action, a guard (an expression that evaluates to a Boolean), and a set of clocks to be reset
- $I : L \rightarrow B(C)$ assigns invariants to locations

A timed automaton accepts timed words; infinite sequences in which each symbol is associated with a real-valued time of occurrence [AD94]. At the simulation start, every single automaton in a network of timed automata begins with all of its clocks initialized to zero. The elapsed time is reflected in the change of clocks, representing the advance of time in reality. All clocks are independent of each other and can be reset at each transition to keep track of the elapsed time since the last reset. A transition may only be taken if the associated time constraint imposed upon that transition is satisfied by the current clock values. This feature allows us to model the time-constrained behavior of real-time systems and capture interesting qualitative and quantitative aspects such as periodicity, bounded response, and timing delays.

Modeling each component of an ICS as a timed automaton constitutes a network of timed automata, where the components can interact with each other using message-passing and shared-variable communications, representing the real-time behavior of the entire system. Furthermore, modeling other entities, such as a potential attacker, as another automaton and adding them to the networked timed automata can allow us to see the interaction between such entities and the system and the impact caused by their actions on the system operations. Therefore, the selection of a timed modeling formalism such as timed automata in our impact analysis approach is critical; it allows us to model the time-constrained behavior of ICS processes (i.e., a process must finish within the specified time) and observe the concurrent behavior of malicious entities (e.g., an attacker attacking) and beneficial entities (e.g., a defensive mechanism) to perform a timed impact analysis.

4.2.2 UPPAAL

Uppaal is an integrated tool environment that supports the modeling and verification of real-time systems as networks of timed automata [BDL04]. Models can be extended with data types (e.g., bounded integers, arrays) while also allowing model checking for verification and validation of system properties. Uppaal was jointly developed by Uppsala University in Sweden and Aalborg University in Denmark. This academic endeavor has received extensions and wide research community support for research on timed automata in recent years. Notable mentions include support for Statistical Model Checking (SMC) in Uppaal 4.1 [DLL+15], strategy space exploration in Uppaal Stratego [DJL+15], timed games in Uppaal Tiga [BCD+07], priced timed automata in Uppaal CORA [BLR04], and black-box conformance testing of timed systems in Uppaal TRON [LMNS05], among others.

Constant developments to Uppaal has led to improvements including data structures [LLPY97], partial order reduction [BJLY98], symmetry reduction [HBL+03], a distributed version [BHV00], guided and minimal cost reachability [BFH+01], work on UML statecharts [DMY02], acceleration techniques [HL02], and new data structures and memory reductions [BLP+99]. The tool is now mature, with Uppaal 4.0 being the current stable academic build[1]. To support our proposed activities, we use Uppaal 4.1 (v4.1.25-5) available under a free academic license[2].

The modeling language of Uppaal extends timed automata with additional features such as bounded model checking, templates, and urgency [BDL04]. It presents a Java

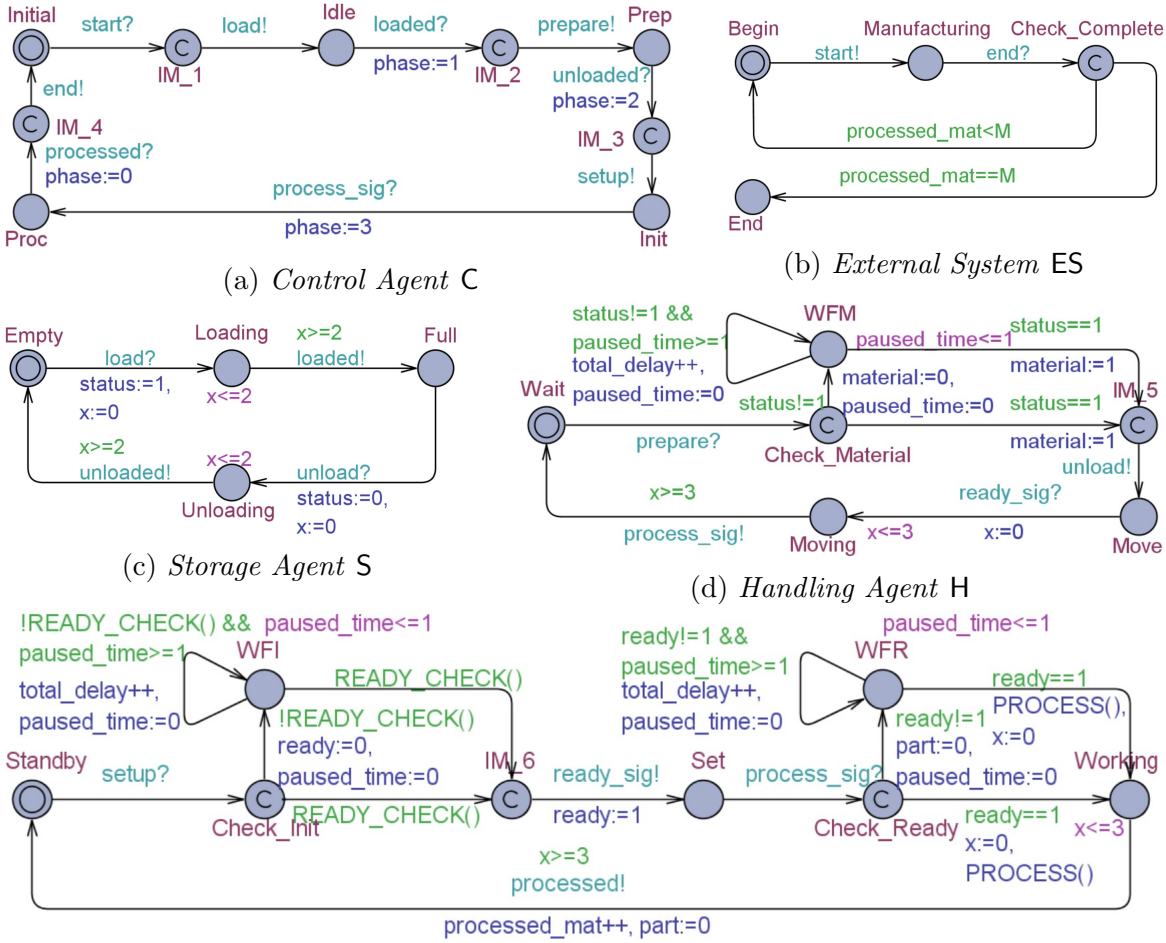
[1] https://uppaal.org/downloads/
[2] https://uppaal.org/downloads/#academic-licenses

graphical interface to facilitate the modeling of real-time systems and a verification engine written in C++ to verify system models with respect to various system specifications. At various stages of our impact analysis approach, the graphical interface will be used to model the timed behavior of various system components as well as potential attackers. Additionally, various model checking activities will be led using the verification engine to verify the system model's intended behavior, and more importantly, to identify and quantify the impact of potential attackers on system mission objectives.

A brief description of the graphical notations, expressions, and timed automata extensions used to model real-time systems in UPPAAL can be found in Appendix A.

4.2.3 The Modeling Process

To illustrate the modeling process, we refer to the system description of the MCCS in Section 3.2. The MCCS consists of four components, *Control Agent* C, *Storage Agent* S, *Handling Agent* H, and *Processing Agent* P. These components use both message-passing and shared-variable communication to communicate with each other. Therefore, we identify the messages start, load, loaded, prepare, unload, unloaded, setup, ready_sig, process_sig, processed, end, among which processed is the only broadcast signal. Additionally, we identify the state variables phase, status, material, setup, part, and processed_mat, and the components that are responsible for maintaining those variables.



(a) *Control Agent* C

(b) *External System* ES

(c) *Storage Agent* S

(d) *Handling Agent* H

(e) *Processing Agent* P

Figure 4.2: The MCCS networked timed automata

With information about the system states, messages, and shared variables, we can create a timed automaton representing each system component. When combined, these individual automata create a network of timed automata capable of interacting with each other to represent appropriate system behavior. Figure 4.2 portrays the networked timed automata model of the example MCCS modeled in UPPAAL. In addition to the core system agents mentioned in Section 3.2, we add the *External*

System ES to represent a component outside the scope of the MCCS. ES communicates with the *Control Agent* to start the manufacturing process and end it when the required M units of material have been produced.

Each individual component, now modeled as a timed automaton, consists of primary and secondary states. Primary states (e.g., Idle, Full) represent the core component behavior of a system and reflect each component performing a specific function, such as the *Control Agent* being idle or the *Storage Agent* being loaded, respectively. Secondary states may be of many different types and can represent the transitional (e.g., Loading, Moving), inspectional (e.g., Check_Material, Check_Ready), or stalled behavior (e.g., WFM, WFR) of components.

We use the example of the *Handling Agent* H to elucidate the modeling process. H has two primary behaviors, waiting and moving. These constitute the primary states, represented by Wait and Move in the timed automaton of Figure 4.2d. When a request to prepare the material for processing arrives, H needs to move the unloaded material from the storage to the processing agent. Before it can start the process, however, it first needs to confirm whether the material storage is full.

Each automaton updates specified state variables to relay its current state to other automata in the network using shared-variable communication. For example, *Storage Agent* S (Figure 4.2c) assigns values to the status variable to indicate whether the storage is empty (status = 0) or full (status = 1). Other automata in the network, once in an inspectional state (e.g., H in Check_Material after receiving the prepare? message) can confirm the status of the storage by checking the value of status to determine whether it can safely perform subsequent actions.

In ICS, faults such as clogged valves or valve leakages are a common occurrence and can often be a reason behind disrupted production [KG13]. Associated with each inspectional state for the MCCS components is a *stalled state* that represents the waiting behavior in the case of a system fault. Unnamed intermediate states (e.g., IM_1 to IM_6) are used to model instantaneous message-passing behavior after the previous action is complete.

As an example of the relation between an inspectional state and a *stalled state*, Check_Material is associated with WFM, representing the Waiting for Material behavior of H. Once H is in the Check_Material state, status should be set to 1 indicating that H can proceed to perform subsequent actions. However, if the check fails due to a fault (e.g., S is not working as intended), H stops further operations and occupies WFM until a corrective operation on S is taken. Similar stalled states, WFI (Waiting for Initialization) and WFR (Waiting for Readiness) are used in the model for P to capture the potential for faults in the checks Check_Init and Check_Ready, respectively. Additional variables are introduced in all stalled states to measure the duration of delayed operation between the identification and the correction of the fault. Understanding the significance of stalled states is critical, as we wish to see whether potential attacks on the system can force system components to behave in an unusual manner and create artificial delays in the system.

After the check in Check_Material succeeds, H moves through an intermediate state (IM_5) to its Move state. While doing so, it assigns the state variable material the value 1 to record that the material is ready to be moved and initiates the unloading process by sending an unloaded! command to S. After a certain sequence of actions, H

receives the ready_sig? request from the *Processing Agent* P (Figure 4.2e), indicating that it is now ready to begin the processing. H can now move to the transitional Moving state, move the material to P (within 3 time units) for processing, and revert to its initial Wait state after broadcasting a process_sig! message.

Transitional states represent time-dependent behavior in a model. The time constraints imposed on such states mimic real-world actions that require a certain amount of time to complete. For demonstrative purposes, the time constraints used in the MCCS model are assumed to be relatively small (e.g., three time units for moving). Depending on the system design description, the abstract time units can be interpreted as seconds, minutes, or even hours to suit individual system needs.

While the explanation provided only details the modeling procedure of the *Handling Agent*, the same approach has been used to model the rest of the networked automata. Any ICS and their individual components can be modeled as a required combination of primary and secondary transitional, inspectional, and stalled states in a similar manner.

4.3 Invariant Specification

Every ICS has its own operational mission, such as manufacturing batch materials, chemical processing, wastewater treatment, and more. In addition to developing a system model, we must ensure that the developed model satisfies the system's mission (i.e., verify the correctness of the model). To that end, information gathered during system requirements analysis can be used in the *Invariant Specification* activity to

identify properties that must hold during system operation. Once identified, these properties can be expressed using temporal logic as a set of system invariants, which are assertions that hold if the system operates in a safe state [SM17]. The satisfaction of the specified system invariants provides assurance that the system model behaves as expected.

An ICS must complete its mission without ever entering an unsafe state (i.e., it never reaches a *deadlock*), and it must do so within its time constraints (i.e., it never reaches a *time deadlock*). A *deadlock* is a system state where there are no outgoing action transitions from that state or any of its delay successors [BDL04]. Encountering a state of *deadlock*, therefore, essentially represents a case of system malfunction where all system activities have been halted. A *time deadlock* is a system state where time constraints placed on particular tasks are violated. Since these time constraints represent time needed to perform real-world actions, an instance of a *time deadlock* is equivalent to a delay in system functionality, prolonging task as well as mission completion.

An ICS must always (1) *reach its mission objectives in time*, and (2) *reach the mission objectives uninterrupted*. For example, the mission objective of the illustrative MCCS is to produce the required M units of material and then shut down (ES reaches End). Additionally, the combined time constraints on all component actions add up to impose a cumulative time constraint of ten time units to produce a complete unit of material. This constraint mandates a total production time equal to ten times the required material number (M) for the MCCS. To ensure the timely and uninterrupted operation of the system, we specify two system invariants for the MCCS

using temporal logic in the verifier in UPPAAL. The syntax and semantics of the query language used by UPPAAL can be found in Appendix A.2.

- *Invariant I*: The system shall eventually reach the mission objective (production end) in a timely manner.

$$ES.Begin \rightsquigarrow (ES.End \wedge total\_time \leq M * 10) \tag{4.1}$$

- *Invariant II*: The system shall invariantly remain uninterrupted (not reach any of the stalled states) during its mission.

$$\mathrm{A}\,\Box\ (\neg H.\mathit{WFM} \wedge \neg P.\mathit{WFI} \wedge \neg P.\mathit{WFR}) \tag{4.2}$$

Invariant I (Expression 4.1) is a time-bounded liveness property (see Appendix A.2) that checks whether the beginning of the mission (i.e., manufacturing) will always eventually lead to a timely end. *Invariant II* (Expression 4.2) is focused on ensuring the uninterrupted operation of the system at all system states.

Specifying a comprehensive set of system invariants is often an iterative and time-consuming process outside the scope of this thesis. For the MCCS, we can think of many other system invariants that follow from the requirement specification documents which may or may not be generic enough to be applicable to other ICS. Rather than coming up with an exhaustive set of system-specific invariants for the MCCS, we are interested in identifying generic system invariants related closely to a system's reliable mission completion. Therefore, the two invariants specified above, emphasizing timely and uninterrupted mission completion, respectively, will be used to verify the

system's correctness and reasonably argue that the system accomplishes its mission objectives. Additional domain or system-specific invariants may be added as needed for a more comprehensive analysis.

4.4 Model Verification

With access to a system model and the specification of a set of system invariants, we can now verify that all specified system invariants are satisfied by the system model using the model checking capabilities of Uppaal. The choice of a formal modeling technique facilitates the verification, reuse, modification, and testing of the model [Wai17]. Verification refers to the process of checking whether all simulations of a model generate the correct system behavior according to the specifications. To start experimenting with the model, we must first verify that the model behavior conforms to its specification. We do that by using model checking to exhaustively check the system model to ensure that all system invariants specified in the *Invariant Specification* activity are satisfied. Additionally, we run many concrete simulations of the system, and observe the simulated behavior of the system to confirm that the system model behaves as expected.

4.4.1 Model Checking

Model checking is an automated technique that can systematically check whether a formally specified property holds for a given state in a finite-state model [BK08]. This

is a verification technique based on an unambiguous and mathematically precise model describing the intended system behavior and systematic exploration of all possible system states in a brute-force manner. A system can be considered "correct" once all properties obtained from the system specification are satisfied through an exhaustive model check. The correctness of the model is, however, not an absolute property: it is measured relative to a certain specification and is only as good as the model itself. System verification should be done as early as possible in the system design phase, as the cost to repair defects detected early are substantially lower, and the influence of such changes on the rest of the design becomes less important.

At important points in our impact analysis approach (e.g., after the system model is built or when attackers are introduced), model checking allows us to check the correctness of the system model (i.e., answer **RQ1**.) to see whether the system objectives are reachable. The exhaustive exploration of the states enables us to confirm the intended behavior during regular operations, or when attacks on the system have no impact, i.e., attacks are ineffective. Model checking also allows us to quickly check for anomalies in the system behavior, e.g., the system not reaching its intended mission objectives due to an attack, without having to run long simulations.

Timed model checking capabilities in Uppaal, such as the time-bounded liveness property [BDL04], enables the inspection of timed mission completion and the detection of timed impact of attacks, e.g., operational delays. Model verification in Uppaal involves verifying a model created as a network of timed automata with respect to some required specifications [BDL04]. The verification engine of Uppaal, written in C++, can be used to model check one or more properties at a time. For

large verification tasks, the stand-alone command-line verifier, named *verifyta*, allows convenient verification of queries on a remote UNIX machine that has memory to spare [BDL04]. *verifyta* accepts command-line arguments for all options available in the graphical user interface, a complete list of which can be found in [BDL04].

4.4.2 Verifying the System Model

We use UPPAAL's verifier to show that Invariants I and II are satisfied for the system model. For the MCCS, the verifier shows that both invariants are satisfied by the model shown in Figure 4.2. This means that the modeled system achieves its mission objectives always with the proper behavior within the allocated time, and in a faultless manner. We also use the concrete simulator in UPPAAL to visualize timed traces of component interactions with varying missions for the MCCS to produce three, 30, or even thousands of units of material.

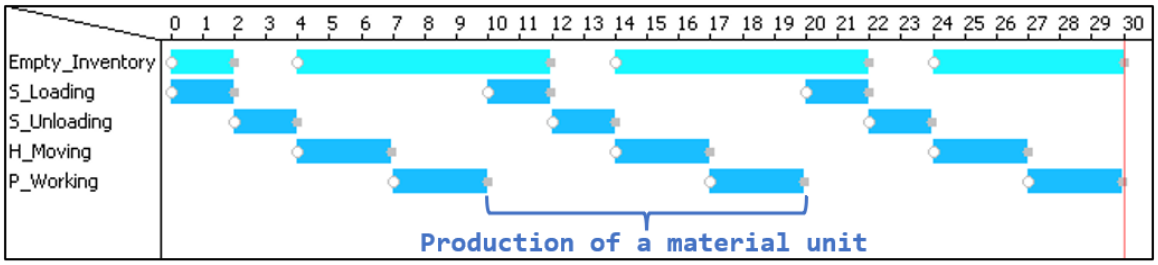


Figure 4.3: Gantt chart of the MCCS operation generated by UPPAAL

To illustrate, Figure 4.3 shows a timed simulation run of the MCCS with an objective to produce three material units. All system actions, shown in light blue, are completed within the specified time constraints (e.g., moving takes three time units). In compliance with *Invariant I*, the MCCS processes each material within ten time units and

all three materials by 30 time units. During all simulations, no stalled states were encountered (*Invariant II*). By observing that in all simulations, the MCCS shuts down after a timely and uninterrupted production, we can reasonably argue that the system model is verified to work as intended.

4.5 Conclusions

In this chapter, we discussed the various activities involved in the system modeling stage, namely *System Model Definition*, *Invariant Specification*, and *Model Verification*. The system model, built using the timed automata modeling formalism, allows us to formally define the system requirements as system invariants. Additionally, this enables us to formally verify the model using model checking. These activities are supported by information from the system design description and requirements specification documents, meaning the modeling phase can begin as early as the nascent stages of the system development life cycle. This also means that integration of our impact analysis approach to the system development process may allow it to coexist and run simultaneously with the system development and may enable significant mutations in the final system implementation depending on the analysis results.

At the end of this chapter, we are left with:

1. *A networked timed automata system model built using* Uppaal
2. *A set of domain-agnostic system invariants*
3. *Verification of the system model to ensure its intended behavior*

The activities involved in the System Modeling stage grant us access to a formal model of the target system of analysis, laying the groundwork for the rest of our impact analysis approach.

Chapter 5

Attacker Modeling

Modeling the behavior of an attacker and its capabilities are crucial steps to observe the impact that different attacker actions can achieve on a system's mission objectives. In this section, we elucidate the details of the second stage of our Impact analysis approach (refer to Chapter 3) by illustrating how to model different attackers and specific classes of malicious attacks targeting ICS. Section 5.1 presents an overview of the attacker modeling stage and the assumptions made about the capabilities of the attacker. Section 5.2 introduces the base attacker model and how the model can be extended to represent varying attacker behaviors. Section 5.3 details common attacks against ICS and the way some of those attacks can be specified in the attacker model. Section 5.4 introduces three different attacker types and demonstrates how the base attacker model can be modified to mimic their behaviors. Section 5.5 briefly discusses existing works related to attacker modeling. Finally, Section 5.6 concludes the attacker modeling stage.

5.1 Overview

An ICS is highly reliant on the correct functioning of individual system agents and their interdependencies. These interdependencies include the correct sequence of message passing as well as the proper assignment of state variables to relay the information about individual agents and the overall system state. For example, if the *Storage Agent* S were to not work as intended in the MCCS, material loading and unloading actions would be hampered. On the other hand, recording of the correct inventory state using the status variable as well as the sending of loaded! and unloaded! messages would be affected, further impacting the handling and processing of those materials.

In this thesis, we focus on cyberattacks deliberately performed to disrupt the expected flow of execution, thereby impacting the system and its operations. To do this, we disregard the existence of system faults and impacts on the system operations due to them. More specifically, our analysis will focus on intentional attacks on the system or its components to compromise the message-passing or shared-variable system communications. To this end, the second stage of our impact analysis approach (see Figure 5.1) focuses solely on attackers and how we can model their manifold behavior and capabilities.

In the *Attacker Model Definition* activity, we start by creating a generic timed automata model of an attacker. The attacker model involves the typical steps that an attacker needs to go through before performing an attack. Next, in the *Attack*

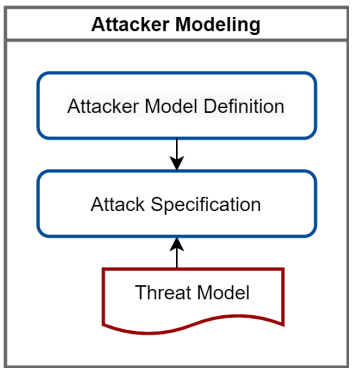


Figure 5.1: The Second Stage: Attacker Modeling

Specification activity, we specify the different capabilities of the attacker. The attacker capabilities, i.e., attacks that it can perform, should relate to the common attacks against the target system of analysis, which can be determined through information from a threat model of the system (refer to Section 3.3). In this work, we consider three different attacker types, all capable of performing various spoofing and tampering attacks on the MCCS.

It is important to understand that our goal is not to do an attack analysis to determine, for example, the probabilistic success or failure of attacks. Rather, we assume that certain attacks will succeed in exploiting existing system vulnerabilities and instead emphasize analyzing the aftermath of these exploits. More explicitly, we assume that the attacker has sufficient knowledge of the system's message-passing and shared-variable communications and may leverage higher privileged (e.g., physical or insider) access to perform attacks. These assumptions enable us to use information from the attack analysis step [CCC19, ZXW+18, MGKL19, YCG+18, FCW+05] and focus on the impact those attacks can achieve on ICS operations to comprise a comprehensive vulnerability assessment process [Dag01].

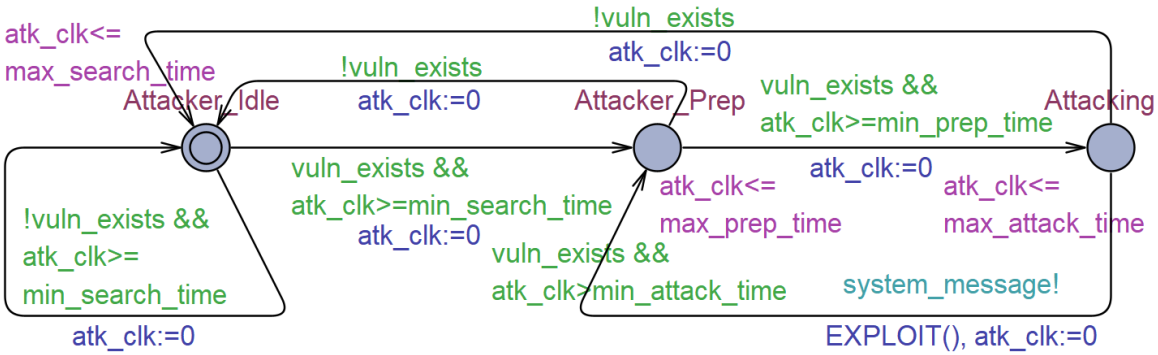


Figure 5.2: The base *Attacker Automaton* ATKR

5.2 Attacker Model Definition

To experiment with different attacks, first, we need to create a timed automaton to capture the behavior and capabilities of an attacker in the *Attacker Model Definition* activity. In this section, we describe the attacker model that, through parallel execution with the system model, will exploit the system vulnerabilities to negatively impact the system's mission objectives.

The general attacker behavior is modeled by the *Attacker Automaton* ATKR shown in Figure 5.2. The attacker, using its own clock atk_clk, can run in parallel but independently from the system. This base model includes the major steps (i.e., vulnerability searching, preparing, attacking) that an attacker needs to perform when conducting an attack. We assume that the system starts with an existing vulnerability, and the attacker knows about the existence and a way to exploit that vulnerability.

Once a system vulnerability is found, the attacker moves from its initial idle behavior (Attacker_Idle) and prepares in a *preparation* state (Attacker_Prep). The preparations may include steps such as performing reconnaissance, preparing scripts or tools, which

are included in the abstraction provided by Attacker_Prep. Once the preparations are completed, the attacker may perform a successful *exploit* from an *attacking* state (Attacking), assuming that the vulnerability still exists in the system. The particular action taken by the attacker in the Attacking state determines its capability, i.e., what attacks it can perform. The outgoing transition can be associated with a *system_message* (e.g., prepare!, or loaded!) to give the attacker the capability to spoof that message. Similarly, the transition can be associated with an *exploit* (e.g., an EXPLOIT() function) where the attack specification determines what is done once an attack is successful.

The goal of the simple attacker model is to show the interaction between the actions of the attacker and the impacted system objectives. This general modeling methodology can be extended to model not only simple scripted behaviors but also sophisticated attacks such as advanced persistent threats [ZXW+18] with multiple *preparation* and *attacking* states. By controlling the number and duration of each such state and what is done during each *exploit*, we can capture a wide range of attacker behaviors.

For example, we can extend the attacker model to involve more than one *preparation* state (each with its own time constraints) to represent a sophisticated attack where a single *attacking* action requires multiple levels of preparation. Alternatively, we can pair different *preparation* states with small but definitive *exploits* to represent more granular attacks and their corresponding effects on the system. Furthermore, associated timed conditions can be modified to control the vulnerability exploration, preparation, or attack duration for specific attacks. Different attacker automata may also be activated in parallel to observe the behavior of simultaneous attackers and/or

more varied attack scenarios. In this thesis, we focus on the attacker behaviors and capabilities revolving around single *preparation* and *attacking* states.

5.3 Attack Specification

Due to the highly interconnected components and their interdependent behaviors, ICSs are vulnerable to a wide variety of cyberattacks. In this section, we specify certain classes of cyberattacks that can leverage those vulnerabilities to compromise the system and its operations as part of the the *Attack Specification* activity.

We start by specifying attacks against the system in the attacker model. Such potential threats and attacks to a system can be identified from a system threat model that systematically analyzes a probable attacker's profile, the most likely attack vectors, and the assets most desired by an attacker [Jas20b]. Ideally, we would need to analyze the target system of analysis to build a threat model of the system and then specify the most common and/or exploitable attacks from that. Building a complete threat model of the MCCS is, however, out of the scope of this thesis since our priorities lie in analyzing the impact of attacks. Instead, we provide a brief description of common attacks against ICS in Section 3.3 and choose potential attacks against the MCCS from them to specify.

Among the many different attacks threatening ICS, data tampering and message spoofing attacks stand out due to their relative ease of execution and the capability to compromise the proper assignment of shared variables or correct sequence of system control messages used to communicate information during system processes.

Therefore, in this thesis, we choose to specify the highly likely data tampering and message spoofing attacks on the MCCS aimed at compromising its shared-variable or message-passing communications.

5.3.1 Data Tampering Attacks

A tampering attack strategy involves the corruption or modification of the the shared variables used by different system components to record their respective task state. For example, the *Control Agent* records the phase variable to keep track of the systems states, whereas the *Storage Agent* records the status variable to relay the state of the inventory to other system agents. The effects of a tampering attack would thus involve the alteration of the values of such variables so that the relayed information can be disrupted. Tampering attacks can be divided into two categories: (1) *data corruption attacks*, where the state variables of the MCCS is corrupted to unexpected and/or garbage values, and (2) *data modification attacks*, that involve the modification of such state variables to allowed but unintended values.

Data Corruption Attacks

Data corruption attacks can be specified by associating the outgoing transition from the Attacking state with a *data corruption exploit* (i.e., a CORRUPT(target_var) function) in the attacker model of Figure 5.3. The effect of data corruption attacks can be modeled by specifying the CORRUPT(target_var) function to cause an alteration of the state variables to unexpected or garbage values. An example of this would be

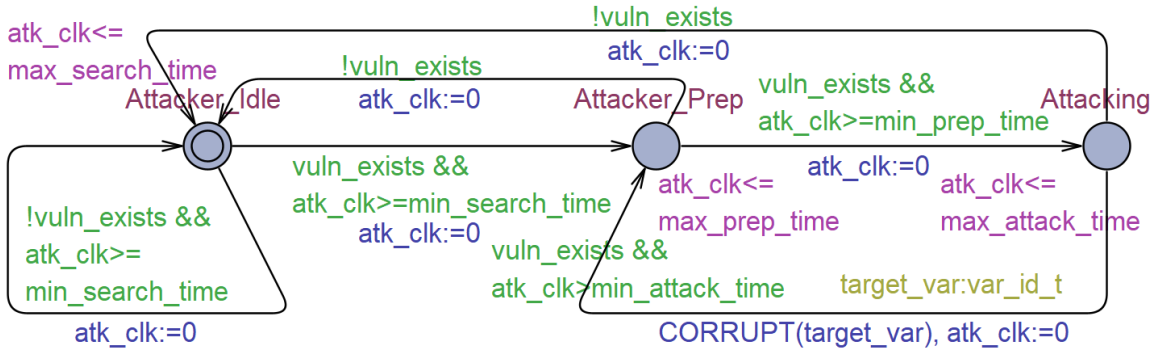


Figure 5.3: Specifying Data Corruption attacks in ATKR

assigning any value except the prescribed values of 0 and 1 (e.g., 13781) to the status variable to corrupt the information recorded by the *Storage Agent*. In this case, every time the attacker performs an attack from the Attacking state, they would target a random state variable (target_var:var_id_t) within the system and subsequently corrupt the value of that variable to compromise any actions that depend on the correct values of that variable.

The specific operations performed in the CORRUPT(target_var) function are given in Listing 5.1. All corrupted values are limited within the bound of $[0, 32767]$ due to the typical 16-bit integer type used by UPPAAL. Line 13 selects a randomly chosen double value to corrupt within the bound $[0, 32767]$, which is then rounded, turned into an integer, and assigned to the arbitrarily chosen target state variable (passed as an argument) in lines 15–24.

```
1 clock atk_clk;
2 const int total_state_var = 5;
3 typedef int[0,32767] int_t;
4 typedef int[1, total_state_var] var_id_t;
5
```

```
/**
 * Indiscriminate Data Corruption Attacks on a system
 * that modify the sensor readings so that different state
 * variables become either unreadable or unexpected
 */
void CORRUPT(var_id_t target_var){
    int_t corrupted_val_limit:=32767;   //Corruptible values are
   limited to the highest possible 16-bit non-negative integer
    int_t val_to_assign:=fint(round(random(corrupted_val_limit)));

    if(target_var==1)   //targeting the first state variable "phase"
        phase:=val_to_assign;
    else if(target_var==2)
        status:=val_to_assign;
    else if(target_var==3)
        material:=val_to_assign;
    else if(target_var==4)
        ready:=val_to_assign;
    else
        part:=val_to_assign;
}
```

Listing 5.1: Specification of Data Corruption Attacks

Data Modification Attacks

Data modification attacks, on the other hand, require more work on the attacker's part. To modify the value of the state variables within the allowed bound, the attacker

needs to know (e.g., through reconnaissance or insider access) the list of allowed values for the system's state variables. With this precondition, data modification attacks can be modeled (see Figure 5.4) by first acquiring a list of allowed values (e.g., through a GET_ALLOWED_VAL() function) before reaching the Attacking state and then specifying a *data modification exploit* (i.e., a MODIFY(target_var) function) to modify the state variables within the allowed bounds (e.g., modifying phase to any value within $[0, 3]$) in an effort to misrepresent the system states and disrupt intended system operations.

The specific operations performed in the MODIFY(target_var) function are given in Listing 5.2. Line 5 defines an array based on the number of total state variables in the system. Each element of the array contains the maximum allowed limit of a state variable identified using the GET_ALLOWED_VAL() function. Line 13 allows the attack to select random integer values within the maximum allowed range for that variable. Finally, lines 15–24 assign that random value to the arbitrarily selected target state variable (passed as an argument) in each attack.

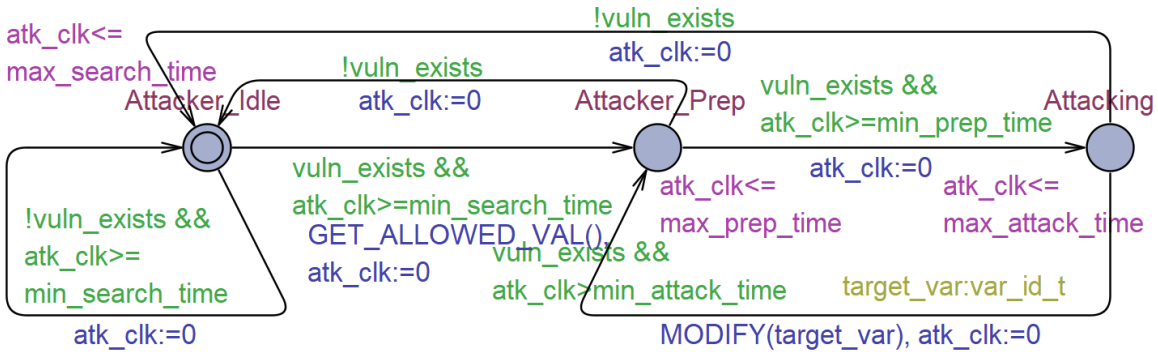


Figure 5.4: Specifying Data Modification attacks in ATKR

```
1 clock atk_clk;
2 const int total_state_var = 5;
```

```
typedef int[0,32767] int_t;
typedef int[1, total_state_var] var_id_t;
int permitted_val[total_state_var+1];

/**
 * Indiscriminate Data Modification Attacks on a system
 * that modify the sensor readings so that different state
 * variables are altered to values allowed by the system
 */
void MODIFY(var_id_t target_var){
    int_t val_to_assign:=fint(round(random(permitted_val[target_var
   ])));

    if(target_var==1)   //targeting the first state variable "phase"
        phase:=val_to_assign;
    else if(target_var==2)
        status:=val_to_assign;
    else if(target_var==3)
        material:=val_to_assign;
    else if(target_var==4)
        ready:=val_to_assign;
    else
        part:=val_to_assign;
}
```

Listing 5.2: Specification of Data Modification Attacks

5.3.2 Spoofing Attacks

A spoofing attack strategy necessitates associating the outgoing transition from the Attacking state with a spoofed system message, imitating such system commands to disrupt the intended sequence of operation. For example, the attacker may choose to spoof the loaded! message (see Figure 5.5) in order to misrepresent the completion of the loading action and prematurely activate other actions in the system. Unlike the function specifications in case of the data tampering attacks, specifying spoofing attacks involves associating the outgoing transition from Attacking with the channel synchronization for the message to be spoofed (e.g., loaded!). For demonstrative purposes, in this work, we will assume that the attacker has spoofed the *Storage Agent* S, thereby gaining the ability to send spoofed loaded! or unloaded! messages to other system components.

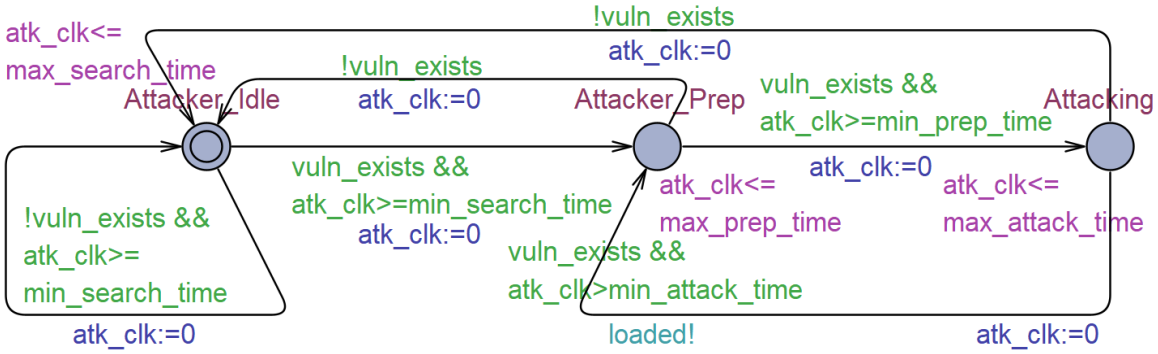


Figure 5.5: Specifying Spoofing attacks in ATKR

5.4 Attacker Types

The threat landscape of ICS is plagued by many different kinds of attackers, starting from script kiddies to disgruntled insiders to government agencies. These attackers differ based on their attacking tendencies as well as their resourcefulness in performing each attack. Therefore, along with specifying different kinds of attacks, we also model the behavior of different attacker types to get a broader view of the various attacker capabilities. In this section, we describe three different attacker types and how we can model their behavior using the *Attacker Automaton* ATKR of Section 5.2.

5.4.1 Random Attackers

We start by modeling *Random* attackers, crafty adversaries that perform arbitrary attacks on the system. These attackers are concerned about detection but lack the necessary information about the system to perform precisely timed attacks. After a successful exploit, each subsequent attack is performed at random intervals to avoid detection as much as possible. Some of these attackers might have minimal information about the system and/or its defensive capabilities so that they can time their attacks to avoid having their pattern identified. An example of these kinds of attackers is inexperienced attackers trying their hand at some tools and performing attacks at different intervals, e.g., after learning new things or making some modifications to their original attack. Other examples include malware in their pre-programmed or dynamically determined hibernation period [JdLAF18] or experienced attackers

Table 5.1: Timing Constraints for Different Attacker Types

Attacker Type	min_search_time	max_search_time	min_prep_time	max_prep_time	min_attack_time	max_attack_time
Random Attacker	1	1	1	5	1	2
Relentless Attacker	1	1	N/A	N/A	1	2
Informed Attacker	1	1	N/A	N/A	1	2

who deliberately attack at random intervals to increase their chances of remaining undetected.

The three different attacker models presented in Figures 5.3, 5.4, and 5.5 for data corruption, data modification, and spoofing attacks, respectively, represent such a *random* attacker. To have *random* attacking capabilities, there must be a positive difference between the time constraints imposed upon each action state (*preparation* or *attacking*) and the outgoing transitions from such states. As presented in Table 5.1, the timing constraints imposed upon *random* attackers in our work is such that vulnerability searching takes a fixed duration of one time unit (we assume the existence of a vulnerability), preparing for each attack takes anywhere from one to five time units, and each attack takes anywhere from one to two time units. The randomly selected times for the latter two cases are uniformly chosen, as mandated by bounded time constraints in Uppaal [BDL04].

5.4.2 Relentless Attackers

Relentless attackers are single-minded entities that care less about being detected and more about being successful in their exploit. Once these attackers find a vulnerability, they immediately start their actions as fast as possible, with no intervals between subsequent attacks. Their attacks are similar to a constant barrage of a simple scripted

function that exploits a vulnerable side of the system in question. A typical example of this would be bots in a botnet that either scan the network for vulnerable devices or perform constant directed attacks on a target website, commanded by a botmaster [AAB+17]. By nature, we expect these attacks to have a very high impact on defenseless systems. Conversely, in a system with an active detection mechanism, we surmise these attacks to be the easiest ones to detect and take preventative measures.

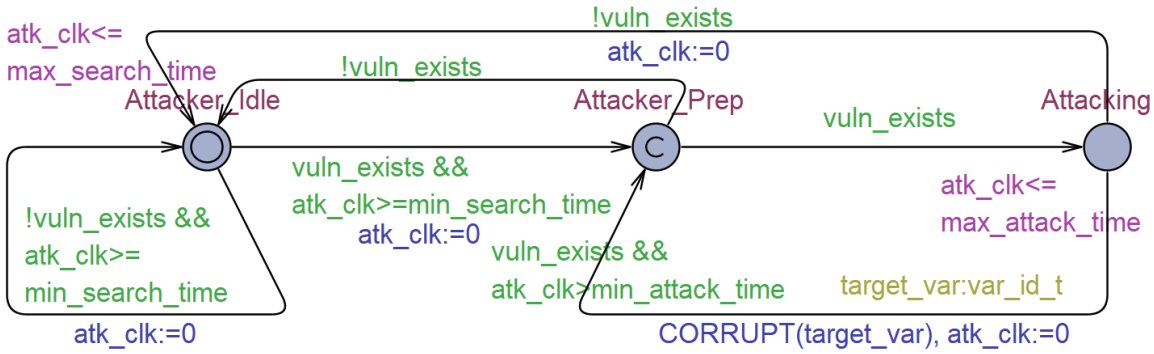


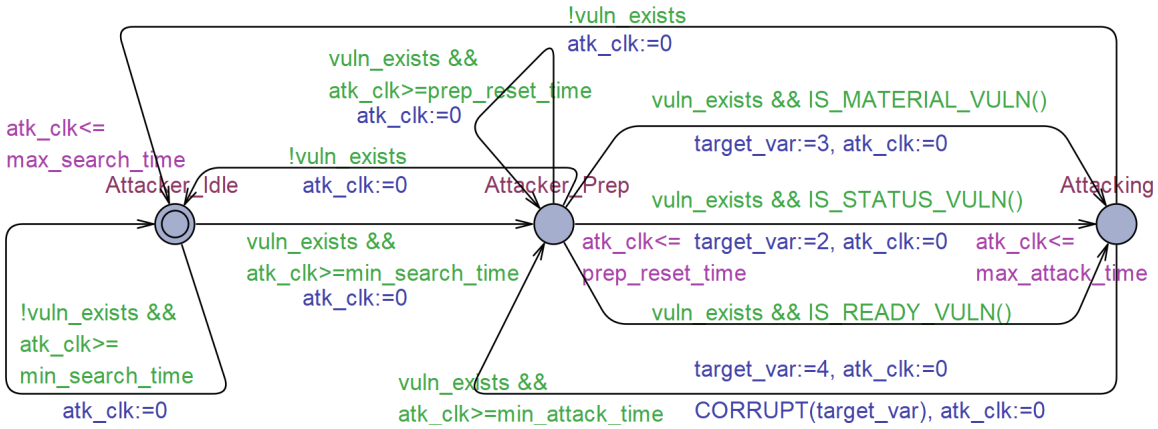
Figure 5.6: The *Relentless Attacker Automaton* for Data Corruption Attacks

Figure 5.6 shows how to modify the base attacker model of Figure 5.2 to capture *relentless* attacker capabilities for data corruption attacks. All the *preparation* states must be turned into *committed* and/or *urgent* states, and the time constraints on such states and all outgoing transitions must be removed. Upon discovering a vulnerability, this will enable the *relentless* attacker to perform incessant *attacking* actions without a need to spend any time preparing. Identical changes can be made for data modification and spoofing attacks to implement the same *relentless* attacking capabilities. The timing constraints imposed upon *relentless* attackers in our simulations are very similar to *random* attackers (see Table 5.1). However, *relentless* attackers do not need any time to prepare between subsequent attacks, and as such, do not contain the parameters `min_prep_time` and `max_prep_time`.

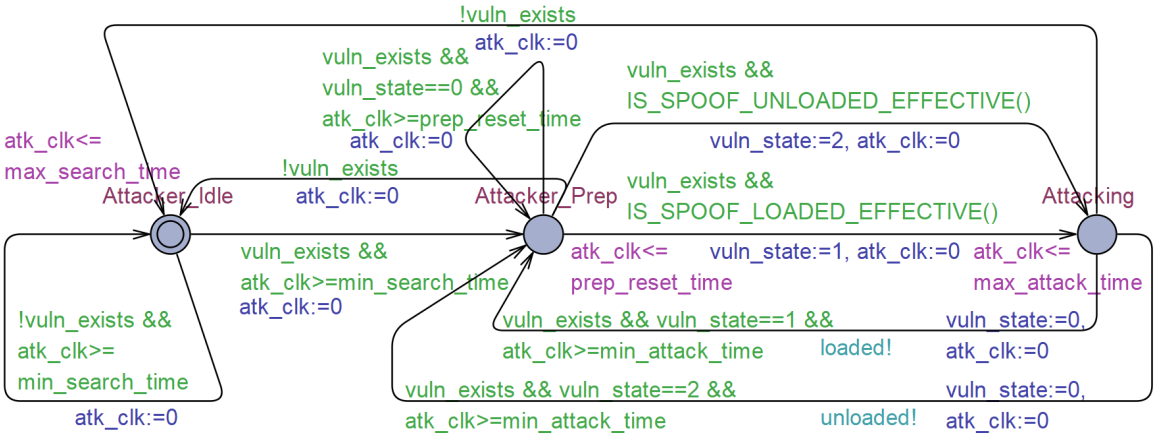
5.4.3 Informed Attackers

Informed attackers are the highly intelligent adversaries that have insider information about the system and its defensive capabilities. They perform targeted attacks to exploit the system vulnerabilities at the exact moments when the system is most vulnerable. These attackers are extremely potent at compromising systems where system functions take a predetermined constant amount of time (e.g., moving always taking three time units) and may decide to perform an exploit only when the system is vulnerable. Examples of these types of attackers are insiders with malicious intentions (e.g., disgruntled employees, ex-employees working for competitors) or an attacker with access to such insider information (e.g., through social engineering, bribing, threatening, or blackmailing) that can perform timed and targeted attacks on the system.

Modeling *informed* attackers requires precise knowledge of the target system operations and modifying the activities in the *preparation* states accordingly. This way, the attacker can prepare for exact moments of vulnerability, and each attack can be performed when the system is known to be most vulnerable to create a definitive impact on the system. The goal of modeling the *informed* attacker is to assume that a more resourceful adversary will be able to gather more information about the system to see how much impact such advanced attackers can create on the system. Since the proposed impact analysis approach is meant to be done at the early stages of the system development lifecycle, the system analysts performing the modeling would have access to such system information. By modeling *informed* attackers with



(a) *Informed Attacker Automaton* for Data Corruption Attacks



(b) *Informed Attacker Automaton* for Spoofing Attacks

Figure 5.7: *Informed Attacker Automaton*

varying capabilities, they would be able to better understand the different possible impacts on the system.

Figure 5.7a shows how transitions from the Attacker_Prep state has been modified from the base attacker model of Figure 5.2 to add the capability of performing timed data corruptions for *informed* attackers. The *informed* attacker no longer moves to the Attacking state from the Attacker_Prep state after a certain time period. Instead, the three state variables status, material, and ready are constantly monitored in the

Attacker_Prep state to identify the exact times when a corruption of these variables would create a definitive impact on the system. Once such an opportune moment is found, a *timed data corruption exploit* (CORRUPT(target_var)) is performed from the Attacking state.

Each of the three variables mentioned is attacked after they have been assigned by the respective system component and before being read by another system component. Variable phase is not considered since there is no delay between its assignment and subsequent checking, and therefore, no vulnerable moment when an attacker can attack. Variable part is not considered as it is never read by any system components, and thus, its corruption does not affect the system operations. While *random* and *relentless* attackers would still attack variables phase and part due to the indiscriminate nature of their attacks and lack of information, *informed* attackers can choose their targets carefully to create definitive impacts on the system.

Data modification attacks by an *informed* attacker can be modeled in the same way as data corruption attacks, except with the acquisition of allowed values (GET_ALLOWED_VAL()) and replacing the CORRUPT(target_var) function with the MODIFY(target_var) function. Spoofing attacks (see Figure 5.7b), on the other hand, can be modeled by identifying the time frame when the spoofing will inevitably cause disruptions in the system operations and performing selected spoofing attacks (e.g., loaded! or unloaded!) as necessary. In our simulations, the timing constraints imposed upon all *informed* attackers are similar to *random* attackers (see Table 5.1) except for the fact that these models do not have fixed minimum or maximum times for preparation. These attackers may prepare for any duration of time before finding the

perfect opportunity to perform an attack. It is important to note that these models have an additional parameter (`prep_reset_time`) to provide a bounded time delay for the Attacker_Prep state to allow the model to avoid getting stuck when none of the outgoing transitions can be taken, e.g., due to a *deadlock*.

The various timing parameters selected for demonstration in the different attacker models (see Table 5.1) are small enough to allow the attacker to attack during each material production cycle and in between system processes for maximum interaction with the MCCS. A sensitivity analysis of the timing parameters, analyzing how changes to such parameters affect our impact analysis results, are presented in Section 9.7.2.

5.5 Related Work on Attacker Modeling

In this section, we discuss studies that attempt to categorize and/or define various attacker models targeting systems involving various IT and OT components such as ICS and CPS.

Rocchetto and Tippenhauer [RT16] present a common terminology for attacker models targeting the CPS domain. The work presents a classification of attacker profiles from surveying existing works on attacker modeling that targets CPS and highlights how a real-life attacker may not fall under one specific profile. Studies on attacker modeling are categorized based on ten different features, including attack dimensions, the number of action types available to attackers, the generality of the model, and the use of time in the model. Finally, the authors propose a formalized attacker

profile and manually define six distinct attacker profiles encompassing most informal attacker models in existing works.

In addition to the attacker profiles described in [RT16], several other studies attempt to provide formal or informal definitions of attacker profiles. Cardenas et al. [CAS+09] highlighted the key challenges in securing CPS and informally defined four attacker profiles (called adversary models) specific to CPS. In [CB06], the authors proposed a taxonomy of sensor network security. Their attacker model (called a threat model) identified the differences between insider and outsider attacks and discussed several dimensions such as skills, costs, and actions related to the attackers. The authors of [PSSS04] categorized attackers (called cyber adversaries) based on dimensions such as skills, knowledge, time, and resources available. The study also focused on rating attacks and adversary profiles based on descriptions of concrete metrics.

Several works in the literature focus on specific attacks or different classes of cyber-attacks targeting CPS such as tampering. The work of Lin et al. [LYY+12] presented formal models of some general attacks on the energy grid based on false data injection to manipulate the quantity of energy supply, energy response, or the energy transmission link state. The attacks on distributed routing showed the disruption of the energy distribution process through attack simulations. The attacker model (called a threat model in this work) is assumed to have the knowledge and the capability to attack the system and can tamper with data or components to compromise them to inject malicious code. Liu et al. [LNR11] presented novel false data injection attacks on electric power grids where attackers could introduce malicious measurement errors into certain state variables that were either tolerated or not detected by existing state estimation

algorithms. The authors extended the attacks to generalized false data injection attacks and tested the attacks against two ad-hoc example systems. Krotofil et al. highlighted modeling the timing of DoS attacks to achieve maximum impact on CPS. Specific to the electric power grid, the authors defined an attacker model capable of performing DoS and false data injection attacks and demonstrated their attacks on a modified model of a real-world plant-wide industrial process. To motivate the study of cyberattacks on complex water systems, Taormina et al. [SO16] modeled attacks that could manipulate the tank water level or control strategies of automated pumps in a water distribution system. They discussed limitations of the EPANET [Ros00] framework (a numerical modeling environment) and how assumptions on the knowledge of physical and virtual attacks, respectively, could lead to modeling direct and indirect actions informally defined in their attacker model. In [TGD+19], the authors introduced epanetCPA, an open-source MATLAB toolbox, and performed single simulation analyses on various attacks, including alteration of PLC and SCADA control statements on a water distribution system. Urbina et al. [UGTC16] showed how a Man-in-the-Middle attacker could launch a range of attacks on a room-sized water treatment testbed to manipulate or replace sensor data leading to incorrect control decisions. The primary characteristics of their attacker model are separated by their objective and resources and can remain hidden from typical bad-data detection mechanisms. Esfahani et al. [EVM+10] performed a safety analysis on the automatic generation control loop of a two-are power system using a reachability framework. The study is focused on the mathematical modeling of the system, aiming to perform a risk analysis by identifying possible policies that an attacker may adopt to disrupt the system operations. Their attacker model assumes access to all system states.

A number of studies involve formal representations of attacker models and/or a system model of the target system of analysis. In [AM16], the authors presented attacker models along with a system model of a CPS. Demonstrated on a water treatment system, the study defined attacker dimensions such as the target system and objective of an attack, including the target property violation (e.g., integrity) and attack performance (i.e., impact). Basin et al. [BCSS11] presented a formal approach to model and verify security protocols involving physical-layer properties and apply their proposed approach to four case studies. Several dimensions related to physical properties, e.g., time, agent locations, physical properties of the communication network, were defined when modeling a CPS, whereas attackers (called intruders) were represented as a set of nodes in the formal CPS model. In [FKL+13, LFK+11], the authors defined introduced attack execution graphs, representing potential attacks steps against the system with a set of abstract components. The study involves the formal definition of a six-dimensional attacker, which has been incorporated in the ADVISE framework, allowing users to define their own models. The authors of [Vig12] presented a formal definition of an attacker model and a system model of a CPS. The attacker model is defined as a set of pairs representing locations in the network topology and attacker capabilities, expressed as a set of tuples representing attacker actions, cost, and range. The study devised a framework to compare between attacks exploiting the physical and cyber weaknesses of CPSs. Teixeira et al. [TPSJ12] presented the model of an attack space involving the primary dimensions (knowledge, disclosure, and disruption resources available to the attacker) and several subdimensions. The proposed attacker model is generic and applicable to networked control systems. The authors discussed various attack scenarios representing replay, zero dynamics, and bias injection attacks

(especially stealthy variants) on a quadruple-tank process controlled over a wireless network.

In contrast, our work focuses on modeling the generic behavior of attackers, e.g., by leveraging existing information on attacker profiles identified by many such works. For example, the *random* and *relentless* attacker types discussed in our work may fall within the "Basic User" attacker profile defined in [RT16]. Similarly, the behavior of *informed* attackers may range anywhere from "Insiders" to "Cybercriminals" to "Nation-State" attacker profiles. Our work intends to leverage the existing knowledge to model various attacker behaviors and see how different attack capabilities (specified in the *Attack Specification* activity from system threat model information) pair with such attacker behaviors. The ultimate goal is to use such attacker models to characterize the impact, and in turn, identify the most impactful attacks, attackers, or a combination thereof.

5.6 Conclusions

To observe the impact caused by different attacks on the system, the development of an attacker model with varying attacking capabilities is a necessity. In this section, we presented a generic attacker model that can be extended as needed to capture a wide variety of attacker behaviors. We demonstrated how information about common attacks against a system could be used in specifying different data tampering and spoofing attack strategies in the attacker model. Finally, we discussed how modifications to the base attacker model enable us to capture the behavior of three different

(*relentless*, *random*, and *informed*) attacker types, differing in their goals, resourcefulness, and inclinations.

At the end of this chapter, we are left with:

1. *Three types of attackers (random, relentless, and informed), each modeled as a timed automaton*
2. *Two distinct attack strategies for the attacker models:*
   (a) *Data tampering attacks (Data corruption and modification)*
   (b) *Message spoofing attacks*

These attacker models, coupled with their varying data tampering and spoofing capabilities, will be used throughout the remainder of the thesis to execute attacks against the system, leading to the observation and quantification of the impact.

Chapter 6

Attack Execution

Following the definition of the system model and the attacker model with various attacking capabilities, in the third stage of our impact analysis approach (see Chapter 3), we move towards executing attacks on the system model to observe how the attacker's existence affects the achievement of the system's mission objectives. Section 6.1 provides a broad overview of the Attack Execution stage. Section 6.2 and Section 6.3, respectively, illustrate how the parallel execution of the system model and the attacker model allows us to simulate and observe the impact of data corruption and modification attacks, whereas Section 6.4 demonstrates the same for message spoofing attacks. Section 6.5 revisits the system invariants to check for invariant violations and shows how such violations aid in understanding the existence and the impact of an attack. Lastly, Section 6.6 concludes the Attack Execution stage.

6.1 Overview

The third stage of our impact analysis approach (see Figure 6.1) will focus on executing attacks on the MCCS to capture different attacker behaviors and identify the potential impact on the system's mission objectives (i.e., answer **RQ1**.). The execution of attacks requires running simulations of the system model and the attacker model in parallel in UPPAAL. The simulations assume that the system has an exploitable vulnerability, and the attacker can leverage it to execute specified attacks on the system. More specifically, we assume that the attacker can exploit the existing system vulnerability to tamper with shared variable communications or send spoofed control commands to the system components.

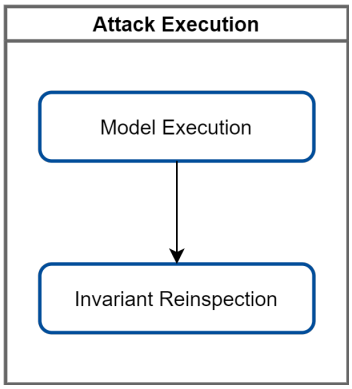


Figure 6.1: The Third Stage: Attack Execution

First, in the *Model Execution* activity, we execute the system model and the attacker model in parallel and study the simulation traces to observe the attempts made by the attacker and the negative impacts on the system operations due to successful attempts. All data tampering and spoofing attacks are executed with three different attacker types, i.e., the *random*, *relentless*, and *informed* attackers as defined in Section 5.4. All simulations involving the attackers follow the timing constraints

specified in Table 5.1. In general, all attackers are assumed to take a fixed duration of one time unit to do a vulnerability search (vulnerabilities are assumed to exist) and take anywhere between one to two time units to perform each attack. While *random* attackers take anywhere between one to five time units between each attack, *relentless* attackers require no preparation between subsequent attacks, and *informed* attackers may prepare for any period before finding a suitable opportunity to perform their attacks. All the timing constraints, along with the capabilities of the attackers, are easily modifiable. However, for our simulations, we will focus on the timing constraints of Table 5.1 and attacker capabilities specified in Section 5.3.

After running the simulations, we can also reinspect the system invariants discussed in Section 4.3 in the *Invariant Reinspection* activity to see if any of them are violated. Since we do not consider the occurrence of system faults, a violation confirms the influence of an attacker on the system operations. In the case of an identified violation, we need to determine what that property violation means in the context of the system's objectives. This can allow us to see disruptions in the system operations, such as malfunctioning system components or undesired delays. The interpretations enable us to move towards the Impact Analysis stage and gather statistical data using SMC to analyze the impact on the system mission objectives.

6.2 Executing Data Corruption Attacks

We start by modeling data corruption attacks that endanger the shared-variable communications within the MCCS. More specifically, we focus on attacks that can corrupt

random state variables to disrupt the communications that rely on the correct values of such variables. We assume that the attacker possesses the capability to tamper with MCCS system components directly or through an external interface, misinforming the other system agents. The simulations below show how different data corruption attacks (specified in Section 5.3) by *random*, *relentless*, and *informed* attackers negatively impact the MCCS mission objectives.

6.2.1 Data Corruption by a *Random* Attacker

First, we execute the system model in parallel with a *random* attacker (refer to Section 5.4.1) corrupting the shared variable communications of the MCCS at irregular time intervals. An example simulation run in UPPAAL depicting the impact of data corruption attacks is shown in Figure 6.2.

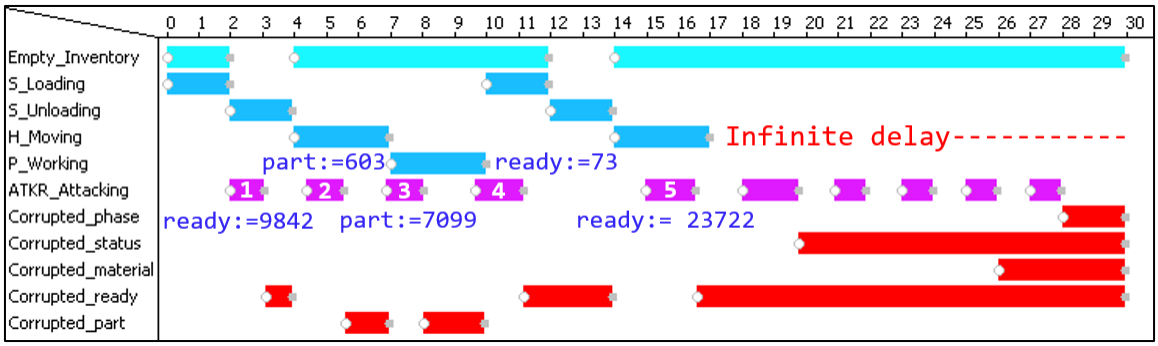


Figure 6.2: Infinitely delayed production due to data corruption by a *random* attacker

The times when the *random* attacker is active, i.e., attack attempts made by the attacker, are denoted by ATKR_Attacking (shown in violet). Attempt 1 of the attacker succeeded at 3.11 time units, corrupting the value of the ready variable (shown in red) to 9842 that should ideally represent 0 to represent the unpreparedness of the

system for processing. The system, however, reset the value of the ready variable as part of its regular operation and the attack failed to affect the system operations. Attempts 2 through 4 all succeeded in corrupting different state variables with arbitrary garbage values, yet they failed to affect the process due to attacking at random inopportune moments. Attempt 5, however, corrupted the value of ready during the moving process. In this particular case, there is no way to reset the corrupted variable before it is inspected at the end of the moving process. Following a corrupted read, the system entered the undesirable stalled state WFI, where it had to wait for the correct system parameters to progress. We stopped the simulation after exceeding 30 time units (time to manufacture three units of material) as we knew that the system had reached a *time deadlock*.

A *time deadlock* is a system state where time constraints placed on particular tasks are violated. Since these time constraints represent the time needed to perform real-world actions, an instance of a *time deadlock* is equivalent to a delay in system functionality, prolonging task and mission completion. With no way to restore the corrupted data, after a successful compromise, a system without any security controls (e.g., a data recovery mechanism) can no longer continue manufacturing. Thus, only one unit is produced in the end, and the production of the second unit is infinitely delayed. Further simulations show that successful corruption of state variables immediately halts the system's progress whenever the system tries to process such information.

6.2.2 Data Corruption by a *Relentless* Attacker

Next, we execute the system model with a *relentless* attacker (refer to Section 5.4.2) capable of performing incessant data corruption attacks on the shared variable communications of the MCCS. An example simulation run in UPPAAL depicting the impact of such relentless corruption attempts is shown in Figure 6.3.

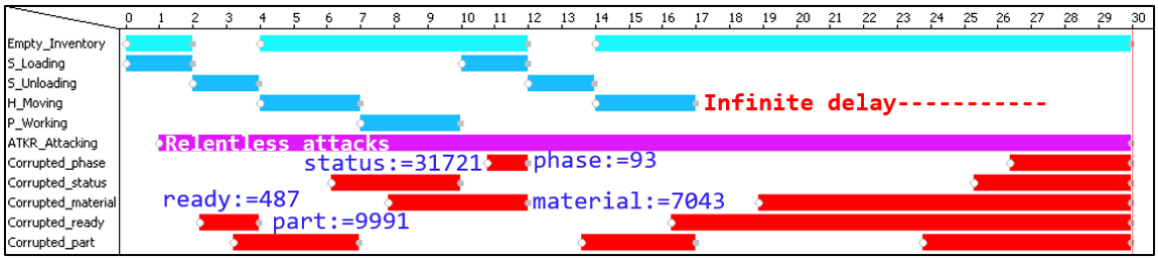


Figure 6.3: Infinitely delayed production due to data corruption by a *relentless* attacker

The simulation shows the significant impact of the *relentless* attacker corruption attempts in a system without a defensive mechanism. After the vulnerability identification, the *relentless* attacker started an incessant barrage of attacks on the system, resulting in all five state variables being corrupted within the first 11 time units. The MCCS, however, was able to remediate some of the corruption attempts due to its regular system operations, and the first material unit was produced without any disruptions. Regardless, the relentless corruptions succeeded in creating a *time deadlock*, and as such, an infinite delay at approximately 17 time units, stopping the MCCS from producing any further materials. Compared to corruptions by the *random* attacker, corruptions by the *relentless* attackers are much more aggressive and result in more frequent corruptions.

6.2.3 Data Corruption by an *Informed* Attacker

Finally, we execute the system model in parallel with an *informed* attacker (refer to Section 5.4.3) capable of performing targeted and timed data corruption attacks on the MCCS. An example simulation run in UPPAAL depicting the impact of such timed attempts is shown in Figure 6.4.

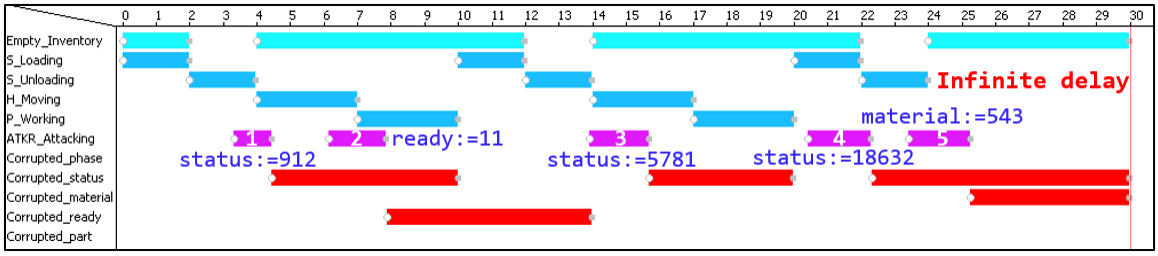


Figure 6.4: Infinitely delayed production due to data corruption by an *informed* attacker

We can see the behavioral tendencies of the *informed* attacker in the various corruption attempts made in the simulation. All attacks on the state variables are made only when that variable has already been assigned by the system and would be read sometime in the future by another system agent. For example, attacks 1, 3, and 4 show how corruption attempts on the `status` variable are made only during the loading and unloading actions of the *Storage Agent*. The attackers, however, are not omnipotent as their attacks still require some time to complete. By that time, the reading of the variables could already be done, and before the corrupted variable is read again, it could be reassigned the correct values by a typical system operation. Such is the case for attacks 1, 2, and 3, where they missed the first window of opportunity to cause a compromise, and the corruptions were remediated later by the typical system operations. The attack that finally causes a *time deadlock* on the system is attack 4,

disrupting the moving action with a corrupted read of status, thus causing an infinite delay on the system.

Compared to the *random* and *relentless* attackers, the corruption attempts made by the *informed* attacker are slow and measured. The *informed* attacker also avoids performing unnecessary attacks. This behavior can been seen by the attacker no longer making a data corruption attempt after attempt 5 since it is already aware of the system compromise. In our simulations, we assume each attack to take some time between one and two time units. If each attack of the *informed* attacker were to be faster, all of them would result in a successful compromise of the system and its operations.

6.3 Executing Data Modification Attacks

Next, we turn our attention towards executing data modification attacks that also endanger the shared-variable communications of the MCCS. Contrary to the arbitrary corruption, modification of data is intended to misrepresent the state of the system components by modifying the value of state variables within the specified limits. However, this also requires more work on the attacker's part as they would need to identify the allowed values for the system state variables before starting their attacks. We assume that the attacker possesses such information for the state variables of the MCCS and can modify them directly or through an external interface.

The simulations below show how different data modification attacks (specified in Section 5.3) by *random*, *relentless*, and *informed* attackers negatively impact the MCCS mission objectives.

6.3.1 Data Modification by a *Random* Attacker

First, we execute the system model in parallel with a *random* attacker that can modify the shared variable communications of the MCCS at irregular time intervals. An example simulation run in UPPAAL depicting the impact of data modification attacks is shown in Figure 6.5.

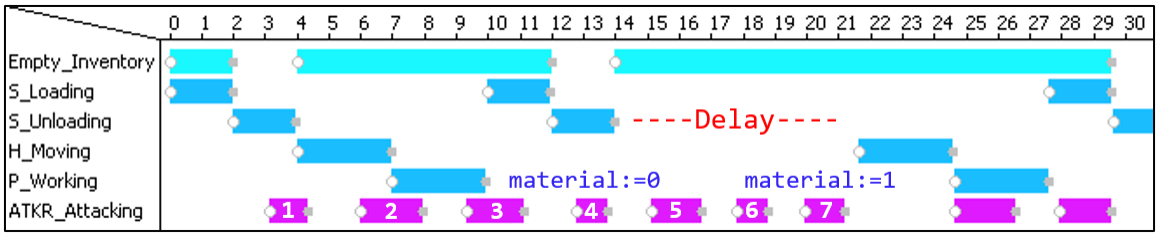


Figure 6.5: Delayed production due to data modification by a *random* attacker

Since the *random* attacker performs arbitrary attacks on the system, they may easily result in modified values that are the intended values for a particular variable at that moment in time. Such is the case for the first three attempts, where the modifications resulted in intended values and no impacts on the system. Attempt 4 succeeded in a compromise by misrepresenting the value of material to show that the material is not ready for moving (even though it had been properly unloaded). This forced the system to move to WFI again. Due to the randomness in the attacker's behavior, it may also end up correcting its own attack. This behavior can be seen for attempt 7, which remodified the material variable, allowing the system operations to continue

with the moving action. While data modification attacks by *random* attackers can create *time deadlocks* similar to data corruption attacks, the impact is milder, as the delays are not infinite.

6.3.2 Data Modification by a *Relentless* Attacker

Next, we execute the system model with a *relentless* attacker with the capability to modify the shared variable communications of the MCCS through relentless modification attempts. An example simulation run in UPPAAL depicting the impact of such incessant data modification attempts is shown in Figure 6.6.

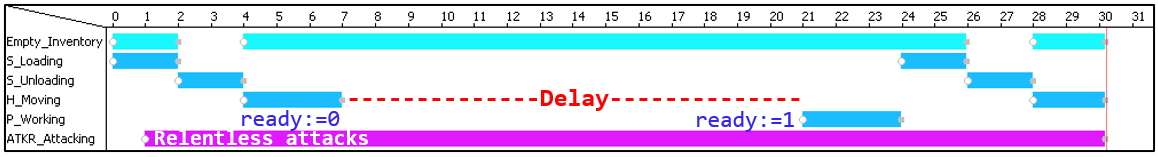


Figure 6.6: Delayed production due to data modification by a *relentless* attacker

The continuous modification attempts made by the *relentless* attacker quickly resulted in a compromise at approximately 7 time units. Due to the modified value of the ready variable, the compromise led to a delayed processing operation. The attacker, however, kept attacking and modifying different system variables, resulting in a remodification of the ready variable and a restart of the delayed processing action. Similar to the *random* attackers, *relentless* attackers performing data modification attacks are only able to create temporary delays and may correct modifications made by their own attacks due to the randomness involved in the modification process.

6.3.3 Data Modification by an *Informed* Attacker

Lastly, we execute the system model in parallel with an *informed* attacker capable of performing targeted and timed data modification attacks on the MCCS. An example simulation run in UPPAAL depicting the impact of such timed data modification attacks is shown in Figure 6.7.

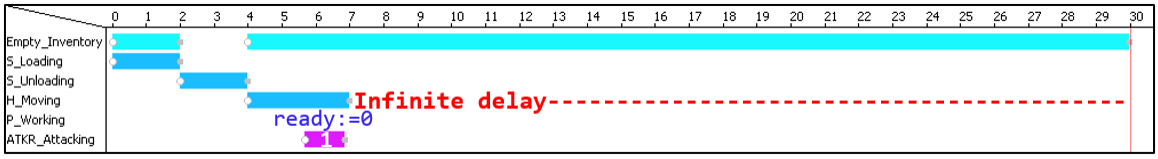


Figure 6.7: Infinitely delayed production due to data modification by an *informed* attacker

The simulation shows how the first modification attempt of the *informed* attacker was able to disrupt the processing action by modifying the ready variable to 0. The attack was timed in a way so that the modification was done during the moving action after which the ready variable would be read to start the processing action. The modification thus resulted in the *Processing Agent* waiting in the stalled state WFR, delaying the processing action. Unlike the *random* and *relentless* attackers, the *informed* attacker stops attacking once it has reached its goal of disrupting the system operations. Therefore, modifications made by *informed* attackers are never corrected by the attacker itself, and therefore, for a defenseless system, *informed* attackers are able to create infinite delays even with data modification attacks.

6.4 Executing Spoofing Attacks

After observing the impact of data modification and corruption attacks, we discuss message spoofing attacks intended to compromise the message-passing communications from *Storage Agent*. These attacks will require more information, e.g., physical or insider access to the system, necessitating higher privilege to know, manipulate, and/or spoof the exact messages being sent out from a component. We do not consider corrupted messages, as components will only respond to specified messages and ignore corrupted requests. Licit spoofed messages, however, can force some components to act earlier than intended, potentially creating a conflict and/or a malfunction during system operations. The simulations below show how different spoofing attacks (specified in Section 5.3) by *random*, *relentless*, and *informed* attackers negatively impact the MCCS mission objectives.

6.4.1 Spoofing by a *Random* Attacker

First, we execute the system model in parallel with a *random* attacker that can spoof the message-passing communications of the MCCS at irregular time intervals. Example simulation runs in UPPAAL depicting the impact of spoofing attacks made through a spoofed *Storage Agent* are shown below.

Figure 6.8 depicts the case of spoofed loaded! messages sent by a *random* attacker. Attempt 3 of the attacker succeeded in sending a loaded! message to the *Control Agent* C before the actual loading could be finished. As a result, C instructed the *Handling Agent* H to start moving, which performed all checks and made a transition

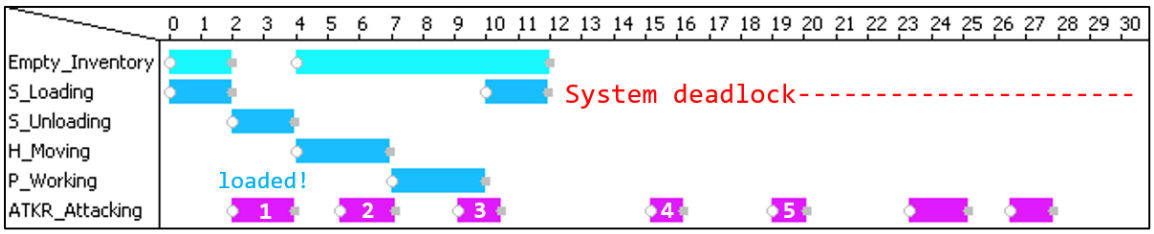


Figure 6.8: Simulation reaching a *deadlock* due to spoofed messages (loaded!) sent by a *random* attacker

to its Move state after sending the unload! message. When S finally finished the loading, it needed an unload! message to perform subsequent actions. But since that message had already been sent beforehand, S kept waiting, which in turn, made C wait for an unloaded! message from S. The system entered a state of *deadlock* due to the disrupted sequence of commands by the spoofed message. A *deadlock* is a system state where there are no outgoing action transitions from that state or any of its delay successors [BDL04]. Encountering a state of *deadlock*, therefore, essentially represents a case of system malfunction where all system activities have been halted before production could naturally end.

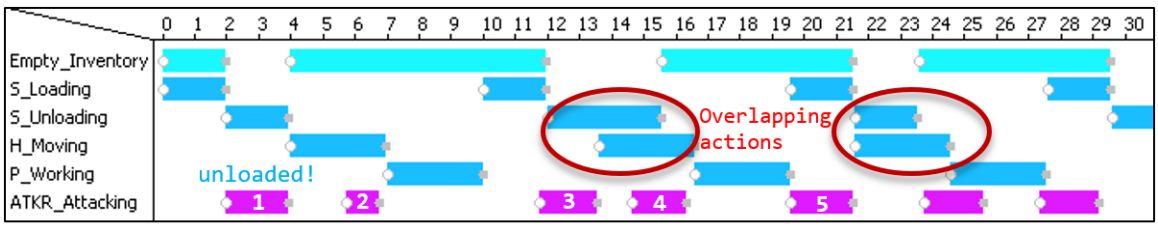


Figure 6.9: Simulation depicting hastened production due to spoofed messages (unloaded!) sent by a *random* attacker

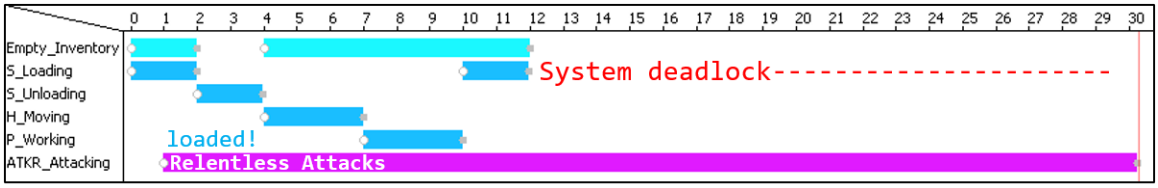
Next, we considered a *random* attacker sending spoofed unloaded! messages that resulted in an interesting phenomenon, illustrated in Figure 6.9. At first glance, it

might seem that the production process is proceeding smoothly despite several spoofing attempts by the attacker. However, when we look closely at the attempts 3 or 5, we can see that the attacks had forcefully activated the moving action of the *Handling Agent* H before the material unloading could finish. In essence, the spoofing forced H to act prematurely and move a material that did not exist. By the time we reached 28 time units, the production of the fourth unit of material had already started, which, according to the specified time constraints, is impossible.

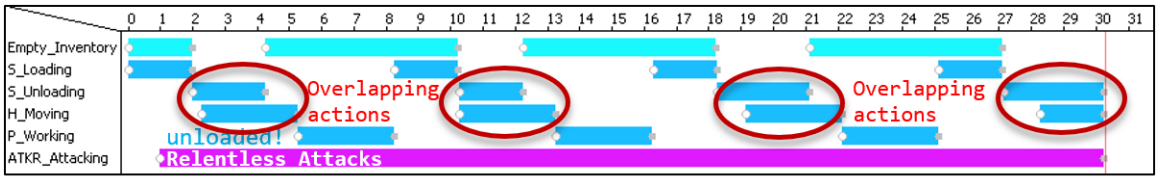
6.4.2 Spoofing by a *Relentless* Attacker

Next, we do the parallel execution of the system model and a *relentless* attacker that can incessantly spoof the message-passing communications of the MCCS. The following example simulations in UPPAAL show the impact of spoofing attacks made through a spoofed *Storage Agent*.

Figure 6.10a and Figure 6.10b show how continuous spoofing attacks using spoofed loaded! and unloaded! messages, respectively, by a *relentless* attacker is able to compromise the MCCS mission objectives. Spoofing attack with the loaded! message achieved a similar impact as the *random* attacker, forcing the system operations to malfunction, albeit with more concentrated attacks. Spoofing attacks with the unloaded! message, however, achieved a much higher impact, causing the unloading and moving tasks to overlap during each production cycle. The result is an extreme case of hastened production, where the apparent production of the fourth material unit had already started at 25 time units. If we had not considered the minimal one time unit to search for a vulnerability, the relentless attacks in the case of the spoofed



(a) Simulation reaching a *deadlock* due to spoofed messages (loaded!) sent by a *relentless* attacker



(b) Simulation depicting hastened production due to spoofed messages (unloaded!) sent by a *relentless* attacker

Figure 6.10: Simulations showing the impact of spoofing attacks by *relentless* attackers

loaded message attacks would be able to disrupt the production of the first material unit as well.

6.4.3 Spoofing by an *Informed* Attacker

Finally, we execute the system model in parallel with an *informed* attacker that can perform timed spoofing attacks to compromise the message-passing communications of the MCCS. An example simulation run depicting the impact of spoofing attacks made through a spoofed *Storage Agent* is shown in Figure 6.11.

The simulation shows how the same attacker can perform spoofing attacks with different spoofed messages. Attempt 1 of the *informed* attacker spoofs an unloaded! message, causing the system component actions to overlap. Conversely, Attempt 2 is made with a spoofed loaded! message, leading to a system malfunction. The two

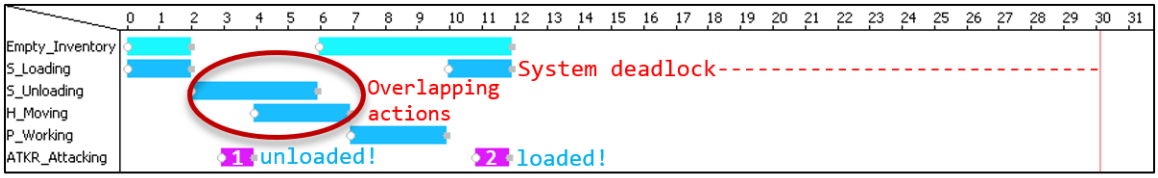


Figure 6.11: Simulation depicting both a case of *deadlock* and hastened production due to timed spoofed messages (loaded! and unloaded!) sent by an *informed* attacker

attempts are made during the times when the *Control Agent* is waiting for a legitimate unloaded! or a loaded! message, respectively. As a result, both attempts succeed in an exploit, albeit resulting in different consequences. This example shows how other spoofing attempts made using different system commands may be able to create widely varying impacts on the system and its operations.

Compared to the slow and sometimes automatically mitigated tampering attacks, the spoofing attacks by all types of attackers had an immediate impact on the system in the form of a system malfunction (*deadlock*) or hastened production (overlapping actions). Systems malfunctions stop the system from proceeding any further with the production, whereas hastened productions show apparent completion of mission objectives before the expected time. As the imposed time constraints do not allow that to happen, we can determine that each unit of material was not processed properly. For example, since the materials are being moved before unloading, futile moving and processing actions are being performed in some cases. This creates an impact in the form of unnecessary resource usage. The material units left in the inventory (not moved) from the previous cycles could also result in a raw material pileup depending on the structure of the inventory and result in other impacts on the MCCS operations.

Table 6.1: Attacks violating system invariants and forcing unsafe states

Attack Type	Violated Invariants	Forced *deadlock*	Forced *time deadlock*	Forced WFM	Forced WFI	Forced WFR
No Attack	N/A	No	No	No	No	No
Data Corruption	I, II	No	Yes	Yes	Yes	Yes
Data Modification	I, II	No	Yes	Yes	Yes	Yes
Message Spoofing (loaded!)	I	Yes	No	No	No	No
Message Spoofing (unloaded!)	N/A	No	No	No	No	No

6.5 Invariant Reinspection

After introducing the attackers to the system, we can revisit the system invariants (refer to Section 4.3) to quickly identify if there are observable impacts on the system objectives. Table 6.1 summarizes the capability of the discussed attacks to force the system to violate its system invariants. Undesirable results for the system operations are marked in red. It is important to note that for a particular attack type (e.g., data corruption), the results presented are the same for all three attacker types.

All attacks except for spoofed unloaded! message attacks violate *Invariant I*. This violation indicates that the system mission objectives could not be reached within the specified time and is a quick way to know that there is something wrong with the system operations. To identify the exact reasons behind the violation, we can study the associated simulation traces to pinpoint the cause, e.g., *time deadlocks* in case of tampering attacks or *deadlocks* in case of loaded! spoofing attacks.

All tampering attacks violate *Invariant II*, leading to one or more stalled states. In the absence of a defensive mechanism, the system has to wait forever in these states. Although not a *deadlock*, with no recovery mechanism, stalled states represent an infinite delay for data corruption attacks, creating a substantial impact on the system. Sustained stalled states also imply not completing an intended action within

the specified time constraints, leading to a *time deadlock*. By using specific queries (e.g., E$\Diamond$ $H.\mathit{WFM}$), we can quickly identify which stalled states were encountered. For both tampering attacks, all stalled states can be reached.

6.6 Conclusions

By developing a system and an attacker model beforehand, we can move towards executing the models in parallel to visualize the impact of attacks on the system's mission objectives. In this section, we discussed the execution of the system model with three different attacker types (*random*, *relentless*, and *informed*) performing various data tampering and spoofing attacks on the MCCS. We observed various simulations of the attacked system to discern how the indiscriminate attacks by *random* attackers differed from the incessant attacks by *relentless* attackers or targeted timed exploits by the more resourceful *informed* attackers. Additionally, we discussed how the system invariants could be revisited to quickly identify (i.e., answer **RQ1**.) if there are observable impacts on the system objectives. The observations provide insights into different attacker behaviors, their attack patterns, and how the level of impact differs even when performing the same attacks.

At the end of this chapter, we are left with:

1. *Simulation traces of various tampering and spoofing attacks on the system by different attackers*

2. *Reverification of the intended system behavior to identify the existence of impact on system mission objectives*

Based on the types of impact identified in this state, the next and final stage in our impact analysis approach focuses on impact quantification to compare and better understand the attackers and their respective capabilities.

Chapter 7

Impact Analysis

After observing simulations involving the different attacker behaviors and their various impact on the system operations, we now gather statistical evidence to corroborate our findings. In the last stage of our impact analysis approach (refer to Chapter 3), we move away from the models and simulations and focus on quantifying the impact using Statistical Model Checking (SMC) to characterize the impact of the discussed attacks and attackers. Section 7.1 provides a broad overview of the Impact Analysis stage. Section 7.2 shows how to specify SMC queries relevant to the system's mission objectives. Section 7.3 elucidates the way to interpret the acquired SMC results to characterize the different types of impact on the system's mission objectives. Section 7.4 discusses why defining an impact ranking model based on the impact analysis results is challenging. Lastly, Section 7.5 concludes the Impact Analysis stage.

7.1 Overview

To get a more concrete idea about how different attackers and their attacks can impact ICS mission objectives, a way to quantify the impact is necessary. The quantification will enable us to compare and contrast between the capabilities of different attackers. Additionally, we will be able to characterize how the same attacks, when performed by different attackers, can result in widely varying impacts on the system. In the fourth and final stage (see Figure 7.1) of our impact analysis approach, we focus on devising ways to quantify, and thus, characterize the impact of the different attackers and attacks discussed in Chapter 5.

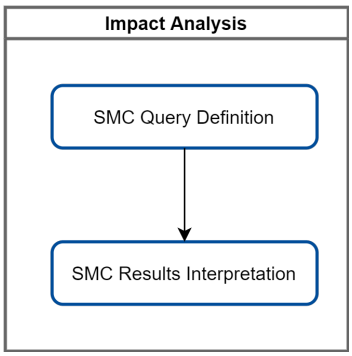


Figure 7.1: The Fourth Stage: Impact Analysis

First, in the *SMC Query Definition* activity, we define different SMC queries to gather statistical data about several system aspects that relate to the achievement of system mission objectives. For example, the queries chosen for inspection can be based on a metric useful for reasoning about a particular aspect of the system objectives. Next, in the *SMC Results Interpretation* activity, these queries can be used to obtain information about the selected aspects during normal system operations and compare them with different system configurations, such as when the system operations are

influenced due to an attack. The results from the SMC queries can be further analyzed to interpret the impact of different attacks and gain insight into their severity (i.e., answer **RQ2**.). Furthermore, the results will enable the comparison of impact caused by different attackers and their various attacking capabilities (i.e., answer **RQ3**.).

7.2 SMC Query Specification

In this section, we show how to define SMC queries to reason about different system aspects related to the achievement of system mission objectives.

While using classical model checking and revisiting the system invariants enabled us to quickly check for the correctness of the system operations (i.e., answer **RQ1**.), the use of SMC queries allows us to go for quantitative analysis of the system mission objectives. In other words, while classical model checking queries (i.e., system invariants) are concerned with the existence of some impact on the system operations (i.e., Is there an impact?), SMC queries allow us to dive a level deeper by quantifying the potential impact (i.e., How much of an impact?). Therefore, for each type of attacker and attack discussed, the next step after identifying system invariant violations is to specify a set of SMC queries. The queries are aimed at quantifying the level of completion of system mission objectives (i.e., answer **RQ2**.) and obtaining insights into the potential impact of different attacks (i.e., answer **RQ3**.).

7.2.1 Statistical Model Checking

Statistical Model Checking (SMC) [You05, BDL+12] has been proposed as a middle-ground between traditional testing and classical model checking techniques. SMC allows us to decide whether the formal model of a system satisfies a property with a certain confidence level δ and maximum error limit ϵ by monitoring some stochastic simulations and leveraging results from the statistics area [DLL+15]. SMC has found its use in a variety of domains including dependable [KBTJ20, KS17, HK19, CFN+18], energy-aware [DDL+13], and heterogeneous communication systems [BBB+10], and systems biology [CFL+08, DDL+12], among others.

SMC has seen a recent surge in studies focused on identifying the impact of attacks on various ICS. Kumar and Stoelinga [KS17] showed how attack-fault trees could be translated into stochastic timed automata to estimate the cost, time, and damage of system faults and security threats using SMC. Huang et al. [HK19] specified the safety and security properties of a cooperative automotive system and analyzed the impact of various attacks compromising the communication between the cooperating vehicles. Cheh et al. [CFN+18] used SMC to analyze attackers with capabilities to remove, delay, or insert network messages and control commands to manipulate the signaling components of a railway system. Munteanu et al. [MPM20] used the MODEST toolset to perform an SMC analysis of a quadruple-tank water system to assess the physical impact of cyber attacks and performance of intrusion detection systems.

The version of Uppaal selected for this study (Uppaal v4.1.25-5) is extended with SMC capabilities, enabling us to reason on networks of complex real-timed systems

with a stochastic semantic [DLL+15]. Over classical SMC model checkers such as YMER [You05], VESTA [SVA05], and MRMC [KZH+11], the UPPAAL SMC extension boasts the advantage of handling timed systems and offers a rich framework to specify stochastic timed systems [DLL+15]. Additionally, UPPAAL outperforms other existing SMC tools for timed automata such as PRISM [KNP11] in terms of overall flexibility and usability [NAAA18].

In our work, we use SMC to analyze the achievable impact of potential attacks on the system model (i.e., answer **RQ2**.). More explicitly, the use of SMC enables us to ask questions about "by how much" an attack is able to impact the system mission objectives and gather statistical results to quantify the impact of such attacks. The various SMC query capabilities of UPPAAL also allow us to take into account different system perspectives relating to the system mission objectives and define domain-agnostic metrics that can help characterize the impact from such perspectives.

The syntax and semantics of SMC queries in UPPAAL can be found in Appendix A.2.

7.2.2 Specifying SMC Queries

To perform a more detailed analysis of the impact of data tampering and spoofing attacks executed on the MCCS, we gather results from three SMC queries. For systems in general, we define three SMC queries based on three system aspects: (1) the reachability of system mission objectives, (2) the expected level of mission completion, and (3) the average duration of delayed operations within a specified time. As the MCCS is a manufacturing system, these generic queries are adapted

to represent the successful completion of the manufacturing process, the number of products manufactured at the end, and the total duration of delayed manufacturing operation, respectively. More concretely, the queries are as follows:

SMC Query 1: The probability of reaching system mission objectives (manufacturing M units) within t time units.

SMC Query 2: The expected level of mission completion (average units of material produced) within t time units.

SMC Query 3: The average expected delay in system operation (delayed production) in a timespan of t time units.

Expressions 7.1–7.3 below show how to specify the three SMC queries using the verifier of UPPAAL. Parameters *sim_time* and *sim_runs* represent the simulation time and the total number of simulation runs, which can be modified as needed. State variable *processed_mat* is a system-specific variable used to keep track of the level of system mission completion (total materials units produced), and *total_delay* measures the total duration of delayed operations through the use of stalled states.

$$\Pr[total\_time \leq sim\_time; sim\_runs] \ (\Diamond\ processed\_mat == M) \tag{7.1}$$

$$\mathrm{E}[total\_time \leq sim\_time; sim\_runs]\ (max : processed\_mat) \tag{7.2}$$

$$\mathrm{E}[total\_time \leq sim\_time; sim\_runs](max : total\_delay) \tag{7.3}$$

These three SMC queries will be used throughout the remainder of this work to illustrate how we can obtain statistical data about the corresponding system aspects. Results from these queries during normal system operations will give us the ideal values for the MCCS. In this way, deviations from the ideal values (e.g., due to an attack) will allow us to characterize the impact of that attack through the quantitative results.

7.3 SMC Results Interpretation

In this section, statistical results from SMC queries specified in the previous activity are obtained and analyzed to assess the potential impact of attacks on the system's mission objectives.

In this work, we will adapt the SMC queries to obtain results for hourly and daily mission objectives. For demonstration, each time unit will represent ten seconds, mandating 100 seconds to produce each material unit for the MCCS. This means that the MCCS is expected to produce 36 material units hourly and 864 material units daily. Therefore, the three SMC queries will be used to analyze the impact on the mission objective of reaching typical hourly production of 36 and daily production of 864 material units without any interruptions.

It is important to note that we are considering a defenseless system, where there is no way to restore the normal operations (e.g., through a data recovery in case of tampering) except through the regular system assignment of variables or by the attackers themselves. In the following subsections, we will quantify the impact of data

corruption, data modification, and spoofing attacks using the SMC Queries defined in the previous section (see Expressions 7.1–7.3). All SMC query results will be obtained for 1000 sample runs. Furthermore, undesirable results will be marked in red, whereas results that require further attention will be depicted in violet.

7.3.1 Impact of Data Corruption

We begin by assessing the impact of data corruption attacks (specified in Section 5.3) on the MCCS operations. Table 7.1 presents results pertaining to the impact on the material production, whereas Table 7.2 presents results related to system delays.

Results from Query 1 (columns 2-3) in Table 7.1 allows us to see the reachability of the MCCS mission objectives. For example, 864 units of material should ideally be produced within a day. Since we consider fixed time constraints for the MCCS operations, 864 materials should also never be produced before a day is over (e.g., in less than 23 hours). Such is the case for a system without any attacker, where there is a 100% probability of producing all 864 materials within a day but a 0% probability of producing the same within 23 hours. Any deviation from these two values allows us to identify the cases where production was either disrupted or hastened, respectively and quantify the degree of the impact. For data corruption attacks by all three types of attackers, the MCCS mission objectives could never be reached in time due to the delays imposed by *time deadlocks*. As such, we observe probability measures of 0% to reach the expected daily production for all attacker types.

Table 7.1: Impact of data corruption attacks on material production

Attacker Type	Probability of producing 864 units of material (%)		Average units of material produced (number of materials)	
	within 1 d	within 23 h	within 1 h	within 1 d
No Attacker	100	0	36	864
Random Attacker	0	0	2.12	2.13
Relentless Attacker	0	0	0.39	0.42
Informed Attacker	0	0	9.6	9.69

On the other hand, results from Query 2 (columns 4-5) in Table 7.1 enables us to assess the expected level of system mission completion. As mentioned before, the MCCS is expected to manufacture 36 and 864 units of material hourly and daily, respectively. Data corruption attacks are able to create system delays, and therefore, all materials are never produced when there is an attacker corrupting data within the system. *Relentless* attackers seem to be the most effective at causing *time deadlocks* through incessant data corruption attacks as not a single unit of material is produced (on average) when such an attacker is present. In comparison, *Random* and *informed* attackers are slightly slower at causing a compromise. However, in both cases, the number of material units produced remains below ten. For a defenseless system, any data corruption attack has further implications; the first successful exploit traps the system in a stalled state forever and prevents the production of more units of material regardless of the simulation time. Thus, we see the MCCS producing the same units of material for both hourly and daily cases.

Lastly, results from Query 3 (columns 2-3) in Table 7.2 allows us to see the average expected delay in the system operations, whereas columns 4-5 represent the results in the percentage of the total simulated time (e.g., a day). Delays are caused by

Table 7.2: Impact of data corruption attacks on the production process

Attacker Type	Average total production delay (minutes)		Production delay (%)	
	within 1 h	within 1 d	within 1 h	within 1 d
No Attacker	0	0	N/A	N/A
Random Attacker	55.54	1435.69	92.57	99.70
Relentless Attacker	58.63	1438.60	97.72	99.90
Informed Attacker	43.61	1422.99	72.68	98.82

time deadlocks, and for a defenseless system, are determined by the first successful attempt by an attacker. This situation is reflected in the extreme average production delay of approximately 99% for daily production in the case of all attackers since the time to cause the first compromise compared to the total production time of a day is significantly smaller. The results for hourly production are slightly different, where *informed* attackers seem to be the least impactful, causing a delay of 72%, whereas *relentless* attackers still lead all the other attacker types with a production delay close to 98% of the total production time.

In general, the impact of data corruption attacks is rather extreme, never allowing the MCCS to reach hourly or daily mission objectives. All attacker types seem to be almost equally effective in causing an impact when data corruption attacks are concerned. To be more precise, *relentless* attackers seem to be the most impactful, and *informed* attackers seem to be the least impactful when comparing results from Queries 2 and 3. *Random* attackers are almost always in the middle, possibly due to their random nature that sometimes leads to a behavior similar to relentless attackers. However, the impacts in all cases are so high that the difference between the attacker types is less significant.

Table 7.3: Impact of data modification attacks on material production

Attacker Type	Probability of producing 864 units of material (%)		Average units of material produced (number of materials)	
	within 1 d	within 23 h	within 1 h	within 1 d
No Attacker	100	0	36	864
Random Attacker	0	0	16.33	**315.38**
Relentless Attacker	0	0	14.61	334.93
Informed Attacker	0	0	18.18	**23.83**

7.3.2 Impact of Data Modification

Next, we analyze the impact of data modification attacks (specified in Section 5.3) on the MCCS operations. Table 7.3 presents results pertaining to the impact on the material production, whereas Table 7.4 presents results related to system delays.

Results from Query 1 (columns 2-3) in Table 7.3 allows us to see the reachability of the MCCS mission objectives. Similar to the data corruption attacks, all data modification attacks create delays imposed by *time deadlocks* and prevent the system from reaching its mission objective in time. Therefore, we observe identical probability measures of 0% to reach the expected daily production for all three attacker types.

Results from Query 2 (columns 4-5) in Table 7.3, however, are quite different compared to that of data corruption attacks. Since data modification attacks are able to create system delays, all materials are never produced when there is an attacker modifying data during the system operations. However, the degree of impact in the case of *random* and *relentless* attackers is significantly less when compared to the data corruption cases.

For example, for *relentless* attackers causing data modification, the MCCS is now able to produce about 14 and 335 material units hourly and daily, respectively. This leads to about 38% of the material units being produced in both cases which is significantly higher than both hourly and daily cases (approximately two material units) in case of data corruption attacks. This happens due to the arbitrary data modification attacks self-correcting their own modifications, causing the system to start operating again. This behavior remains true throughout the simulated time, which is why we see similar results for the daily and hourly cases.

Random attackers, on the other hand, end up producing about 16 and 315 materials hourly and daily, respectively. Interestingly, the hourly production of about 16.33 (44%) material units is higher than *relentless* attackers, whereas the daily production of about 315.38 (36%) material units is slightly lower than that of *relentless* attackers. We find that the *random* attacker ends up performing fewer attacks compared to the *relentless* attacker within the same period. While this results in fewer successful modifications, it also results in fewer unintentional corrections by the attacker. That is also why we see the *random* attacker being more effective when a longer simulated time (a day) is considered.

In stark contrast to both *relentless* and *random* attackers, *informed* attackers do not attack once their attacks have successfully caused a compromise. This means that their modifications are not self-corrected, and they can create sustained impacts similar to data corruption attacks. This behavior can be seen in a system under the influence of an *informed* attacker producing approximately 18 material units hourly (slightly less impact than the other two attackers) and a staggeringly low number of

Table 7.4: Impact of data modification attacks on the production process

Attacker Type	Average total production delay (minutes)		Production delay (%)	
	within 1 h	within 1 d	within 1 h	within 1 d
No Attacker	0	0	N/A	N/A
Random Attacker	31.95	901.56	53.24	62.61
Relentless Attacker	34.04	861.04	56.74	59.79
Informed Attacker	28.05	**1397.14**	46.74	**97.02**

23 material units daily. Therefore, data modification attacks seem to be the most impactful when done by *informed* attackers, especially when longer simulated times are concerned.

Lastly, results from Query 3 (columns 2-3) in Table 7.4 allows us to see the average expected delay in the system operations, whereas columns 4-5 represent the results in the percentage of the total simulated time (e.g., a day). Since data modification attacks hinder the production through delays caused by *time deadlocks*, these results are directly related to the number of material units produced on average (see Table 7.3). *Relentless* attackers create similar delay percentages both in the hourly and daily case, whereas for the *random* attacker, the delay caused in the daily case is greater. In case of the *informed* attacker, however, the delays are significant when daily production is concerned. Since these attackers do not self-correct their own modifications, they result in a daily production delay of approximately 97%, which is considerably higher compared to the approximately 60% delays caused by *random* and *relentless* attackers.

Overall, data modification attacks are less impactful than data corruption attacks since the attackers may unintentionally correct the modifications themselves. The difference is more visible in the case of *relentless* and *random* attackers, as systems under the influence of those attackers are limited to producing less than half the expected material units in both hourly and daily cases. While data modifications by these attackers no longer result in infinite delays, frequent attacks by an attacker can still result in a significant impact as the delays over time add up. *Informed* attackers, however, can create sustained delays even with data modification attacks, and thus, are the most impactful attackers among all the attackers considered.

7.3.3 Impact of Spoofing

Finally, we assess the impact of spoofing attacks (specified in Section 5.3) on the MCCS operations. We start by discussing results related to the material production caused by spoofed loaded! and unloaded! messages. Results for *Informed* attackers are discussed separately since their attacks can involve both loaded! and unloaded! messages.

Impact of Spoofed loaded! Messages

Results from Query 1 (columns 2-3) in Table 7.5 allows us to see the reachability of the MCCS mission objectives for *random* and *relentless* attackers. Similar to both data tampering attacks, spoofing attacks with loaded! messages also prevent the system from reaching its mission objective in time, albeit through different means. These

Table 7.5: Impact of spoofed loaded! messages on material production

Attacker Type	Probability of producing 864 units of material (%)		Average units of material produced (number of materials)	
	within 1 d	within 23 h	within 1 h	within 1 d
No Attacker	100	0	36	864
Random Attacker	0	0	2.24	2.33
Relentless Attacker	0	0	1	1

attacks lead the system to *deadlocks* and leave no chance for the MCCS to produce all required materials. This behavior is reflected in the probability measures of 0% to reach the expected daily production for both attacker types.

Results from Query 2 (columns 4-5) in Table 7.5, however, are similar to that of data corruption attacks. Both *random* and *relentless* attackers create *deadlocks* that cause sustained malfunctions on the system, not allowing the system to produce any more materials after the first successful exploit. *Relentless* attackers seem to have the edge over *random* attackers, as they do not let more than one unit of material be produced in both hourly and daily cases. The production of one material unit is always possible since we consider a minimal vulnerability searching period of one time unit and an attack duration of at least one time unit. Combined, they only allow the first spoofing attempt to take effect after two time units, and by that time, the MCCS already finishes the loading process (the only time when spoofed loaded! messages can be successful). If we allowed the *relentless* attacker to be able to attack from the very beginning, they would always be able to prevent the production of the first unit of material as well.

Delays are caused by *time deadlocks*, and as such, are not seen in the case of spoofing attacks with loaded! messages. Therefore, we do not discuss results from Query 3 since they always result in a delay of 0.

Impact of Spoofed unloaded! Messages

Results from Query 1 (columns 2-3) in Table 7.6 allows us to see the reachability of the MCCS mission objectives for *random* and *relentless* attackers. Reaching the mission objectives before a day has passed should be impossible. However, unloaded! message spoofing attacks can (seemingly) hasten production by overlapping system processes, as shown by the high probability of 77.5% in case of *random* attackers and a certain probability of 100% in case of *relentless* attacker to produce all 864 units of materials within 23 hours.

Results from Query 2 (columns 4-5) in Table 7.6 show the average expected production of material units when under the influence of spoofing attacks. As an example, the MCCS under attack by a *relentless* attacker ends up producing approximately 40.63 (113%) material units hourly and 983.31 (114%) material units daily. Here, the case of hastened production is reiterated by the unusually higher number of units manufactured by both attackers. Due to their more frequent attacks, and subsequently, a higher number of successful exploits, spoofed unloaded! messages by *relentless* attackers are able to create more overlapping actions, which results in a higher number of material units (apparently) produced for both hourly and daily cases.

Table 7.6: Impact of spoofed unloaded! messages on material production

Attacker Type	Probability of producing 864 units of material (%)		Average units of material produced (number of materials)	
	within 1 d	within 23 h	within 1 h	within 1 d
No Attacker	100	0	36	864
Random Attacker	100	77.5	37.2	902.89
Relentless Attacker	100	100	40.63	983.31

Similar to spoofed loaded! messages, spoofed unloaded! messages do not cause *time deadlocks*. Once again, we do not discuss results from Query 3 since they always result in a delay of 0.

Impact of Spoofing Attacks by an *Informed* Attacker

The impact of spoofed loaded! and unloaded! messages on the MCCS is shown by the SMC results of Table 7.7. The reason behind separating the *informed* attacker is twofold: (1) we want to show how the impact of performing spoofing messages of different types is dominated by the attack with the stronger impact, and (2) how the combination of more than one spoofing attack can result in a different impact than the individual attacks.

Because the *informed* attacker can perform spoofing attacks with both loaded! and unloaded! messages, they can cause both *deadlocks* (system malfunctions) and (apparent) hastened production, respectively. The hastening of production due to overlapping system processes can be considered the milder impact, as the system does end up reaching its mission objectives, albeit with some problems in its processes or components. Compared to this, a full system malfunction caused by *deadlocks*

Table 7.7: Impact of both spoofed loaded! and unloaded! messages on the MCCS

Attacker Type	Probability of producing 864 material units (%)		Average units of material produced (number of materials)		Average total production delay (minutes)		Production delay (%)	
	within 1 d	within 23 h	within 1 h	within 1 d	within 1 h	within 1 d	within 1 h	within 1 d
No Attacker	100	0	36	864	0	0	N/A	N/A
Informed Attacker	0	0	25.29	51.51	13.12	1346.20	21.86	93.49

is a much bigger issue and seems to always take precedence when SMC queries are consulted. This is shown by the system showing 0% probability to reach its daily mission objectives (columns 2-3), and never producing all expected materials on average (columns 4-5). There are, however, cases where most attacks are made with spoofed unloaded! messages or the attacks with spoofed loaded! messages not being successful within an hour, thus producing all 36 material (and sometimes more). In contrast, when daily production with a larger simulation time is considered, attacks with spoofed loaded! messages are always successful, and the production of units is limited to approximately 51 material units.

On the other hand, the ability to spoof two different messages can lead to a completely different type of impact. This is shown by the time delays (columns 6-9) caused by *time deadlocks*, which can be caused by neither spoofed loaded! messages nor spoofed unloaded! messages individually. This happens due to a certain sequence of actions by the *informed* attacker. When the attacker first spoofs a loaded! message, it causes the *Control Agent* C to prematurely activate the moving action by sending a prepare! message to *Handling Agent* H. Normally, this would cause a *deadlock* since the unload! message would be sent by H before the *Storage Agent* S can finish the loading process, and with no unload! message afterwards, S would not be able to proceed. This

sequence of actions would also mean that C would never receive an unloaded! message from S since the unloading action would never begin. However, if a spoofed loaded! message is followed by a spoofed unloaded! message, this would allow C to proceed to the initialization phase, commanding *Processing Agent* P to start processing the material. P would get stuck in the stalled state WFM since the material was never unloaded (value of status is still 1) due to the premature activation of H and P. In this way, spoofing attacks by *informed* attackers can result in *deadlocks*, (apparent) hastened production, and *time deadlocks* due to the combined impact of two different spoofed messages.

Overall, attacks with spoofed loaded! messages can achieve impacts similar to the data corruption attacks by causing system malfunctions, while attacks with spoofed unloaded! messages can cause (seemingly) hastened production by forcing overlap of system actions. *Informed* attackers with the capability to spoof both messages can cause system delays in addition to these impacts and are the most versatile among all three attackers considered.

7.4 Rankings Based on Impact

While the impact analysis results provide us with statistical data on impacted system objectives, defining an impact ranking model based on such results is not a trivial task. In this section, we discuss the challenges in ranking attacks and attackers based on impact.

7.4.1 Ranking Attacks

Determination of the most impactful attack is not always straightforward. Based purely on the SMC query results of Section 7.3, data tampering attacks may seem more impactful than message spoofing attacks. However, the severity of the impact and the time, cost, and effort required to repair the damages are also important considerations. While system malfunctions (*deadlocks*) may seem more dangerous than component action delays, the situation changes if the component is central to the system's functionality (e.g., the *Control Agent* in the MCCS). Tampering attacks are typically slow-acting, effective only when done between the timeframe between the proper assignment and subsequent inspection of a state variable, and may be mitigated by the system's normal operations. This means that systems with smaller, faster task scheduling are inherently more resilient to tampering attacks than the others. Spoofing attacks, on the other hand, almost always result in instantaneous compromises, leaving a much smaller window to be detected and/or recovered from than data tampering attacks.

The impact of certain attacks might not be obvious at first glance. For example, the unloaded! message spoofing attack in Section 6.4 may seem innocuous, even helpful, as the system seemingly produces more material units than required. However, the results from SMC Queries 7.1 and 7.2 (refer to Section 7.3) go against the requirement specifications and mandates further scrutiny. By inspecting the simulations traces, we identify that many unloaded materials not moved from the previous cycle are left due to the forced moving action by the spoofed command. Such overlapping actions can easily cause component or material damage and may lead to a material

pileup. These actions can be severely disruptive depending on the storage type or how materials are handled in the production process and result in lower quality and/or defective products. Additionally, when attacked by this type of attack, the MCCS passes verifications for both timely and uninterrupted mission completion (i.e., system invariants). This particular case, therefore, exemplifies how the impact of certain attacks may be missed by classical model checking but still be identified through SMC results.

7.4.2 Ranking Attackers

Judging only by the SMC query results, the *relentless* attackers almost always seem to be the most impactful attacker when compared to both *random* and *informed* attackers. However, we must bear in mind that we are assuming each attack of a particular type (e.g., data corruption attacks) are *equally likely to succeed* and will be *equally effective on the system*. This assumption is made since we are mostly concerned about the impact when such attacks are performed by different attacker types. In reality, only simple attacks that require little to no preparation at all, such as scripted behaviors, can be performed by *relentless* attackers. On the other hand, *random* attackers that had more time to prepare (in some random attacks) or *informed* attackers that intentionally took more time to prepare may have a higher chance of succeeding and causing a more pronounced impact on the system operations.

Another factor to determine the ranking between attacks is the idea of *ease of detection*. Before taking any defensive action to prevent or mitigate the impact of attacks, the effect of the attacks (e.g., the corruption of data) would first have to be detected.

Relentless attackers do not try to hide, and their behavior has no variations. Therefore, these attacks would probably be the easiest attacks to detect and take defensive measures against. In comparison, the *random* attackers have arbitrary behavior, and if the attacks are done at widely varying times and are sufficiently random in nature, they may be much harder to detect. Attacks by *informed* attackers would be even harder to detect, as they can choose when to attack, may know vulnerable moments in system operations, and only attack at times when they cannot be detected or when the detection does not prevent the attack from causing an impact on the system operations.

7.5 Conclusions

Characterization of the impact of different attacks through concrete evidence can enable us to raise awareness among system developers and operators. To this end, in this section, we defined three domain-agnostic SMC queries related to the system aspects of mission objective reachability, expected level of mission completion, and expected system delays. Furthermore, we gathered results from the SMC queries for typical system operations and compared them to results for systems under attack from different attackers. The resulting analyses allowed us to characterize the impact of different tampering and spoofing attacks (i.e., answer **RQ2**.) as well as how different attackers can cause varying impacts (i.e., answer **RQ3**.) with a single or combination of attacks.

At the end of this chapter, we are left with:

1. *A set of domain-agnostic SMC queries*
2. *Statistical data from the specified SMC queries to quantify impact*
3. *Insight into the impact on the system mission objectives caused by different attackers and their various tampering and spoofing capabilities*

Thus far, we have only considered our impact analysis approach on a defenseless system. More meaningful results can be obtained in a system with defenses against attempted modifications and corruptions, which is the focus of the next chapter.

Chapter 8

Defense Modeling

The identification and quantification of the impact of data tampering and spoofing attacks by different attackers allowed us to see how attacks against an ICS can impact a system's operation using varying means. In addition, it is important to see how the implementation of targeted countermeasures in the system can reduce that impact, allowing the system to resume its normal operations. In this chapter, we take a brief look at systems with defensive mechanisms and whether they are more resilient to the impact of different attacks compared to defenseless systems. Section 8.1 provides a broad overview of the Defense Modeling process. Section 8.2 shows how to model targeted defenses within the system. Section 8.3 explores different levels of defensive capabilities against data corruption attacks. Section 8.4 discusses why ranking defenses based on the analysis results is difficult. Section 8.5 elaborates on the limitations of modeling system defenses. Section 8.6 briefly discusses existing works related

to defense or defender modeling. Lastly, Section 8.7 concludes the defense modeling process.

8.1 Overview

Prior to this section, we have considered a system model where no countermeasures against the discussed attacks exist. As a result, we have performed activities in the four stages of our impact analysis approach to such a defenseless system where a critical impact, such as system malfunctions (*deadlock*) or delays (*time deadlock*) could inhibit the system operations for an extended time. Despite the myth that "ICS are immune to cyberattacks" due to their typically proprietary nature and reliance on "security by obscurity" [BL04], modern ICS may have some form of defensive mechanisms against common attacks. Consequently, our impact analysis approach should also consider such systems and how the impact of attacks by different attackers change based on the existence and capability of such defenses. Additionally, this will allow us to perform our impact analysis approach on a system with defenses and assess whether the impact of attacks can be reduced if defenseless systems were to be equipped with such a defense.

In this work, we use data corruption attacks (refer to Section 5.3) by *random*, *relentless*, and *informed* attackers (see Section 5.4) to redo parts of our impact analysis approach on a system with defenses. While many different types of defenses can be modeled, for demonstration, we will consider a simple detect and recover mechanism.

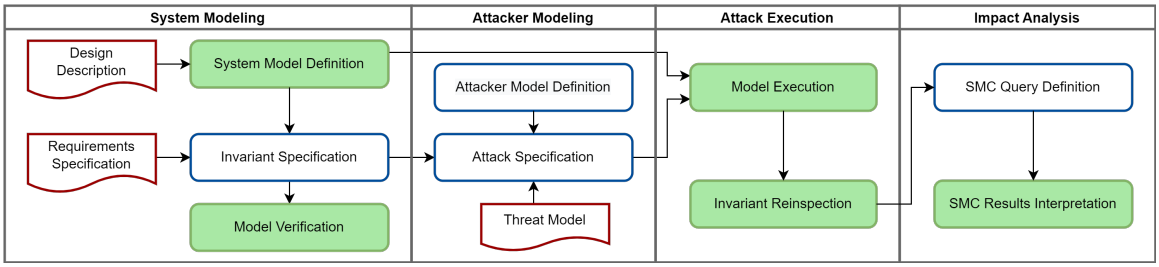


Figure 8.1: Required impact analysis activities for modeling defenses

The defense modeling requires performing certain activities in our impact analysis stage again, as depicted in green in Figure 8.1.

First, as part of the *System Modeling* stage, we consider an extended system model with agents responsible for detecting and recovering from attempted data corruptions. The agents will rely on process invariants to detect any corruption of data and recover from such anomalies. We also define the capabilities of the defenses by determining how fast and in what form they can recover from data corruption. This is followed by a system model verification to ensure that the system model, now augmented with defensive capabilities, still work as expected.

After the system model with defenses is built and verified, we execute our various attacker models in parallel with it in the *Attack Execution* stage to visualize how the defenses mitigate the impact of attacks. Additionally, we reinspect our system invariants to identify whether there is still an attack impact despite the existence of the defensive mechanisms. After the identification, we move to our *Impact Analysis* stage to obtain statistical data on the impact and compare them against the impact on the previously discussed defenseless system.

8.2 Modeling Defenses

We start by equipping the system model with targeted defenses against common attacks. In this section, we explain the concept behind modeling defenses within a system and demonstrate the process through different abstract system agents. In essence, this activity is an extension of the first stage of our impact analysis approach, namely the *System Modeling* stage, where we extend the system model capabilities with defensive mechanisms.

There are two important parts to modeling defenses; (1) the detection process and (2) the defensive action. Depending on the type of attack and its relative difficulty of detection, the detection process can be very fast, or it may take some time. If the attack and/or the effect of the attack on the system can be detected immediately, we can think about two types of defensive actions. Taking *immediate defensive action* against the system will essentially result in prevention as the effect of the attack (e.g., a data corruption) may not have enough time to result in an impact. On the other hand, taking a *delayed defensive action* may mean that the effect of the attack has already resulted in an impact (e.g., data corruptions leading to *time deadlock*), and at that point, the defensive action may be equivalent to that of a mitigation mechanism. The detection process itself may be delayed enough for the effect of an attack to cause an impact, in which case, even an instantaneous defensive action may result in mitigation, not prevention.

In this work, we focus on three types of defenses:

1. **Instant Defense:** The detection process, as well as the defensive action, are both instantaneous, essentially resulting in the prevention of an attack before it can cause an impact.

2. **Delayed Defense:** Both the detection process and the defensive action take some amount of time to complete, constituting a more realistic defensive mechanism. This is similar to mitigation of an attack since the attack may have already caused an impact by the time the defensive action has finished.

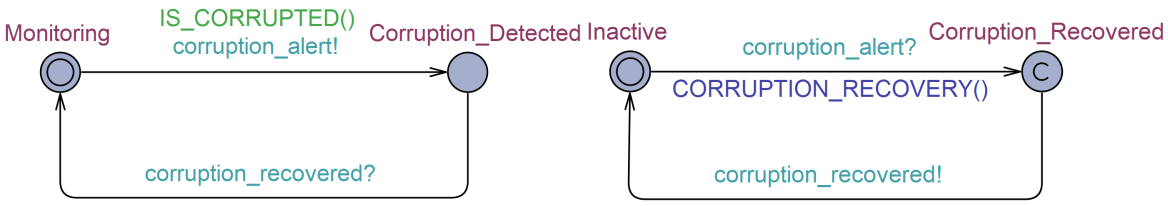
3. **Delayed Defense in Safe Mode:** A special case of the delayed defense where the system is taken to a safe mode during the defensive action where system processes are temporarily halted (so that a fault does not occur), and attackers are prevented from affecting the system processes.

The behavior and efficacy of these three types of defenses will be demonstrated using data corruption attacks against the system. Consequently, *the detection process* would involve detecting attempted data corruptions and *the defensive action* will entail a *corruption recovery* that can restore system variables to their ideal values.

8.2.1 Modeling Instant Defenses

To model a system with instant defenses, we must add a detection process and a defensive action capability to the system that can be performed instantaneously. For the MCCS, we can do such by extending the system model with two new agents focused on defense against attacks.

The first agent (shown in Figure 8.2a) is a *Monitoring Agent* MON that can monitor the shared variables in the system and raise an alert (corruption_alert!) whenever there is a corruption of such variables. The corruption can be detected by continuous monitoring of the shared variables (e.g., through an IS_CORRUPTED() Boolean function) and detecting whether a certain variable is assigned a value outside its allowed value range. For example, in case of the phase variable, the allowed values are 0, 1, 2, and 3, representing the four system phases. Therefore, at any point during the system operations, assigning any values other than these four allowed values would make MON send out a corruption_alert!.



(a) The *Monitoring Agent* Automaton (MON) (b) The *Recovery Agent* Automaton (REC)

Figure 8.2: Detection and Recovery Modeling

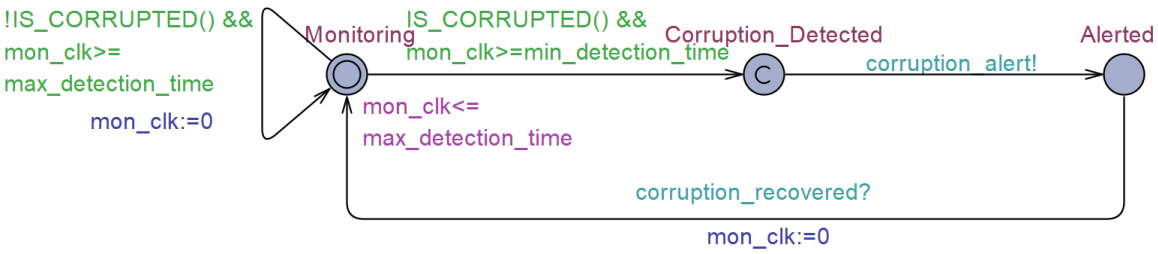
The corruption_alert? is received by the second agent, named *Recovery Agent* REC (shown in Figure 8.2b), that can take an instantaneous defensive action by performing an immediate corruption recovery. The recovery action (CORRUPTION_RECOVERY()) can be performed by determining the ideal state of the corrupted variable based on the current overall system state and reassigning to it the ideal value. Once the corruption is recovered, REC can send a corruption_recovered! message to MON, informing that the effect of the corruption has been removed and that MON can start monitoring the system again. The use of *urgent synchronization channels* allows both the detection and recovery action to be instantaneous, thus resulting in immediate recovery from any corruptions.

In essence, this kind of instant defensive mechanism is reflective of a prevention mechanism. As long as the data corruption is correctly identified, the effect of the attack (e.g., corruption of data) can be prevented immediately by performing instantaneous data recovery. This will prevent data corruption from affecting the system processes, essentially nullifying the impact of data corruption attacks. This duo of a monitoring and recovery agent will be used in our impact analysis approach to demonstrate the impact of attacks on a system that has instant defenses against data corruption attacks.

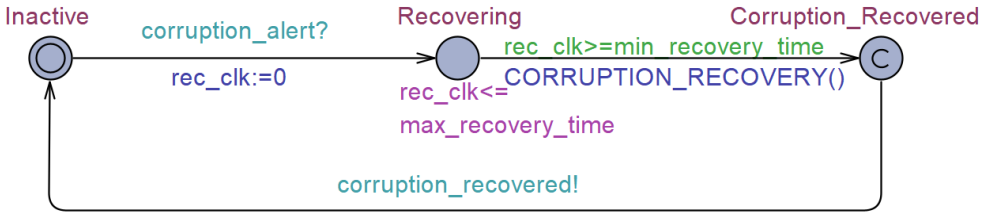
8.2.2 Modeling Delayed Defenses

While the capability to instantaneously detect and recover from any attempted corruption seems great on paper, in reality, this is hardly feasible. There may always be delays due to various reasons in both the detection and the recovery process. Even among attacks of the same type, some attacks may also be harder to detect than others. Furthermore, the time to perform the recovery may include the recovery of system data and parameters as well as resetting the system data to their original state [CMM+14]. To model this varying delay, we extend the *Monitoring Agent* and the *Recovery Agent* models discussed before. The goal is to replace the instantaneous detection and recovery behaviors with those that require some amount of time to complete.

Figure 8.3a shows the changes made to the *Monitoring Agent* MON. An additional clock mon_clk records the passage of time for MON. Additionally, the parameters `min_detection_time` and `max_detection_time` allows one to select a minimum and



(a) The *Monitoring Agent* Automaton (MON) with Delay



(b) The *Recovery Agent* Automaton (REC) with Delay

Figure 8.3: Detection and Recovery Modeling with Delayed Defensive Actions

maximum time for the detection. Once a corruption attempt is identified in the system, MON will need to wait for a uniform random time between the assigned minimum and maximum detection time, after which a corruption_alert! can be sent. The self-loop in MON allows it to reset the clock after a certain period to avoid running into self *time deadlocks*.

The modification to the *Recovery Agent* REC is shown in Figure 8.3b. Similar to MON, REC is also extended with an additional clock variable rec_clk to keep track of the passage of time for REC. Upon receiving a corruption_alert? message from MON, REC now makes a transition to a Recovering state. The outgoing transition from the Recovering state is paired with the actual recovery action (CORRUPTION_RECOVERY()), and can only be taken between the uniform random delay allowed by the `min_recovery_time` and `max_recovery_time` parameters.

The two additional clocks and the set of four new parameters allow one to control the detection and recovery time after the identification of a vulnerability. In our simulations, we will assume that both the detection and the recovery process take at least one time unit to complete. This will allow us to see the capabilities of a system with a defense that has delays. The defensive action taken in this case is similar to a mitigation mechanism; by the time the detection and recovery process finishes, there may already be an impact (e.g., *time deadlock*) due to the passage of time after the initial attack effect (e.g., a corruption).

8.2.3 Modeling Delayed Defenses with Safe Mode

Finally, we discuss how to model a system with a delayed defense with a safe mode and the underlying reason behind such a design. When allowing multiple different entities to modify a shared variable, there are always chances of encountering a race condition. With the inclusion of the *Recovery Agent* for defensive purposes, there are now three different entities — i.e., a system agent, an attacker, and a defense agent — that can modify a shared system variable. For example, the `status` variable can be assigned by the *Storage Agent* during normal system operations, by the *Attacker Automaton* to cause data tampering, and the *Recovery Agent* to recover from tampering simultaneously. Therefore, the existence of multiple actors leads to a few different problems and escalates quickly when delays are involved.

For one, when we consider delayed defenses (see Section 8.2.2), the detection and appropriate defensive action decisions are made before the delay is applied. This means

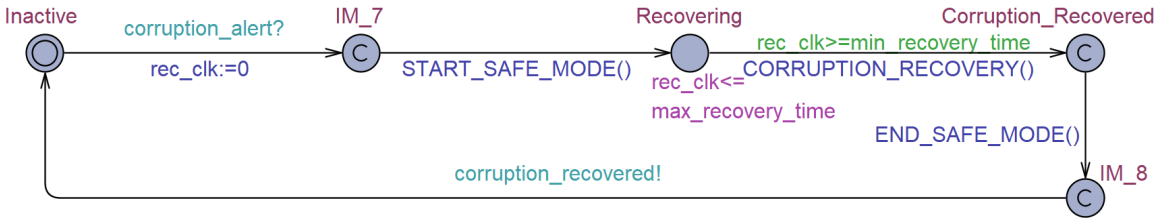


Figure 8.4: The *Recovery Agent* Automaton (REC) with Secure Mode

that if an attacker attacks and causes another compromise (e.g., data corruption) between the time when the recovery decision was made and the actual recovery is done, it can lead to an impact despite the recovery action. Additionally, the system itself may complete a process (e.g., processing), leading to a system assignment of different variables while the detection and recovery are being done. When the recovery finally finishes, the system will be in a completely different state than when the recovery started, and as a result, the recovery may be done incorrectly. In consideration of the various actors involved in the assignment of shared variable values, a delayed defense must take into account the various race conditions and the many different undesirable results that may result from it.

To augment the system model with delayed defenses with a safe mode, we use the same *Monitoring Agent* of Figure 8.3a while extending the capabilities of the *Recovery Agent*. Figure 8.4 shows the *Recovery Agent* with the capability to take delayed defensive actions in a safe mode. The model is extended with two intermediate committed states IM_7 and IM_8 that start (START_SAFE_MODE()) and end (END_SAFE_MODE()) the safe mode, respectively. The safe mode spans the duration of the recovery action.

The goal of the safe mode is twofold; (1) the system may not complete any processes, and thus, may not assign to any shared variables while in the safe mode, and (2) an attacker may not attack the system during the safe mode, and therefore, may not tamper with any shared variable. The first goal is reflective of pausing the ongoing system processes and putting a lock on shared variables until defensive measures are completed. The second goal is reflective of putting the system in a state where attacks cannot happen (e.g., by closing certain ports to prevent remote connections from external entities). While both of these activities are likely to slow down the system processes, thus extending the time needed to reach the system mission objectives, they will allow the recovery to be done correctly without the possibility of being affected by the system process assignments or compromises by an attacker.

8.2.4 Verification with Modeled Defenses

Any time a new agent (e.g., a defense agent) is added to a system, it is important to recheck the system invariants (specified in Section 4.3) in the *Invariant Specification* activity. This is to ensure that the system still retains its former capabilities, and the addition of new functions does not prevent or inhibit it in any way from reaching its mission objectives. For the MCCS, the addition of the *Monitoring Agent* and the *Recovery Agent*, therefore, should not prevent it from reaching its mission objective of producing M number of materials. By rechecking the system invariants in UPPAAL, we confirm that *all system invariants are still satisfied* even after the addition of the two defense agents, allowing us to proceed to the next stages of our impact analysis approach.

Overall, the defense models presented in this chapter are reusable for any type of system as long as the detection function (i.e., IS_CORRUPTED()) and the recovery function (i.e., CORRUPTION_RECOVERY()) are adapted to meet the needs of the system in question. The base defense model of Figure 8.2 is extensible, as shown by the modeling of different defense capabilities with the addition of clocks, parameters, and additional states.

8.3 Impact Analysis with Different Defenses

After equipping the system model with different defenses, we can proceed to execute attacks on various system defense configurations. We will follow the same impact analysis approach activities outlined in Chapter 3, albeit with a different system model augmented with various defenses. In this section, we execute data corruption attacks in the *Model Execution* activity by *random*, *relentless*, and *informed* attackers on the different MCCS system model configurations. Additionally, we analyze the impact of attacks in the *SMC Results Interpretation* activity to assess the effectiveness of the different types of defenses. As mentioned before, we will demonstrate our approach using a system defense that can detect and recover from data corruption attacks. In other words, we will consider a "recovery" mechanism for the three defense types discussed in Section 8.2.

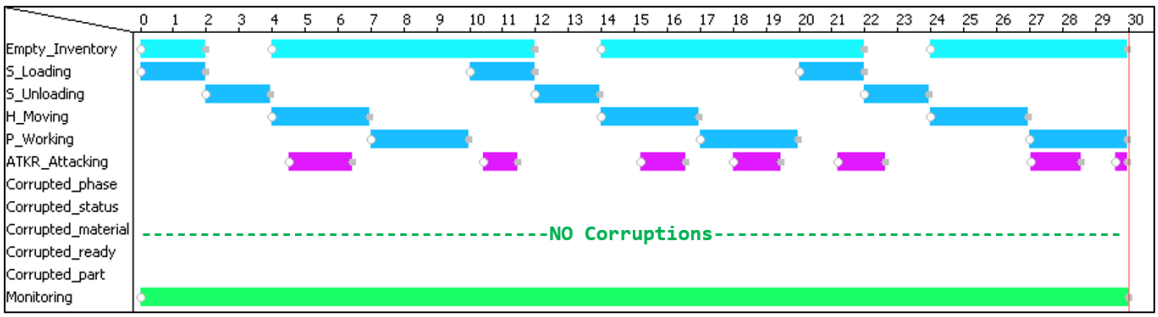


Figure 8.5: Effect of instant recovery against data corruption by *random* attackers

8.3.1 Instant Recovery

We start by looking at the MCCS system model equipped with an instant recovery mechanism (refer to Section 8.2.1). Figure 8.5 shows an example simulation run of a *random* attacker attempting to corrupt the data within the system. The *Monitoring Agent* MON can be seen to constantly monitor the system (depicted in green) for any data corruption. Any attempted corruption by the attacker was immediately found, and the *Recovery Agent* REC was alerted, which recovered from the corruption immediately. This is an ideal case scenario where both detection and recovery are instantaneous and can be seen by the system remaining free of any corruption even after multiple attempts by the attacker. While not exactly prevention (since the recovery is done on the already corrupted data), this is a close equivalent (for harder to prevent attacks) since the effects of the attacks are still prevented from impacting the system.

Two example simulation runs showing data corruption attacks by *relentless* and *informed* attackers are presented in Figure 8.6 and Figure 8.7, respectively. Even after the incessant attacks by *relentless* attackers, and the targeted attacks by *informed*

Figure 8.6: Effect of instant recovery against data corruption by *relentless* attackers

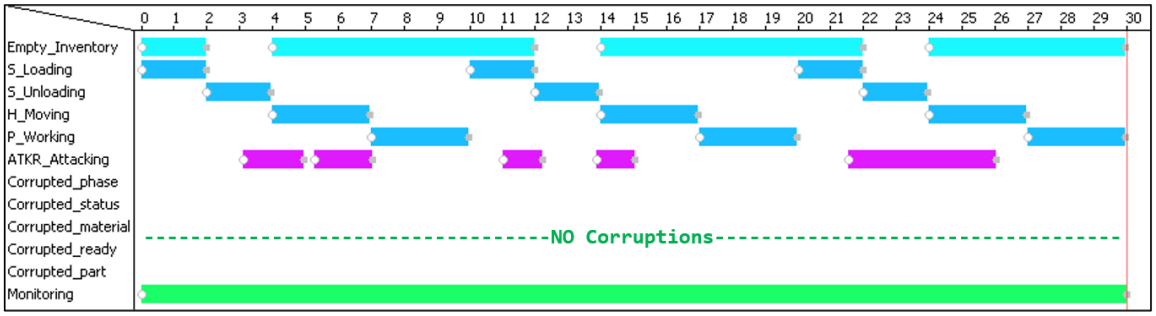


Figure 8.7: Effect of instant recovery against data corruption by *informed* attackers

attackers, they still failed to corrupt any data within the system due to the instant defense capabilities. The MCCS ended up producing all three material units within the allowed 30 time units in case of all the discussed attackers.

While the seamless monitoring and recovery by the system with instant defenses look omnipotent on paper, that is not always the case. This can be seen from the SMC query results presented in Table 8.1 for all the attackers in the system. Despite the instantaneous actions by the system defenses, there is only about 60% and 50% probability, respectively, of reaching daily production goals in time for *random* and *relentless* attackers. The probability is, however, significantly higher (above 99%) in the case of Informed attackers.

Table 8.1: Impact analysis results for a system with an instant recovery mechanism against data corruption by different attackers

Attacker Type	Probability of producing 864 material units (%)		Average units of material produced (number of materials)		Average total production delay (minutes)		Production delay (%)	
	within 1 d	within 23 h	within 1 h	within 1 d	within 1 h	within 1 d	within 1 h	within 1 d
No Attacker	100	0	36	864	0	0	N/A	N/A
Random Attacker	59.1	0	35.99	863.6	0.06	3.02	0.10	0.21
Relentless Attacker	51.2	0	35.99	851.43	0.39	42.72	0.65	2.97
Informed Attacker	99.6	0	36	863.43	0.00	0.55	0.00	0.04

The same trend is seen in the case of material production; hourly production of materials was similar for all attackers, but the daily production results had a noticeable difference as the simulation time increased. While the resulting daily production for *random* and *informed* attackers were similar (about 863 units of material on average), the system under attack by the *relentless* attacker produced significantly less (about 851 units). The delays caused by *random* and *informed* attackers were, once again, negligible. Delays caused by *relentless* attackers were, however, higher with about 3% of the total simulated time.

The two things that stand out in the results are the inability to reliably reach timely mission objectives and the higher degree of impact caused by *relentless* attackers. Even after instant defenses, timely production of materials is still not possible, and the results indicate that as simulations times are increased. By studying various simulation traces, we identify that the instant defenses capabilities provided by MON and REC can only be thwarted by a very specific sequence of actions[1]. If a system process

[1]This is a case of UPPAAL not being able to model "true concurrency", i.e., if more than one possible transition can be taken at the exact same time, one of them needs to go before the other and can be thought of as a modeling artifact.

completion (e.g., moving) overlaps with the completion of an attack, the system may take action after the detection but before the recovery action can finish. This sequence of events results in the system moving to a different phase (e.g., preparation), changing shared variable values which differ from values used to make the recovery decision by the detection mechanism beforehand. Thus, the system taking action between the detection and recovery process confuses the system defense and may result in unpredictable circumstances. This situation is reflective of the idea that any system parameters may change during defensive operations, which may jeopardize the defensive operations themselves and/or invalidate their outcomes.

The chances of this rare but possible overlap happening is higher in case of *random* attackers than in the case of *informed* attackers, since *informed* attackers can choose to not attack whereas *random* attackers will always attack randomly within its bound, resulting in more frequent attacks. The chances are the highest in case of *relentless* attackers with their very frequent barrage of attacks, also evident in the results of Table 8.1. Overall, while the system with instant defenses seem impervious to most attacks, there are loopholes that can be exploited by intelligent adversaries to bypass such defense capabilities.

8.3.2 Delayed Recovery

Next, we look at data corruption attacks on a more realistic MCCS system model equipped with a detection and recovery mechanism with some delays. In all simulations, we assume both detection and recovery actions to take a uniform random time up to one time unit to perform respective actions.

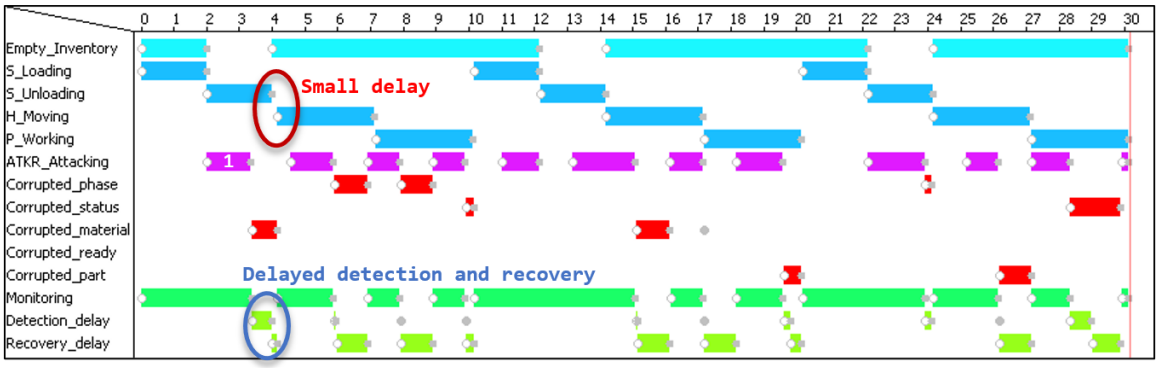


Figure 8.8: Effect of delayed recovery against data corruption by *random* attackers

Figure 8.8 shows an example simulation run of a *random* attacker attempting corruption of system data. We can see the first attempt of the attacker resulting in a corruption of data slightly after three time units. The corruption was present in the system for a short while as the detection and recovery mechanism with delays required some amount of time to complete. The system, however, completed a system process (unloading) before the recovery could be finished. This led to the corruption impacting the system processes as the moving action was delayed for a short time. The system resumed its moving action after the corruption was recovered and all three material units were produced, albeit taking slightly longer than 30 time units.

Figure 8.9 shows a more consequential scenario due to the *relentless* attacker exploiting the delays in defenses. Slightly after five time units, the continuous attacks by the *relentless* attacker resulted in the third corruption in the system. As the system defenses detected and started recovering from the problem, another attack caused the fourth corruption. By the time the system defenses managed to recover from the previous corruption, the MCCS finished its moving process. The MCCS operations were impacted by both the newer corrupted data as well as the system phase change

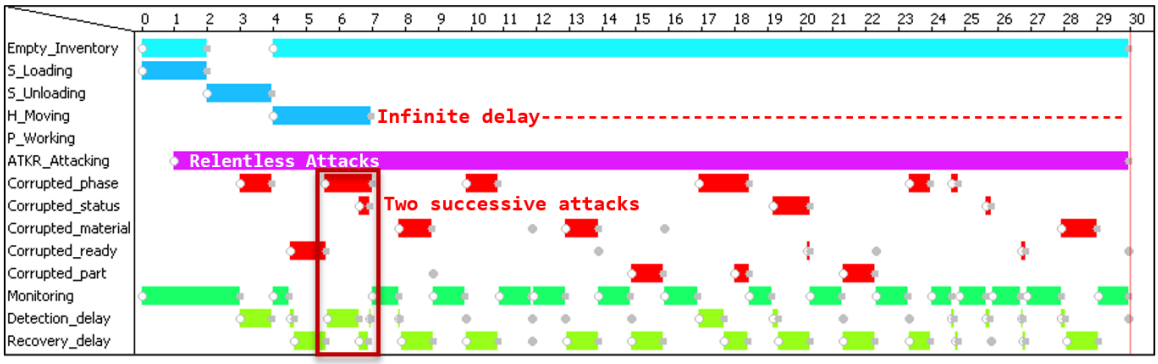


Figure 8.9: Effect of delayed recovery against data corruption by *relentless* attackers

(phase set to 3), going into a *time deadlock* and resulting in an infinite delay. In other words, if successive attacks are fast enough, they can be done in between defensive actions. This can cause the system defenses to consistently be busy with recovering from attacks (as seen from the many different corruptions and recovery in the figure) and present a higher chance for the system to be forced into an undesirable state.

In the example simulation of Figure 8.10, we can see data corruption attempts by *informed* attackers resulting in both small (attempt 2 and 5) and extended delays (attempt 9). The *informed* attacker is also seen to behave sometimes as *relentless* attackers (attempts 4 and 5) and other times as *random* attackers. The frequency of attacks was still lower than that of *random* attackers, resulting in a smaller number of corruptions, and consequently, a smaller number of defensive actions taken.

The possibility of the attacker or system taking an action in between the delays of defensive mechanisms increases the chances for the system to be forced into undesirable states. This is reflected in the SMC query results of Table 8.2, as the MCCS can never (0% probability) reach its daily mission objectives in time when attacked by any attacker.

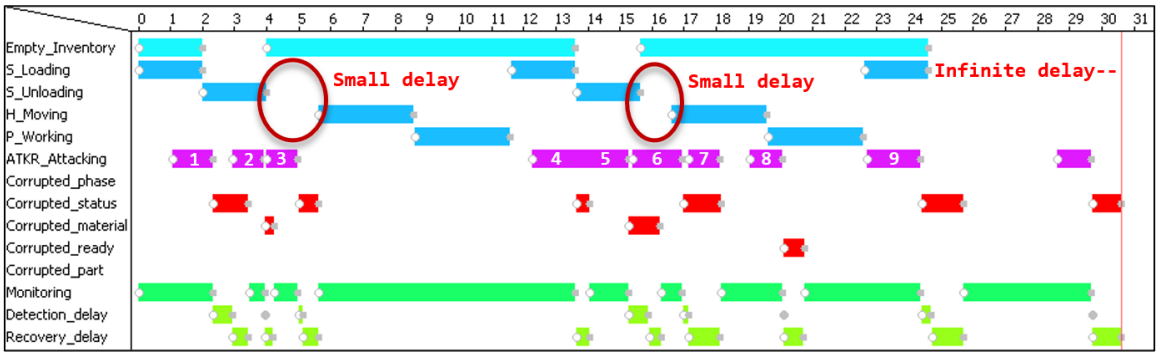


Figure 8.10: Effect of delayed recovery against data corruption by *informed* attackers

The daily production for *random* and *informed* attackers are considerably lowered (about 13 and 22 materials, respectively) compared to the system with instant defenses. The MCCS is also almost always forced into an infinite delay (*time deadlock*) by both attackers as material production gets severely limited (only about 14 and 36 materials, respectively) in the case of daily production. This is also indicated by the extremely high production delays of above 95% for all attackers. The *relentless* attacker is, once again, seen to be the most impactful type of attacker with significantly low hourly and daily material production (less than five material units in both cases) and with an overwhelming majority of the production time (more than 99%) spent in a state of *time deadlock*.

Overall, the more realistic system defense with assumed minimal delays of up to one time unit proved to be much less effective when compared to the system with instant defenses. The primary reason behind this is the other two actors, namely the attacker and the system itself, being able to take actions in between the defensive action, thus leading to various undesirable states.

Table 8.2: Impact analysis results for a system with a delayed recovery mechanism against data corruption by different attackers

Attacker Type	Probability of producing 864 material units (%)		Average units of material produced (number of materials)		Average total production delay (minutes)		Production delay (%)	
	within 1 d	within 23 h	within 1 h	within 1 d	within 1 h	within 1 d	within 1 h	within 1 d
No Attacker	100	0	36	864	0	0	N/A	N/A
Random Attacker	0	0	13.59	14.3	36.50	1413.65	60.84	98.17
Relentless Attacker	0	0	**4.32**	**4.9**	**51.90**	**1432.89**	**86.50**	**99.51**
Informed Attacker	0	0	22.43	36.71	20.19	1381.69	33.64	95.95

8.3.3 Delayed Recovery with Safe Mode

Finally, we take a look at data corruption attacks on the MCCS model with delayed recovery capabilities in safe mode. The safe mode prevents the system agents and the attacker from taking any action during the recovery process. Our goal is to see whether this attempt to prevent race conditions leads to other drawbacks in the system operations and whether the advantages outweigh the drawbacks.

Figure 8.11 shows an example simulation run of a *random* attacker targeting corruption of system data where the attacker can never attack during the recovery process. The first attempt of the attacker leads to a corruption slightly after three time units which result in a small delay before the data recovery is done in the safe mode. The drawback of stopping the system processes, however, can be seen from the second attempt by the attacker. The safe mode pauses the system processes during the recovery, and as a result, the moving process takes an extended amount of time (more than five time units) to complete. Similar extensions can be seen in the case of processing (about four time units) in the same production cycle and in the case of the

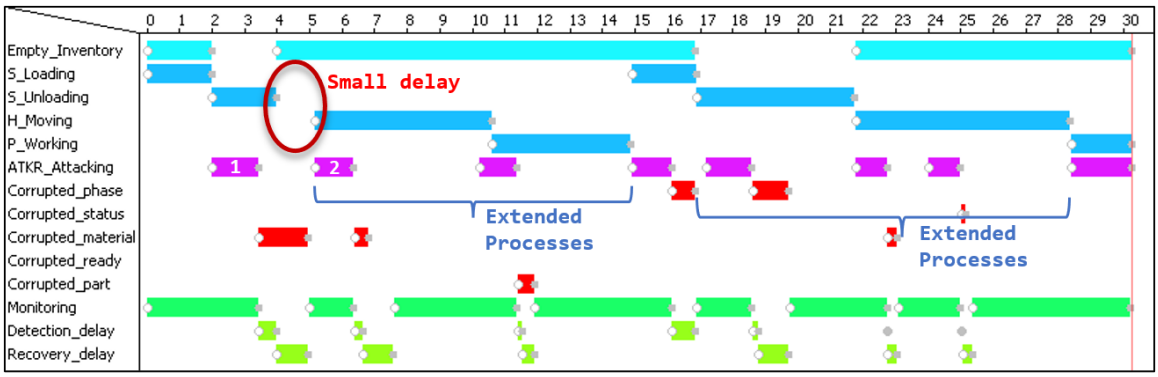


Figure 8.11: Effect of recovery in secure mode against data corruption by *random* attackers

unloading and moving processes (combined duration of about 11 time units) in the second production cycle. The MCCS could not even produce two units of material within 30 time units due to both delays due to corruption as well as the extended processes caused by the safe recovery action.

The drawbacks of the extended system processes are more pronounced in the case of simulations with *relentless* attackers, as shown in Figure 8.12. After every recovery action in the safe mode, the paused system process is restarted, and the process must again take the specified time for it to complete (e.g., 2 time units for unloading). However, because the *relentless* attacker can attack very frequently, it can almost indefinitely extend the process completion by forcing frequent recovery actions for longer processes. This behavior can be seen by the extremely extended unloading (more than 13 time units) and moving processes (more than 10 time units) in the first production cycle, as the MCCS fails to produce a single unit of material within 30 time units.

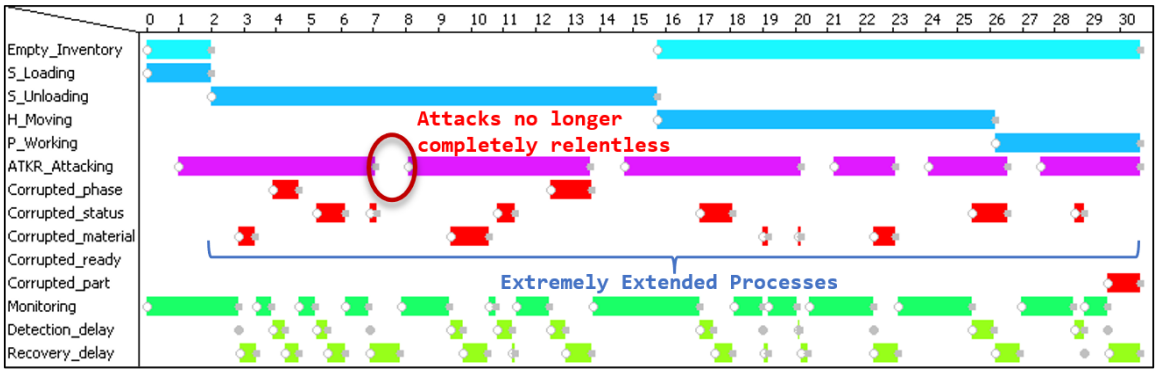


Figure 8.12: Effect of recovery in secure mode against data corruption by *relentless* attackers

Each time the delayed recovery is done in a safe mode, the *relentless* attacker needs to identify that the system is vulnerable again to start attacking. This can be seen by the *relentless* attacker no longer being able to perform incessant attacks as it must wait for the system to become vulnerable out of the safe mode. Even then, the high frequency of attacks still forces many recovery actions thus preventing the system actions from progressing. This behavior indicates that a higher frequency of attacks and/or a lower rate of recovery in safe mode may make the system processes pause indefinitely.

The impact of data corruption attacks caused by *informed* attackers can be seen in the simulation run of Figure 8.13. While the attack frequency is less than that of *random* attackers, each attack results in either causing a delay due to corrupted data or extending the system processes due to delayed recovery in the safe mode.

Table 8.3 presents the impact analysis results from SMC queries for the MCCS with a safe recovery mode. Similar to the case of delayed defenses, the MCCS can never produce (0% probability for all three attacker types) the daily required materials within

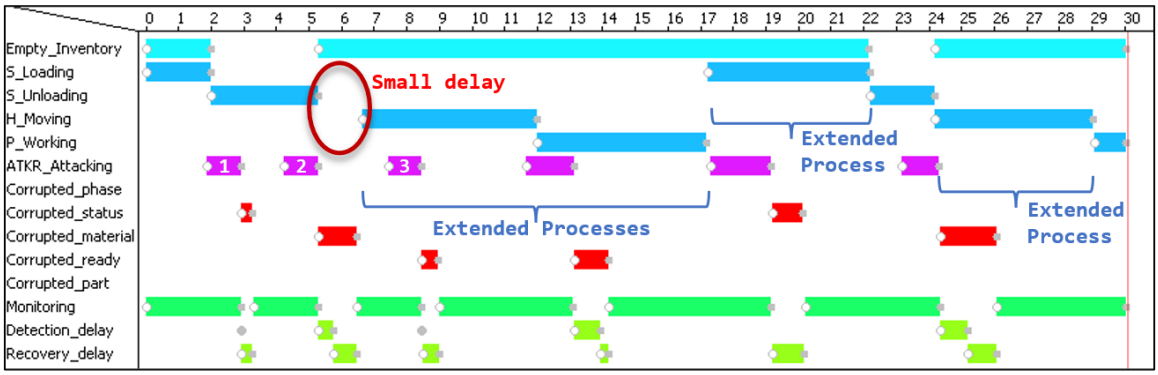


Figure 8.13: Effect of recovery in secure mode against data corruption by *informed* attackers

the allocated time. There is however a significant improvement in the case of the number of produced materials. Compared to the very low production of about 14.3 (2%) and 36.71 (4%) units of material for *random* and *informed* atatckers, respectively, in the case of the delayed defenses, the MCCS with the safe recovery mechanism produces a noticeably higher 475.61 (55%) and 835.86 (97%) units of material. On the other hand, the very frequent attacks by *relentless* attackers take advantage of the safe mode by forcing them again and again, causing no materials to be produced on average for the hourly case and a miniscule average production of 0.003 (close to 0%) units of material in the daily case. The number of materials that could not be produced, however, is not primarily due to system delays (almost no delays in case of all attackers) rather due to the interruption in the system processes caused by the safe mode itself.

In summary, compared to the delayed defense, the delayed defense in safe mode allows more materials to be produced when attacks are not highly frequent or when the system processes are short enough for the system to not be paused indefinitely. The experimental results in this section are reflective of how the characteristics of

Table 8.3: Impact analysis results for a system with a secure mode recovery mechanism against data corruption by different attackers

Attacker Type	Probability of producing 864 material units (%)		Average units of material produced (number of materials)		Average total production delay (minutes)		Production delay (%)	
	within 1 d	within 23 h	within 1 h	within 1 d	within 1 h	within 1 d	within 1 h	within 1 d
No Attacker	100	0	36	864	0	0	N/A	N/A
Random Attacker	0	0	19.59	475.61	0.00	7.37	0.00	0.51
Relentless Attacker	0	0	**0**	**0.003**	**0.07**	**1.61**	**0.11**	**0.11**
Informed Attacker	0	0	34.46	835.86	0.00	2.41	0.00	0.17

a defensive mechanism can prove to be a bad match-up or be targeted by certain types of attackers. Adding poorly designed system defenses that are vulnerable to a certain type of attack or attacker may even compound the impact of an attack. This is shown by the delayed defense in secure mode being abused by the frequent attacks of the *relentless* attacker. The experiments also allowed us to see how the activation of defensive mechanisms over and over again can lead to accumulating delays over time that can affect timely mission objective requirements. To prevent such undesirable outcomes, more comprehensive defensive actions, such as recovery with patching, should be considered.

8.4 Ranking System Defenses

Similar to ranking attacks and attackers based on the SMC query results, defining an impact ranking model for system defenses is non-trivial. In this section, we discuss the challenges in ranking system defenses based on impact.

When ranking system defenses, we have to take into account the feasibility of implementation. For example, the instant defense mechanism may seem to be the most effective defense as it is capable of preventing the attack effect from causing an impact on the system, essentially preventing the impact. Such a defensive mechanism might seem attractive in theory, however, designing a system that would be able to instantly detect as well as instantly perform a defensive action, e.g., to recover from the effect of attacks, would be difficult or impossible in reality. In comparison, the delayed defense and the delayed defense in safe mode seem more practical with delays in both the detection and the defensive action. While the delayed defense in safe mode seems to be more effective as the MCCS ends up producing more materials in the long run, it is also more heavily impacted by *relentless* attackers that can force it to halt longer system tasks enough to completely halt the system progress. Therefore, to determine the most effective system defense, all the common attacks to the system have to be considered, and how the interplay among the implemented system defenses and such attacks or different attacks.

8.5 Limitations of Defense Modeling

During our experiments with different system models with defenses, we came across a few different limitations and questions. For example, even in the case of the instant defense, the MCCS was not able to produce all materials within the allocated time. This means that it is still not possible to prevent the impact of all attacks as attacks can be done at the exact same time when system processes finish and even the instant defensive action can be outsmarted. While it is possible to model defenses where the

defensive action always takes priority, questions remain as to whether this will always be the case in the case of real systems.

Extending the defense models for other types of attacks also quickly gets complicated. The same two defense agents can be used to recover from data modification as long as the IS_CORRUPTED() and CORRUPTION_RECOVERY() functions are replaced by IS_MODIFIED() and MODIFICATION_RECOVERY(). The automata could also be extended to dynamically switch between corruption and modification recovery based on the type of attack detected. However, for data modification attacks, the same reliance on process invariants no longer works. That is because there are no guarantees that the shared variables have not been modified between the time when the detection is done and the recovery is complete. Additionally, depending on the order of checking for detection, the wrong variable could be detected to be modified. Trying to strengthen the detection process by relying on the correct state of a combination of multiple variables may also not work since a single variable modification can distort the decisions made. In a way, attacks that target the system parameters (e.g., data modification) on which the defense decisions rely can be seen as a way to target and compromise the operation of the defensive mechanisms themselves. All of these could result in the addition of system defenses being a liability, producing even fewer materials compared to defenseless systems when under attack from an attacker.

Up until now, we have discussed detecting and recovering from data tampering attacks by using various defensive means. However, we must also keep in mind that we are only allowed to do so since the data tampering is not impactful until the tampered data is actually used in a system process, and thus, we are allowed to have some

time to take defensive actions. Questions arise in the case of attacks for which the impact is instantaneous, such as how we should detect and recover from those attacks. For example, the effect of spoofing attacks (premature actions) are almost always instantaneous (see Section 7.3.3) and may lead to fast impacts (system malfunctions). How to recover from these situations without doing a full system reset, thus affecting the entire system processes, still remains to be explored.

In other words, modeling defenses is a challenging task, especially when time is involved in the already complicated equation of different ICS processes and different attacker behaviors. In this work, we showed a few examples of how different types of defenses can be modeled and presented some results from our preliminary analyses. However, to cement the understanding of the interplay between different system processes, attacker actions, and defensive measures, a more structured and comprehensive investigation is in order.

8.6 Related Work on Defense Modeling

To the best of our knowledge, a comprehensive study encompassing the many different suggested system defense or defender models for ICS or CPS does not exist. Most of the work on modeling defenders in the literature is based on game-theoretic approaches. In [RPM+16], the authors present a game-theoretic attacker-defender model with a system modeled as discrete systems of cyber and physical components. The goal of the defender is to reinforce the infrastructure parts or components to defend against incidental degradation of the system or service interruptions by an attack with

information on the incidental degradation, costs incurred to defend, and strategies to decide which part to reinforce. In [HCSB16], the authors proposed a multi-defender model, where a CPS is protected by a group of defenders in a static game. Each defender is responsible for one or more CPS components and shares management of the nodes at the edge of their areas of responsibility. Hasan et al. [HGD+18, HDKK20] presented a game-theoretic approach to attacker-defender modeling in smart grids. The goal of the formal dynamic defense model, given a defense budget, is to minimize the damage caused by cyberattacks by identifying the critical subset of substations based on a polynomial-time algorithm. Yuan et al. [YZZ14] presented an efficient algorithm to solve the defender-attacker-defender problem for practical scenarios in power grids to improve grid survivability. Fielder et al. [FLH16] proposed a simulation of attackers and defenders where the objective of the defender is to identify the appropriate deployment of a specific defensive strategy such as defense-in-depth. The study highlights how the best use of available resources and effort is to apply the defense to the most valuable nodes when there are critical components or few valuable assets in the system. A stochastic game model for cross-layer security decision-making in Industrial CPS is presented in [HZQT19].

In contrast to modeling defense strategies, our work looks at various defensive capabilities that differ based on time (i.e., how fast is the defense?) and what is done during that time. Similar to attacker modeling, the goal is to use the existing knowledge to model system defenses applicable to ICS and see how the existence of such defenses affects the impact caused by various attacks and attacker behaviors. The results from such an analysis may allow us to identify, for example, the best overall

defense or the best defense against certain types of attacks or attackers in mitigating impact, informing the development of secure ICS.

8.7 Conclusions

The modeling of different types of system defenses extends the applicability of our impact analysis approach to analyze the impact on systems with defenses to compare it against the impact on defenseless systems. In this chapter, we modeled three different kinds of defenses, namely instant defenses, defense with delays, and defense in secure mode using two additional system agents. We explored the capability of different system defense models to fend against data corruption attacks by *random*, *relentless*, and *informed*. We studied system simulation traces to observe the behavior of various defensive capabilities and measured the impact of attacks on such systems using statistical results from SMC queries. Overall, this chapter demonstrated the applicability of our impact analysis approach by covering the *System Model Definition* and *Model Verification* activities from the System Modeling phase, *Model Execution* activity from the Attack Execution stage, and the *SMC Results Interpretation* activity from the Impact Analysis stage on systems with different defensive capabilities.

At the end of this chapter, we are left with:

1. *Three different types of system defenses, each modeled as a set of two timed automata*

2. *An understanding of how the existence of system defenses can mitigate cyber-attack impact compared to defenseless systems*

3. *An overview of the specific activities that need to be repeated to obtain new impact analysis results*

Chapter 9

Assessment of the Approach

The results presented by the proposed impact analysis approach so far allowed us to observe and quantify the impact on ICS operations in the presence of attacks by different attackers and various system defenses. In this chapter, we assess our contributions by discussing the strengths and limitations of the various aspects of the proposed approach and how they satisfy the research objectives outlined at the outset of this study (see Section 1.4). Sections 9.1 through 9.4, respectively, examine the advantages and drawbacks of the required prior information, modeling choices, analysis approach, and the tool selected for this study. Section 9.7 outlines the importance of modeling time and examines the sensitivity of various attacker model parameters. Lastly, Section 9.8 concludes our assessment.

9.1 Prior Requirements

In this section, we discuss the strengths and limitations associated with the information required prior to starting our impact analysis approach.

The proposed impact analysis approach is meant to be applied at the early stages of the system development lifecycle and requires information on different fronts to get started. First, to model the system and define system invariants, we need access to the system design descriptions and requirements specifications. Furthermore, to model different attackers and assign attacker capabilities effective against the system, we require information on common attacks against the system, which can be obtained by performing a threat analysis or from an existing threat model of the system.

The advantage of such requirements is that different activities of the approach can begin very early, which are modular in nature. For example, as soon as we have the system design description, we can start the *System Modeling* stage (see Chapter 4), and can finish all activities in it when we get access to the requirements specifications. Both of these documents are usually available early in the system development lifecycle, and as such, they correspond well to the early stages of the approach. Once we have information on common attacks on the system, e.g., through historical data or a threat model of the system, we can complete the *Attacker Modeling* stage (see Chapter 5). From that point onwards, the rest of the approach is mostly automated and does not have any specific requirements.

The approach can, therefore, start very early along with the early stage of system development. Continuous iterations of the approach throughout various stages of

system development can provide data on impacted system operations so that judicious design choices and/or changes can be made as the cost to do so in the early system development stages is the lowest [BK08]. The modularity of the approach is useful for dealing with changes throughout system development; if there is a change in the required information, adjustments are only required in the corresponding activity (e.g., changes in the requirements specifications will only require revisions in the *Invariant Specification* activity). Furthermore, the impact analysis results can lead to much-needed awareness among system developers and operators, encouraging revisions in the system design to include proper security controls. This undertaking would result in a final system deployment with "built-in" security, thus leading to improved system quality and reliability.

On the other hand, the start of the approach is limited primarily by the same information (design description, requirement specification, threat model) that may not be readily available and the effort required to build the system and the attacker models. A method to automatically generate timed automata models does not exist, and as such, for large ICSs, it becomes a largely manual and time-consuming practice [JJ21a]. Additionally, changes to the required documents would also require changes to the system and/or attacker models, and parts of the approach may have to be repeated.

9.2 Modeling Considerations

In this section, we assess the selection of timed automata as our modeling formalism of choice and the advantages and disadvantages associated with it.

When modeling systems, our impact analysis approach is applicable to ICSs of varying sizes. The flexibility of modeling systems at different granularities afforded through the use of a general-purpose modeling technique such as timed automata is critical. Assessing only the overall system state is insufficient to ensure system reliability, as tampered component actions may result in invalid products (e.g., altered concentration of the final product in a chemical processing system) [JJ21b]. This flexibility is what allowed us to identify the overlapping action, and consequently, the case of hastened production in Section 6.4, a feature missing from many traditional testing or simulation methods.

In our approach, the choice of granularity in modeling the system is left to the modeler. For example, for a smaller system requiring higher security assurance, each component, and even their tiniest actions can be modeled. In contrast, for larger systems, salient components may be grouped into agents, and only the primary actions may be chosen for modeling. For even larger systems (e.g., Systems-of-Systems [GD14]), each system can be modeled as a black box to see the impact of cascading events through systems. Impact can also be considered first on the overall system, then on the grouped agents, and finally on the agent components, if necessary. Overall, this leads to modeling activities that can deal with any granularity present in the system, and satisfies our research objective **RO4**. by providing the *necessary flexibility when modeling and analyzing systems with different granularity.*

The work is also heavily focused on clarity instead of trying to demonstrate the efficacy of the approach on a complicated system. While small, the illustrative MCCS

contains the necessary internal component communication, dependencies, and time-constrained actions commonly found in almost all ICS [JJ21a]. It is important to understand that we do not necessarily need to model every component of the target system. What is crucial is identifying the vital system processes that we are concerned about (e.g., loading, unloading, handling, and moving for the MCCS) and modeling all entities involved in such a process as an agent. An example of this is the *Handling Agent* abstraction, where a set of components (e.g., robotic arms, conveyor belts, etc.) may work together to constitute the moving process. For larger industrial systems, agents can be grouped together [JV17a], or we can prioritize and analyze a single process at a time. The MCCS represents a typical material manufacturing process, similar to the scale of an industrial mixing process [LXDW16] or a real-world wastewater dechlorination process [Jas20a], for which other security-related analyses have been performed.

Since the illustrative MCCS has a small number of automata when components are grouped into agents, at this point in our research, we have not yet faced the issue of state space explosion. However, modeling large systems with any state-based modeling language such as timed automata may cause the state space to explode. Challenges with this issue can be reasonably dealt with by grouping components into agents [KFM+11], using partial order reduction [BK08], or adopting alternative evaluation approaches [SDF17], among others. The approach has not yet been tested on real-world complex systems, which is, however, a prime candidate for our future work.

9.3 The Analysis Approach

In this section, we explore the capabilities and shortcomings of the proposed impact analysis approach.

The proposed approach relies, in various stages of the analysis, on distinct model checking activities. First, it relies on classical model checking to verify whether the system model is performing its intended functions to reach its mission objectives in the *System Modeling* stage (see Chapter 4). The same system invariants are revisited at the end of the *Attack Execution* stage (see Chapter 6) to check whether the existence of an attacker on the system can affect the system mission objectives, i.e., to identify the existence of an impact. On the other hand, Statistical Model Checking (SMC) queries are used throughout the *Impact Analysis* stage (see Chapter 7) to analyze and quantify the impact on the mission objectives.

The use of different forms of model checking allows us to quickly identify and/or quantify the impact of attacks made by different attackers on system operations. The analysis also illustrates how manipulating state variables or communication messages can prove to be highly impactful for a defenseless system. Our timed analysis approach highlights the case of delayed actions due to attacks and results from SMC allows us to reinforce our understanding with concrete data over numerous simulation runs. Additionally, it allows us to determine whether the presence of system defenses leads to impact prevention or mitigation and assess the degree of mitigation of the impact.

Rather than trying to be exhaustive, our approach specifies a small set of system invariants and SMC queries to demonstrate their effectiveness at analyzing the impact on ICS mission objectives. All system invariants and SMC queries specified in this work are domain-agnostic in nature as the reusability of the approach for other ICS was considered the highest priority during their definition. This, in essence, satisfies another one of our research objectives (**RO3**.) by *defining impact metrics that are domain-agnostic and reusable* for all ICS.

The analysis approach also allowed us to see the impact on more than one ICS security objective. For example, tampering with sensor values to corrupt or modify data is an *integrity* issue. However, we demonstrated how data tampering attacks could lead to temporary or extended delays in system operations, which relates to *availability*. This example shows the interplay between the cyber and the physical infrastructure in ICS where attacks aimed at the *integrity* of the cyberinfrastructure can end up compromising the *availability* of the physical infrastructure.

Similarly, in the case of spoofing attacks, the attacker first has to obtain information about licit control messages (a *confidentiality* issue) and then spoof such messages to impersonate a system component (an *integrity* issue) targeting the cyberinfrastructure. These attacks targeted at the cyberinfrastructure, however, can also lead to *availability* issues for the physical infrastructure through a complete system malfunction. Additionally, they may lead to a deceptive case of hastened production where the physical processes are available, performing their typical functions, and in the correct sequence, but the timing of the physical processes is not synchronized (a special case of impact on *availability*). Therefore, throughout the proposed approach,

we have attained our research objective **RO2**. by *considering the impact on different security objectives pertaining to ICS*.

The efficacy of using invariant violations to detect attacks for further analysis, however, relies on the completeness of the set of system invariants. For complex systems involving many intricate interactions, proving that the current set of invariants is exhaustive is non-trivial and may require domain expertise. Caution must be exercised, as not all stakeholder requirements but only those that correspond to proper operational behavior should be regarded as system invariants. The security analyst is responsible for identifying the correct definitions of similar system invariants to ensure expected system behavior. Additionally, consultations with system developers may provide valuable inputs to determine specific system functionalities that must be preserved and assist in recognizing some of the more subtle specification requirements that are not obvious at first glance.

9.4 Discussion on the Selected Tool Support: UPPAAL

The use of the modeling tool Uppaal [BDL04] and the provided extensions to timed automata enables us to automate the later activities of our impact analysis approach. The SMC capabilities of Uppaal also allow us to scrutinize the real-time behavior of the system under attack by analyzing the concrete simulation traces to study patterns in the attacker's behavior or identify vulnerable moments in the system operation. By investigating the traces of tampering attacks, for example, we discovered that

the attacker exploits the time taken to perform a particular action after a state variable has been assigned and before it is read. This revelation indicates that in a system where component actions are lengthy and result in larger time constraints, the attacker will have more opportunities to succeed in its attacks. This leads us to believe that small, quicker actions and faster task scheduling may be adopted during the design phase when building newer ICS to make its processes inherently resilient to attacks.

The possibility of being affected by a shared-variable compromise is also heavily influenced by how dependent other system components and operations are on the compromised variable and how many times it is read during each production cycle. The highly connected components within the system (e.g., the *Control Agent*), therefore, represent areas where we should prioritize stringent security controls (e.g., investing in tamper-resistant sensors or actuators) to ensure their integrity.

There are, however, certain limitations in modeling ICS behavior using Uppaal. We note that our observations in this thesis are impacted by the environment in which we simulated, especially by the various timing constraints assigned to the models. At the time of writing the thesis, a method to automatically generate timed automata models in Uppaal from a system specification does not exist [JJ21a]. While the graphical editor of Uppaal is helpful, it still requires an understanding of the semantics of timed automata, along with the extension of timed automata provided by Uppaal, for a proper model. Uppaal does not allow the assignment of floating-point values, and thus, the conversion from floating-point values to the nearest integer in Uppaal may lead to loss of accuracy and missed or incorrect detection of attacks [SM17].

Additionally, in ICSs where component tasks can take anywhere from milliseconds to hours to finish, using time units to represent the lowest unit of time may lead to long simulation times and slower processing of queries.

9.5 Summary of Assumptions

Throughout our impact analysis approach, we make several assumptions about the modeled system, attackers, and defenses. In this section, we provide a brief discussion of the assumptions and the underlying reasons.

The information required for modeling is available. The approach relies on several required information, such as the system design description and requirement specification documents, respectively, to model the target system as a network of timed automata and specify the system invariants for the system model. The assumption is that such information is available and they are adequate so that the *System Model Definition* and *Invariant Specification* activities can be completed. This information can be expected to be available before starting the approach since the approach is designed to be used at the early stages of system development. Similarly, specification of the attack capabilities of the attacker models in the *Attack Specification* activity depends on sufficient information on common attacks or a threat model of the target system of analysis.

System failures are non-existent. After the system model has been built and verified to work as intended, we assume that the system does not face a failure during typical system operations. While system faults are possible and a likely occurrence, our goal is to see whether an attack from an attacker can force the system to fail some of its operations and/or cause a system-wide malfunction. This also allows us to use any deviations from the expected system behavior in the *Invariant Reinspection* and *SMC Results Interpretation* activities to attribute that to the actions of an attacker.

There exists a persistent system vulnerability. We assume that the system has a persistent system vulnerability that can be exploited by all attackers to mount an attack on the system. The focus of our work is to analyze the impact, i.e., how attackers can use such a vulnerability within a system to cause various impacts on the system's mission objectives. Future works may explore the existence of the vulnerability to be determined by some probability distribution, the difficulty of exploiting the vulnerability, and the resources or skill level needed for an attacker to find and/or exploit such a vulnerability.

All attackers cause the same impact upon success. Regardless of the attacker type, we assume that if an attack is successful, it will inevitably lead to an impact on the cyberinfrastructure. For example, even though *informed* attackers may have more information on the system than a *random* or *relentless* attacker, we assume that all of their attacks will lead to a successful alteration of data when performing a data corruption or modification attack. Whether these attacks will lead to an impact on

the physical infrastructure (e.g., a malfunction), however, will depend on whether the attack can be mitigated by the typical system operations or existing system defenses.

Spoofing attacks require the compromise of a system agent. To perform spoofing attacks, an attacker first has to obtain information on licit system messages used for communication. The assumption here is that one or more of the system agents (in our case, the *Storage Agent*) can be compromised by an attacker so that the attacker can impersonate the behavior of such agents by spoofing their messages (loaded! and unloaded! for the *Storage Agent*) and sending them outwards to other system agents.

The system mission objectives are known. We assume that the mission objectives are known for the target system of analysis. For example, since the MCCS is a manufacturing system, its primary objective is *to produce a fixed number of materials*. Other objectives may include that *the production process shall complete within the prescribed time* or that *the materials shall be of expected quality*. This is what allows us to tailor the domain-agnostic system invariants in the *Invariant Specification* or the SMC queries in the *SMC Query Definition* activity to the mission objectives of the MCCS. After the adaptation, more domain or system-specific queries (e.g., to deal with the product quality for the MCCS) can also be specified in those stages to be more comprehensive.

The system defenses are allowed to perform the required defensive activities. We assume that the system defenses have proper access to the system to detect

attacks and perform defensive actions on the system. For example, the detection and data recovery mechanism considered in this work is assumed to have read access to the shared variables to identify any corruptions and write access to be able to recover from an attempted corruption. The delayed defense with safe mode is assumed to have the additional capability of preventing system processes from progressing and taking the system to a safe state where no attacks on the system are possible during the recovery action.

9.6 Threats to Validity

The *validity* of any study is the extent to which its design and conduct mitigate or prevent systematic errors or bias [DD08]. *Threats to validity* or *validity threats* refer to specific ways in which the study might prove to be wrong [FM10]. Threats to validity are always assumed to be present in any empirical research, leading to the goal of trying to mitigate as many known threats as possible [CbO17].

In [CbO17], the authors present a classification of validity threats, separating them into categories of (1) conclusion validity, (2) internal validity, (3) construct validity, and (4) external validity. In this section, we examine the threats to the validity of the impact analysis approach presented in this thesis and discuss how our approach deals with the majority of them.

Threats to conclusion validity. *Conclusion validity* refers to the belief in the ability to derive conclusions from the relationships between the treatment (method or

process) and the outcomes, represented by the independent and dependent variables, respectively [CbO17]. *Threats to conclusion validity* are limitations to the study that affect the ability to derive conclusions about these relations [WRH+12].

Our impact analysis approach mitigates *threat to statistical validity* by having high confidence (95%) in the statistical tests (each with 1000 runs) conducted using the SMC queries, indicating the ability of the results to assert a true pattern. It deals with the *threat of fishing for the result* by generating statistical results by not trying to identify any specific result, but comparing SMC query results with and without the presence of attackers to compare the differences to create interpretations of impact. As the model checking is automated, both classical and statistical model checking provides reliable measurement of independent variables, mitigating *threats to the reliability of measures*.

The implementation of the treatment strictly follows our impact analysis framework and is the same for all ICS, dealing with the *threat to reliability of treatment implementation*. The raw data are collected from the *SMC Results Interpretation* activity of the impact analysis approach. Therefore, problems such as missing data, outliers, and wrong data values are easily understood and corrected, mitigating the *threat to lack of data pre-processing*. However, the approach is limited by the *threat of lack of expert evaluation* as interpreting the SMC query results require having deep knowledge of the target system of analysis and its processes to define what can be considered as impact and/or opinions of experts in the system or security domain.

Threats to internal validity. *Internal validity* refers to the belief that the changes to a dependent variable in a model are solely caused by changes of the independent variable set [CbO17]. *Threats to internal validity* are, therefore, influences that can affect the independent variables with respect to causality [WRH+12]. These are the most significant threats to the validity of the proposed approach.

First, the approach suffers from the *threats to treatment design*; the artifacts used in the treatment, such as the design descriptions and requirement specification documents used to build and verify the system model could affect the results if not adequate, complete, or consistent. Additionally, the approach also has to deal with the *threat of ignoring relevant factors* and *motivation*, as not all relevant factors are considered in the experimental setup. These factors may include the tool usability, the expertise needed to create timed automata system models and/or specify relevant queries, and the user resistance to any formal method based technique. Another consideration is the *threat of deficiency of treatment setup* where variables such as tool performance or limitations are not taken into account, which may impact the results.

Threats to construct validity. *Construct validity* is the belief that the defined independent and dependent variables represent the theoretical concept of the phenomenon being studied accurately [CbO17]. The typical *threats to construct validity* are related to the study design or to social factors [WRH+12].

The approach deals well with the *threat of mono-method bias* by using multiple metrics (e.g., reduced production, delays in operation, etc.) to measure the impact that can be verified by examination of system simulation traces. It also deals with the *threat of*

mono-operation bias by including more than one independent variable (e.g., different attackers, attacks, system defenses, etc.), one framework, and one subject in the study. The data are directly collected from the target system of analysis, thus mitigating the *threats to the appropriateness of data* by avoiding irrelevant and/or inappropriate records resulting from heuristics or keyword searches. The measurement method is automated, and the metrics used to measure impact are objective and solely related to the system's mission objectives. Therefore, during the measurement, the approach deals well with different types of measurement bias (e.g., observer or attention bias), mitigating *threats to measurement metrics* or bias resulting from the human aspect (e.g., years of experience, different perception or understanding), mitigating the *threat of experimenter bias* that may impact the study results.

Threats to external validity. *External validity* refers to the generalization of the results, e.g., the results obtained in a specific setting to be applicable to results of all software or system of that type [CbO17]. *Threats to external validity* are conditions that prevent or limit the ability to generalize the study results.

The study is affected by changes in the target system of analysis but not by changes in time and location: performing the analysis in a different country or a few years later will not change the approach and the outcomes of the study, thus dealing with the *threats to the study context*. It also mitigates the *threat to the representation of the setting*; the tools used represent a real setting that is representative of the study goal (i.e., identifying impact in an ICS) and changes in the setting, e.g., alterations in the system design or emergence of new attackers are easy to address due to the modular nature of the approach. The *threats to population representation*, however,

are not mitigated as the study is done only on a single case study system (the MCCS), which may be representative of manufacturing systems, but not all types of ICS in general.

9.7 Considerations for Time

Time is a critical factor in each phase of our impact analysis approach. In this section, we discuss the effectiveness of the timed impact analysis approach proposed in this work and how sensitive the analysis results are to certain timing parameters.

9.7.1 Assessment of the Timed Approach

The use of timed automata allowed us to model the timed behavior of the system agents and see how attacks made by different attackers or system defense capabilities that vary based on the aspect of time affect such timed behavior. This is shown throughout the modeling phases of our impact analysis approach; (1) the *System Modeling* phase must take into account the different time-constrained system processes, (2) the *Attacker Modeling* phase has various timing considerations based on the type of attackers we are modeling, and (3) timing becomes an important issue when modeling different system defense capabilities and discussing their practical use.

On the other hand, the system invariants used for the identification of attacks in our approach focus on the timely achievement of mission objectives. Similarly, the SMC queries used for the quantification of impact also focus on timed characteristics (e.g.,

probability of reaching the mission objectives within time or the expected duration of interrupted operation). The inclusion of time allowed us to highlight the often ignored case of time-based impacts such as operational delays and the otherwise unseen hastening of processes. In essence, this allowed us to reach our research objective **RO5**. by *taking a timed perspective in analyzing impact.*

9.7.2 Sensitivity of Model parameters

The reliance on time, however, can also lead to certain complications. Throughout this work, we had to make several assumptions about time to demonstrate our approach with different attacks, attackers, and system defenses. The assumptions made for the system processes (e.g., moving) is easily verifiable since security analysts would have the information about the time needed to complete system processes or an objective from the system design descriptions or the requirement specifications. The cases of different system defenses also do not rely much on assumptions on time, rather they focus on different defensive capabilities. Since we want our system defenses to be fast-acting and capable of performing defensive measures to mitigate the impact of attacks, we assume that they are either instantaneous or take a minimal (at least one time unit) to complete. Increasing the delay in either the detection or the defensive action would, thus, inevitably lower the effectiveness of the defensive capabilities.

Of more importance is the timed behavior of the attackers as there are different timing parameters involved. Each of these parameters affect how the attacker behaves in parallel with the system processes, allowing for e.g., more frequent and/or faster

attacks on the system operations. We would like to see how changes to such parameters affect the impact analysis results, i.e., how sensitive are the impact analysis results to attacker parameter changes. The attacker models have three sets of timing parameters:

- `vulnerability_search_time`: Determines the duration of searching for an existing system vulnerability.
- `attack_preparation_time`: Determines the duration of time needed by an attacker to prepare for an attack.
- `attack_time`: Determines the duration of each attack before they can lead to the intended malicious effect on the system.

The default values of the parameters `vulnerability_search_time` (one time unit) and `attack_time` (one to two time units) are same for all three attackers types. On the other hand, the parameter values for `attack_preparation_time` varies for each attacker; the *random* attacker has a random duration (between one to five time units) of `attack_preparation_time`, whereas the *relentless* attacker has no `attack_preparation_time`, and the *informed* attacker dynamically determines how long it should prepare through the information that it has on the target system.

The default values of the parameters were originally assigned in a way where the attacker's attacks would be fast enough to attack during each material production cycle and in between system processes for maximum interaction. However, now we wish to see how variations in the values of these parameters affect our impact analysis

results and whether the results are more sensitive to some parameters than others. For each of the three parameters, we will consider how increasing and decreasing the time constraints affects the results of our impact analysis. For demonstration, we do this for the *random* attacker causing data corruption attacks first on a defenseless system and then on systems with the three different types of defenses discussed in Section 8.2.

Sensitivity of Vulnerability Search

We start by taking a look at the sensitivity of the `vulnerability_search_time` parameter. The default values assigned to our attacker models mandate that an attacker searches for a vulnerability for a fixed duration of one time unit. Table 9.1 shows the variations in the impact analysis results (average number of materials produced daily) when the `vulnerability_search_time` is decreased or increased. The results presented are for a *random* attacker performing data corruption attacks, where the other two attacker parameters retain their defaults values.

We can see an MCCS with no defenses being the most impacted by the attacks, as the severity of the effect increases as we increase the values of the `vulnerability_search_time` parameter. The comparative low sensitivity is due to the fact that the vulnerability search is done only once at the beginning, and as long as the vulnerability exists, an attacker would continue to prepare and attack instead of searching for another vulnerability. Similar trends can be seen for both the system with a delayed defense and the delayed defense in safe mode, albeit the results have a higher sensitivity in the latter case as the safe mode forces the attacker to

Table 9.1: Impact analysis results (average number of materials produced daily) across different system defensive capabilities to determine the sensitivity of the `vulnerability_search_time` parameter of a *random* attacker performing data corruption attacks

Type of Defense	Duration of each vulnerability search (time unit)				
	None	0 to 1	1	1 to 3	3 to 5
No defense	1.83	1.72	2.13	2.24	2.56
Instant defense	864	863.8	863.9	864	863.9
Delayed defense	14.42	14.54	14.74	14.77	14.85
Delayed defense in safe mode	464.37	474.39	475.61	483.2	500.53

search for the vulnerability again (e.g., re-establish a remote connection). The results for the system with the instant defense capabilities are not sensitive to the parameter changes, and all results are consistently very close to the ideal daily production of 864 units of material (deviations are due to the concurrency issue discussed in Section 8.3.1).

Sensitivity of Attack Preparation Duration

Next, we observe how changes in the `attack_preparation_time` parameter affect the impact analysis results (average number of materials produced daily). The default values assigned to a *random* attacker model mandate that an attacker prepares for a duration of one to five time units before mounting an attack, whereas the considerations for the other two attacker types are different. Table 9.2 shows the sensitivity of the impact analysis results when the `vulnerability_search_time` is altered. The results presented are for a *random* attacker performing data corruption attacks, where the other two attacker parameters retain their defaults values.

Table 9.2: Impact analysis results (average number of materials produced daily) across different system defensive capabilities to determine the sensitivity of the `attack_preparation_time` parameter of a *random* attacker performing data corruption attacks

Type of Defense	Duration of each attack preparation (time unit)				
	None	0 to 3	1 to 5	3 to 5	5 to 7
No defense	0.45	1.36	2.13	2.8	4.24
Instant defense	858.88	863.9	863.9	864	864
Delayed defense	4.13	9.785	14.74	18.53	24.9
Delayed defense in Safe mode	**0.005**	330.43	475.61	549.2	628.92

Overall, the impact analysis results are more sensitive to changes in the `attack_preparation_time` parameter as lower values allow the attacker to finish its preparation within less time, and thus, perform more frequent attacks. We, however, also explore a larger range of parameter values as the original range assigned was much broader compared to the `vulnerability_search_time` parameter. Of note is the production of an extremely low number of materials when the attacker is allowed to have an instantaneous preparation capability (no preparation time), essentially mimicking the behavior of the *relentless* attacker. This is because the instantaneous preparation capability, along with the fairly low default duration (one to two time units) of the `attack_time` allows the attacker to perform attacks in quick succession.

The system with a delayed defense in safe mode is affected the most by the parameter variations, restating how more frequent attacks can cause a severe impact (not a single unit of material produced when the attacks are relentless) in these systems. The system with the instant defense capabilities is the least affected, but the previously mentioned concurrency issues are more pronounced when the attacker takes less time to prepare.

Table 9.3: Impact analysis results (average number of materials produced daily) across different system defensive capabilities to determine the sensitivity of the `attack_time` parameter of a *random* attacker performing data corruption attacks

Type of Defense	Duration of each attack (time unit)				
	None	0 to 1	1 to 2	2 to 3	3 to 5
No defense	1.47	1.79	2.13	2.75	3.4
Instant defense	862.77	863.7	863.9	864	864
Delayed defense	10.05	10.63	14.74	15.31	23
Delayed defense in safe mode	318.37	373.82	475.61	552.95	618.44

Sensitivity of Attack Duration

Finally, we examine the sensitivity of the `attack_time` parameter. The default values assigned to our attacker models mandate that an attacker requires a duration of one to two time units to perform each attack. Table 9.3 shows the variations in the impact analysis results (average number of materials produced daily) when the `attack_time` is decreased or increased. The results presented are for a *random* attacker performing data corruption attacks, where the other two attacker parameters retain their defaults values.

Overall, the results follow the same general trend, where allowing lower values to the `attack_time` parameter results in higher impact since each attack takes less time to complete. The sensitivity of the results is higher than that of the `vulnerability_search_time` parameter but lower than some of the extreme cases of the `attack_preparation_time` parameter. Parts of the reason behind this low sensitivity is the fact that the default value (one to five time units) of the `attack_preparation_time` parameter has a rather large range and mitigates some of the extreme cases, such as when the attacks are allowed to be instantaneous.

To sum up, our impact analysis results have a low sensitivity to the `vulnerability_search_time` parameter and higher sensitivity in the case of the `attack_preparation_time` parameter and the `attack_time` parameter since the later parameters are taken into account for each attack. However, the reasons behind the variations in the results are easily understood and do not invalidate any of the previous analyses done throughout this work.

9.8 Conclusions

In conclusion, the impact analysis approach presented in this thesis satisfies the research objectives outlined in Section 1.3 by *taking a timed perspective in analyzing impact* (**RO5**.), providing the *necessary flexibility when modeling and analyzing systems with different granularity* (**RO4**.), *defining impact metrics that are domain-agnostic and reusable* for all ICS (**RO3**.), and *considering the impact on different security objectives pertaining to ICS* (**RO2**.). We also explored how sensitive the results are to certain timing parameters. However, the various design choices made in different stages of the approach may suffer from limited available prior information, state space explosion, incompleteness of the set of system invariants, and limitations in the capability of the modeling environment, among others. Overall, the various general aspects of the proposed impact analysis approach lead to a framework that is *widely applicable to ICS in general*, and thus, reaching our final research objective (**RO1**.).

At the end of this chapter, we are left with:

1. *A firm grasp on the strengths and limitations of the proposed approach*

2. *An understanding of how all of our research objectives have been attained through the different choices and activities in our impact analysis approach*

Chapter 10

Conclusions and Future Work

The IT-OT convergence presents new security challenges, mandating that ICS security be considered differently. The impact analysis approach presented in this thesis is an effort to show, with statistical evidence, that attacks can create an observable and sustained impact on ICS mission objectives, causing damage, delays, or disruptions in system operations. The proposed analysis approach based on timed automata models of an ICS, potential attackers, and system defenses allowed us to see the impact of different data tampering and message spoofing attacks on real-time ICS operations and how the impact of these attacks can be mitigated. The impact analysis results provide actionable information on the most impactful attacks and attackers under consideration and what defensive capabilities provide effective mitigation.

The following sections provide more detailed outcomes of the study undertaken in this thesis. Section 10.1 discusses the outcome of each related chapter. Section 10.2 outlines open research problems. Finally, Section 10.3 presents concluding remarks.

10.1 Outcomes of the Approach

The proposed impact analysis approach, presented in a four-stage framework, enables us to take a modular approach in analyzing the impact of attacks on ICS. In this section, we provide a summary of the chapters related directly to the approach, their outcomes, and how the outcomes resolve the four research questions (outlined in Section 1.4) that we set out to achieve.

Chapter 3: Approach provided us with an overview of the different activities involved in each stage of our impact analysis approach. Additionally, it presented examples of the prior information required to start particular activities in different stages, such as information about the system design description, requirement specification, and common attacks against the target system of analysis. The outcome of this chapter is *a broad understanding of the tasks to be performed in each stage of the proposed approach and what is required to get started.*

Chapter 4: System Modeling presented detailed descriptions of the activities of the first stage of the proposed approach, centered around building timed automata models of an ICS in Uppaal. The activities included modeling an ICS from the system design descriptions, defining various system invariants based on the requirement specifications, and verifying through classical model checking that the system model works as intended. The outcome is understanding *how to use information from the early system development stages to model an ICS and define relevant system invariants to verify the reachability of system mission objectives.*

Chapter 5: Attacker Modeling described how an attacker and their various capabilities could be modeled in the second stage of the proposed approach. The activities revolved around creating different (*random*, *relentless*, and *informed*) attacker models and identifying common attacks against the target ICS from system threat model information to specify such capabilities in the attacker models. The outcome of this chapter is an understanding of *the primary components of a generic attacker model and how to equip it with different attacking capabilities based on system threat model information, and how the model can be extended to capture the behavior of different types of attackers.*

Chapter 6: Attack Execution presented the activities involved in the third stage of our impact analysis approach that is geared towards answering **RQ1**., and in part, **RQ3**. The activities included executing the system model and various attacker models in parallel to see the interactions between the attacks and system operations, and revisiting the system invariants and/or studying the simulation traces to identify the existence of an impact. The outcome of this chapter is the understanding of *how the parallel execution of system and attacker models can allow one to observe the many different impacts of different attacks and how classical model checking can be used to identify the existence of such impacts. The outcomes of this chapter confirm the fact that the impact of cyberattacks aimed at ICS system operations can, in fact, be identified using the proposed approach (***RQ1***.).*

Chapter 7: Impact Analysis presented the fourth stage of the proposed approach and the involved activities centered around answering **RQ2**. and **RQ3**.. The activities entail specifying a set of SMC queries related to system mission completion and

interpreting the SMC results to quantify the impact. The outcome is understanding *how domain-agnostic impact metrics can be defined using SMC queries, and how the query results can be interpreted to estimate the impact on ICS mission objectives. The impact analysis results confirm that the impact of different attacks on ICS can be quantified using diverse metrics (***RQ2***.), which can, in turn, be used to inform appropriate defensive mechanisms. Additionally, the results from this chapter, along with those from Chapter 6 enable us to confirm the differences between the behavior of various types of attackers and the varying degree of impact that they can achieve of ICS mission objectives (***RQ3***.).*

Chapter 8: Defense Modeling revolved around answering **RQ4**., highlighting the applicability of three of the four stages of the proposed approach when system defenses are involved. The activities involved the modeling and verification of the system model augmented with various defensive capabilities in the *System Modeling* stage, parallel execution of the new system model with various attackers to identify the impact of attacks in the *Attack Execution* stage, and interpreting the new SMC query results to quantify the impact of attacks in the presence of different defensive capabilities. The outcome is an understanding of *the activities involved in modeling different system defenses and how only specific activities of the proposed approach need to be repeated when changes are made to the model to obtain new impact analysis results. The outcomes confirm that the impact of attacks can, in fact, be lessened when different system defenses are in place, and the difference between having such a defensive mechanism and being completely defenseless is substantial (***RQ4***.).*

Lastly, **Chapter 9: Assessment of the Approach** discussed different aspects of the proposed impact analysis approach, evaluating their various strengths and limitations. The outcome is an understanding of *how the diverse research challenges (outlined in Section 1.3) in existing impact analysis works have been dealt with in our impact analysis approach and how the discussed limitations open up new doors for future research.*

10.2 Open Research Problems

Our experimental findings and the various existing limitations in the proposed approach pave their way to a wide variety of open research problems. In this section, we discuss such open research areas based on our contributions.

Improvements to automation: At the time of writing this thesis, a way to automatically generate timed automata models from ICS system specifications does not exist [JJ21a]. Therefore, for large ICS, security analysts would have to manually perform the modeling process, which is both time-consuming and error-prone. Similarly, it would be great if researchers had access to Timed Automata specifications of common attacks (e.g., spoofing, tampering, denial-of-service) against ICS. Two possible areas of research would therefore be in developing a tool to automatically translate system specifications to timed automata models or a comprehensive database of common attack specifications.

Extensions to the presented models: Another possible area of work would be in the extension of the various attacker and system defense models presented in this

work. It would be interesting to see, for example, how more complicated attacks by a single attacker, such as advanced persistent threats [ZXW+18] or attacks that require multiple attackers, such as distributed denial-of-service attacks by botnets, can be modeled using timed automata. Another line of work may be to model *colluding attackers* where an attacker would cause a vulnerability to happen (e.g., an insider) in order for another attacker to take advantage of that situation. Other possibilities entail examining if there is a way to identify the type of attacker (e.g., *random*, *relentless*, or *informed*) based on their attack behaviors, i.e., attack signatures. Some possible extensions to defense modeling include the possible negative impact of additional defenses on the system operations or whether the system defenses themselves can be attacked to cause an impact on the system operations.

Research on a bigger scale: The impact analysis approach presented in this work is demonstrated on a Manufacturing Cell Control System that is relatively small and uses abstractions (i.e., agents) to reduce the state space in the model. The applicability of the approach would involve further investigation into how it scales when confronted with real-world industrial systems along with the additional complexity that they present. The determination of a comprehensive set of system invariants and SMC queries would likely be equally challenging since such systems may involve different subsystems with different mission objectives for each of them. Exploration of the exact challenges and the possible ways to overcome them, therefore, opens up another possible path for future research.

Development of a comprehensive impact ranking model: A possible way to extend the work presented in this thesis would be to develop an impact ranking

model that encompasses all the different types of ICS. For a business-oriented ICS (e.g., a manufacturing system), metrics such as the loss of profits, loss in the quality of the produced materials, or loss in reputation due to service unavailability can be considered good candidates to include in an impact ranking model. However, for non-profit systems such as wastewater treatment systems, the considerations would be vastly different. How to develop an impact ranking model that takes into account the widely differing goals of seemingly similar ICS would, therefore, require considerable investigation.

Integrating ICS impact analysis in a secure system development lifecycle: Alongside risk and exploitability, impact is identified to be an integral factor that comprises a comprehensive vulnerability assessment process for smart grids [Dag01]. Our work presents a systematic ICS impact analysis approach highlighting the necessity of impact analysis at the early stages of the system development lifecycle. To extend the contributions, one possible area of research would involve defining what components constitute *a comprehensive ICS impact analysis framework*, how to incorporate such a framework to *a comprehensive ICS vulnerability assessment process*, and how to integrate such a process in various stages of the system development lifecycle to promote *secure ICS development* with built-in security for future ICS.

10.3 Concluding Remarks

The security considerations for ICS will only get more involved with the ensuing integration of various enabling technologies such as IoT, cloud computing, and big data analytics [CRFAF17]. Consequently, security analysts would have to take into account all the inherent and emerging weaknesses in such technologies and the interplay when the OT-aspects of ICSs are entered into the equation of security evaluation and assurance. Impact analysis can provide useful metrics, acting as a criterion to help system developers prioritize the most impactful attackers and attacks as well as the most effective defenses to mitigate such impacts. The work presented in this thesis provides a systematic ICS impact analysis framework, the use of which at various stages of the system development lifecycle can lead to the much-needed awareness of system security among system developers and operators, encouraging revisions in the system design to include appropriate security controls. We believe that such an undertaking would result in a final system deployment with "built-in" security, thus leading to a secure and reliable ICS development for years to come.

Appendix A

UPPAAL

The modeling tool Uppaal [BDL04] and its various modeling and verification capabilities (based on the theory of timed automata [AD94]) are integral to our impact analysis approach. In this chapter, we provide a brief description of the syntax and semantics of the Uppaal modeling language. Section A.1 discusses the graphical notations, expressions, and timed automata extensions used to model real-time systems in Uppaal, whereas Section A.2 describes the query language used by Uppaal.

A.1 Modeling in UPPAAL

Uppaal presents a Java user interface and has its own notations to model timed automata models and the supported extensions. In this section, we introduce the basic graphical notations, the extensions of timed automata, and expressions supported in the Uppaal modeling language.

A.1.1 Graphical Notations in UPPAAL

Figure A.1 shows some basic graphical notations used to model systems in UPPAAL. A brief description of the notations are as follows:

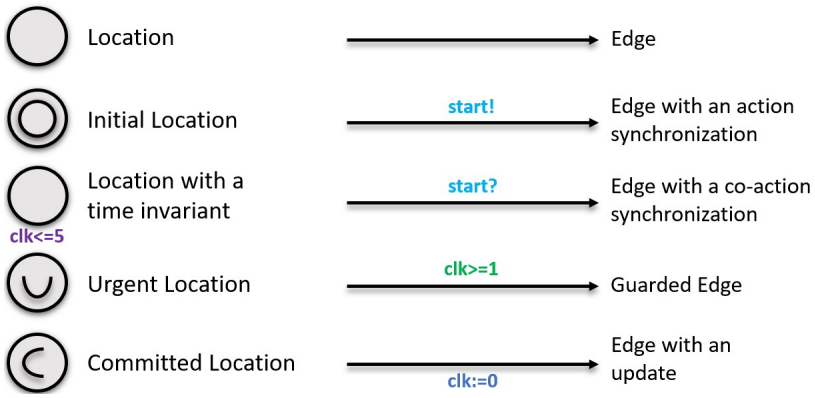


Figure A.1: UPPAAL graphical notations

Locations: *Locations*, denoted by circles, represent the state of an automaton performing a specific function. Regular *locations* are denoted by an empty circle, whereas a single *initial location* for every automaton is depicted by an additional inner circle, specifying the behavior of that automaton when the system starts. Regular and initial locations can be accompanied by a time invariant which are time constraints imposed upon the location. A location with a time invariant x<=5 must be left before five time units pass, for example. Locations are also synonymous to *states* of an automaton, which is the term we will be using extensively throughout the thesis.

Edges: An arrow depicts an *edge*, representing the transition from one state (location) to another. UPPAAL uses a handshaking mechanism to synchronize communication between different automata. During a transition, an edge can send an action synchronization message. This action message (e.g., start!) must synchronize with

another *edge* with a corresponding co-action synchronization message (e.g., start?), enabling the latter transition to take place. An automaton may use state variables to record and share its state with other automata in the network. In addition, clock variables can be used to record the passage of time and impose timing constraints on locations and edges. *Guarded edges* represent conditional transitions, only allowed once the assigned Boolean condition (e.g., clk>=1) over conjunctions or disjunctions of state and/or clock variable expressions are satisfied. Additionally, edges can be accompanied by an *update* (e.g., clk:=0) that updates the value of a state and/or clock variable once the transition is completed.

Urgency: Urgency is an extended behavior of timed automata provided by UPPAAL. *Urgent states* (denoted by a U inside a circle) represent states that are time-critical. No time can pass while an automaton is in such a state, and as such, they cannot have any time invariants, and their outgoing transitions cannot be accompanied by a guard. However, if any other automaton in the network can perform an instantaneous action at that exact moment, that action will be allowed. Semantically, urgent states are equivalent to adding an extra clock variable `x`, that is reset on all incoming edges, and having an invariant x<=0 on the location. *Committed states* (denoted by a C inside a circle) are even more restricted states that allow neither the passage of time nor any action by any other type of state. At any point in time, committed states have the highest priority as all committed states must be left before urgent states are taken into consideration, or time can start to flow again.

A.1.2 Extensions of Timed Automata in UPPAAL

The UPPAAL modeling language provides several extensions of timed automata, the definitions of which have been extracted from [BDL04] below.

Templates: UPPAAL allows the creation of timed automata model templates that can optionally be defined with a set of parameters of any type (e.g., `int, bool`). In the process declaration, these parameters can be substituted for concrete values.

Constants: Constants are declared as `const type name`. By definition, constants cannot be modified and must have an integer value. An example declaration is:

- `const int max_level = 10;` where `max_level` is a constant integer variable, contains the value `10`, and is not modifiable.

Bounded integers: Bounded integer values are declared as `int[min,max] name`, where `min` and `max` are the lower and upper bounds, respectively. Guards, invariants, and assignments may contain expressions ranging over bounded integer variables. The bounds are checked upon verification and violating a bound leads to an invalid state that is discarded (at run-time). UPPAAL uses the default range of -32768 to 32768 if bounds are omitted. Some example declarations include:

- `int[0,10] var1;` where `var1` is a bounded integer variable and can be assigned integers from 0 to 10, and
- `int var2;` where `var2` is bounded within the default range of -32768 to 32768.

Clocks: Clocks, declared as `clock name` are special variables that record the passage of time in an automaton. During simulations, clock values increase as time advances and they can be reset or assigned integer values during transitions. An example declaration is:

- `clock x;` where `x` is a clock variable.

Binary Synchronization: Binary synchronization channels are declared as `chan c`. An edge labelled with `c!` synchronizes with another channel labelled `c?`. If several combinations are enabled, a synchronization pair is chosen non-deterministically. An example declarations is:

- `chan go;` where `go` is a binary synchronization channel.

Broadcast Synchronization: Broadcast synchronization channels are declared as `broadcast chan c`. In a broadcast synchronization, one sender `c!` can synchronize with an arbitrary number of receivers. Whenever a broadcast synchronization is sent, any receiver that can synchronize in their current state must do so. Even if there are no receivers, the sender can still execute the c! action, i.e., broadcast sending is never blocking.

- `broadcast chan go_all;` where `go_all` is a broadcast synchronization channel.

Urgent Synchronization: Urgent synchronization channels are declared by prefixing the channel declaration with the keyword `urgent`. Delays can not occur if a synchronization transition on an urgent channel is enabled. Edges using urgent

channels for synchronization cannot have any time constraints, i.e., no clock guards. Some example declarations include:

- `urgent chan go_now;` where `go_now` is an urgent synchronization channel.
- `urgent broadcast chan go_all_now);` where `go_all_now` is an urgent broadcast synchronization channel.

Arrays: Arrays in UPPAAL are allowed for clocks, channels, constants, and integer variables. They are defined by appending a size to the variable name. Examples include:

- `chan go[3];`
- `clock system_clk[2];`
- `int[1,5] new_var[7];`

Initializers: Initializers are used to initialise integer variables and arrays of integer variables. Examples include:

- `int i := 0;`
- `int arr[3] := 1, 2, 3;`

A.1.3 Expressions in UPPAAL

Expressions in UPPAAL range over clocks and integer variables and are used with colored labels in UPPAAL models. The following definitions of the labels have been

extracted from [BDL04]. The Backus–Naur Form (BNF) of the complete set of UPPAAL expressions can be found in [BDL04].

Guard: A guard is a particular expression satisfying the following conditions: (1) it is side-effect free; (2) it evaluates to a Boolean; (3) only clocks, integer variables, and constants (or arrays of these types) are referenced; (4) clocks and clock differences are only compared to integer expressions; (5) guards over clocks are essentially conjunctions (disjunctions are allowed over integer conditions). Guards may also include side-effect-free functions that evaluate to a Boolean. Examples of guards (depicted in green in UPPAAL) include,

- clk>=3, where `clk` is a clock variable
- i==2, where `i` is an integer variable
- IS_SYSTEM_SAFE(), where `IS_SYSTEM_SAFE()` is a Boolean function
- clk>=3 && i==2, as conjunctions of clock `clk` and integer `i`
- i==2 || IS_SYSTEM_SAFE(), as disjunctions over an integer `i` and a Boolean function `IS_SYSTEM_SAFE()`

Synchronization: A synchronization label is either on the form `expression!` or `expression?` or is an empty label. The expression must be side-effect free, evaluate to a channel, and only refer to integers, constants, and channels. Examples of synchronization label pairs (depicted in light blue in UPPAAL) are go! and go?, or start! and start?.

Assignment: An assignment label is a comma-separated list of expressions with a side-effect; expressions must only refer to clocks, integer variables, and constants, and may only assign integer values to clocks. Assignment labels are depicted in blue in UPPAAL. Assignments may also include functions with a side-effect. Examples include:

- clk:=0, to reset a clock variable `clk`
- i:=5, clk:=0, to assign 5 to an integer `i` and reset a clock variable `clk`
- STOP_PROCESS(), to execute a function `STOP_PROCESS()` that has a side-effect

Invariant: An invariant is an expression that satisfies the following conditions: (1) it is side-effect free; (2) only clock, integer variables, and constants are referenced; (3) it is a conjunction of conditions of the form `x<e` or `x<=e`, where `x` is a clock reference and `e` evaluates to an integer. Invariants are assigned to locations that are not *urgent* or *committed*, and may include side-effect free functions that evaluate to a boolean. Examples of invariants (depicted in violet in UPPAAL) include:

- clk<=10, to enforce the rule that the location must be left within 10 time units using clock variable `clk`
- clk<=10 && pressure<=max_pressure, to enforce the rule that the location must be left within 10 time units, and during that whole time variable `pressure` should remain below the value of the constant `max_pressure`

- IS_SYSTEM_SAFE() && secure, to enforce the rule that in this location a system must always be safe and secure through a Boolean function `IS_SYSTEM_SAFE()` and a Boolean variable `secure`

A.2 Query Language in UPPAAL

The UPPAAL query language consists of *state formulae* and *path formulae*, similar to those of TCTL [BDL04]. In this section, we provide a brief description of the query language of UPPAAL.

Individual states in TCTL are described by state formulae, whereas paths or traces of the model are described by path formulae [BDL04]. A timed automaton TA satisfies a TCTL state formula φ if and only if $s_0 \vDash \varphi$ for each initial state s_0 of TA [BK08]. At each moment in time, TCTL has a branching, tree-like structure, where time may split into alternative courses. The UPPAAL modeling language uses a subset of TCTL where only the operators A, E, $\Box$, and $\Diamond$ are used. A brief description of the operators are as follows:

- A: refers to the idea of "for all" or "inevitably" meaning that along all paths in the model (the specified property is satisfied).
- E: refers to the idea of "there exists" or "possibly" expressing that there exists at least one path in the model (where the specified property is satisfied).
- $\Box$: refers to the idea of "always" meaning now and forever in time. When combined with the A operator, A$\Box\,\varphi$ means that along all possible paths,

a state formula φ is satisfied now and will continue to be satisfied forever. Whereas, combined with the E operator, E$\Box$ φ expresses that there exists at least one path where φ is satisfied now and forever in time.

- $\Diamond$: refers to the idea of "eventually" where something will eventually happen at some point in the future. When combined with the A operator, A$\Diamond$ φ means that along all possible paths, a state formula φ will eventually be satisfied. Whereas, combined with the E operator, E$\Diamond$ φ expresses that there exists at least one path where φ will eventually be satisfied.

For the UPPAAL modeling language, the following definitions of state formulae and path formulae are extracted from [BDL04]:

State Formulae: A state formula is an expression (see Section A.1.3) that can be evaluated for a state without looking at the behavior of the model. The syntax of state formulae is a superset of that of guards; a state formula is a side-effect free expression, but the use of disjunctions is not restricted. An example of this is the simple expression `i == 7`, that is true whenever there is a state where integer `i` equals 7. State properties are evaluated for the initial state and after each transition, meaning that the property `i == 7` may be satisfied even if the value of `i` becomes 7 momentarily during evaluations of initializers or updates on edges.

It is also possible to check whether a particular process is in a given location using an expression of the form `P.l`, where `P` is a process an `l` is a location. A state of *deadlock* is expressed using a special state formula, consisting of the keyword `deadlock`, which

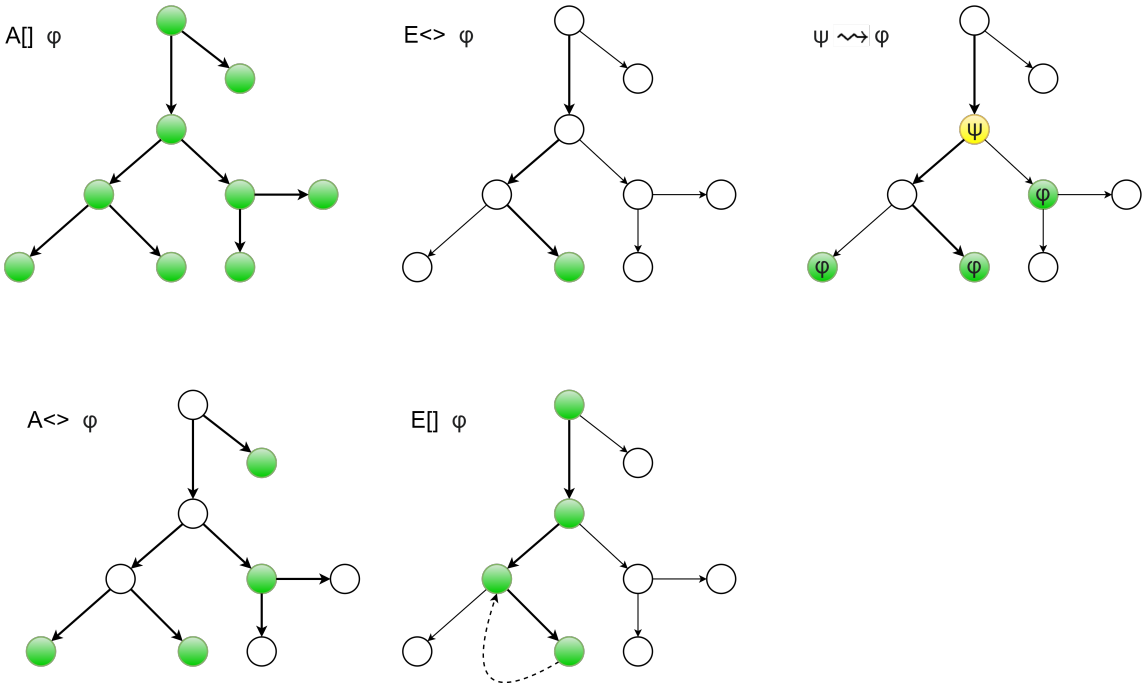


Figure A.2: Path formulae supported in UPPAAL [BDL04]. The filled states represent states for which given state formulae ψ (yellow) and φ (green) hold, respectively. Bold edges show the paths on which the formulae evaluate on.

is satisfied for all deadlock states where there are no outgoing action transitions neither from the state itself or any of its delay successors.

Path Formulae: Path formulae can be categorized into three different classes: (1) reachability properties, (2) safety properties, and (3) liveness properties. The properties are illustrated in Figure A.2 and described below. The descriptions also contain pseudo-formal semantics for the requirement specification language of UPPAAL. In all the descriptions, we assume the existence of a timed transition system $(S, s_0, \rightarrow)$ and state properties ϕ and ψ, as defined in the semantics of UPPAAL [BDL04].

Reachability Properties: Reachability properties are the simplest form of properties and are often used while designing a model to perform sanity checks. They ask

whether a given state formula ϕ possibly can be satisfied by any reachable state. In essence, this asks: "does there exist a path starting at the initial state, such that φ, is eventually satisfied along that path". These properties do not, by themselves, guarantee the correctness of the model, but they validate the basic model behavior.

We express that some state satisfying φ should be reachable using the path formula $\mathrm{E}\Diamond\,\varphi$. In UPPAAL, we write this as the *possibly* property using the syntax `E<>`φ. The *possibly* property `E<>`φ evaluates to true for a timed transition system if and only if there is a sequence of alternating delay transitions and action transitions $s_0 \rightarrow s_1 \rightarrow ... \rightarrow s_n$, where s_0 is the initial state and s_n satisfies φ.

Safety Properties: Safety properties are of the form: "something bad will never happen." For example, in a chemical reactor model, a safety property might include that the pressure of a boiler is always (invariantly) under a certain threshold, or that an explosion may never occur. In UPPAAL this is called a *invariantly* property and is formulated positively, e.g., something good is invariantly true. We express that state formula φ should be true in all reachable state with the path formula $\mathrm{A}\Box\,\varphi$. In UPPAAL notations, this is expressed as `A[]`φ, which evaluates to true if (and only if) every reachable state satisfies φ.

Another way to state safety properties is that "something will possibly never happen." For example, in a nuclear power plant model, a safety property may include that the operating temperature is never below a certain threshold. This can thus be expressed as a possibly property $\neg\,\mathrm{E}\Diamond\,\neg\varphi$ and written as `not E<> not` φ in UPPAAL. Another variation is the *potentially always* property expressed as $\mathrm{E}\Box\,\varphi$, stating that there

should exist a maximal[1] path such that φ is always true. In UPPAAL, this is expressed as `E[]`φ.

Liveness Properties: Liveness properties are of the form: "something will eventually happen." An example of this is that when pressing the laptop start button, it should eventually turn on. In its simple form, liveness is expressed with the path formula $\mathrm{A}\Diamond\,\varphi$, meaning φ is eventually satisfied along the path. In UPPAAL, this is expressed as the *eventually* property `E<>`φ, which evaluates to true if and only if all possible transition sequences eventually reach a state satisfying φ. Alternatively, this can be expressed as the *potentially* property $\neg\,\mathrm{E}\Box\,\neg\varphi$, written as `not E[] not` φ in UPPAAL.

A more useful form of liveness properties is the *leads to* or *response* property of the form: "when something has happened this will eventually lead to something else happening". An example of this is a model of a communication protocol where, when a message has been sent, it will eventually be received. The leads to property is expressed as $\psi \rightsquigarrow \varphi$ and written as ψ `-->` φ in UPPAAL, meaning that whenever ψ holds, φ will hold as well. Alternatively, a leads to property can also be expressed as $\mathrm{A}\Box(\varphi \implies \mathrm{A}\Diamond\,\psi)$ written as `A[](`ψ `imply A<>` φ`)` in UPPAAL.

General liveness properties are often not sufficiently expressive to ensure the correctness, especially in the case of real-time systems where hard real-time deadlines must be ensured. Therefore, rather than inspecting whether a particular property φ is guaranteed to hold eventually should be associated with the examining whether φ will also hold within a certain upper time limit. A *time-bounded leads to* operator

[1] A maximal path is a path that is either infinite or where the last state has no outgoing transitions.

Table A.1: Properties supported by UPPAAL

Name	Property	Equivalent to
Possibly	`E<>`φ	
Invariantly	`A[]`φ	`not E<> not` φ
Potentially always	`E[]`φ	
Eventually	`E<>`φ	`not E[] not` φ
Leads to	ψ `-->` φ	`A[](`ψ `imply A<>` φ`)`

is expressed as $\psi \rightsquigarrow_{\leq t} \varphi$ meaning whenever the state formula ψ holds, the state formula φ must also hold within at most t time units thereafter. In UPPAAL, this can be checked by first extending the model with an additional clock `z` which is reset whenever ψ starts to hold. The time-bounded leads to property can then simply be obtained by verifying $\psi \rightsquigarrow (\varphi \wedge z \leq t)$.

In summary, the UPPAAL requirement specification language supports five types of properties, which can be reduced to two types as illustrated by Table A.1.

Statistical Model Checking in UPPAAL

UPPAAL can estimate the probability of expression values statistically [DLL+15]. There are four types of statistical properties: quantitative, qualitative, comparison, and probable value estimation. In our work, we use the probability estimation (quantitative model checking) and probable value estimation properties, the definitions of which are as follows:

Probability Estimation (Quantitative Model Checking) In UPPAAL, probability estimation queries are expressed as

```
'Pr' SMCBounds '(' ('<>' | '[]') Expression ')'
```

These quantitative queries estimate the probability of a path expression being true given that the predicate in probability brackets is true. Intuitively the model exploration is bounded by an expression in the brackets: it can be limited by setting the bound (i.e., `SMCBounds`) on a clock value, model time or the number of steps (discrete transitions). The result is an estimated probability (with confidence interval ϵ) and a number of histograms over the values of the variable specified in the probability brackets. Histograms omit runs that do not satisfy the property. Examples include:

- `Pr[clk<=1000; 100] ([] temp <= 45)`, estimates the probability of the temperature always staying at or below 45 degrees throughout 1000 time units and with 100 simulation runs, where `temp` is an integer representing temperature and `clk` is a clock variable.
- `Pr[clk<=1000; 100] (<> temp > 100)`, estimates the probability of the temperature eventually exceeding 100 degrees within the same `SMCBounds`.

Value Estimation In UPPAAL, probable value estimation queries are expressed as

```
'E' SMCBounds '(' ('min:'|'max:')  Expression ')'
```

The queries estimate the expected mean value of an expression by running a given number of simulations. Examples include:

- `E [clk<=3600; 25] (max:  pressure)`, estimates the average expected maximum pressure (i.e., maximum value of the integer variable `pressure`) within

3600 time units and 25 simulation runs, where `clk` is a clock variable. The `SMCBounds` of 3600 time units represent an hour, if each time unit is considered to be one second.

- `E [clk<=3600; 100] (min: pressure)`, estimates the average expected minimum pressure (i.e., minimum value of the integer variable `pressure`) within 3600 time units and 100 simulation runs.

Bibliography

[AAB+17] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J Alex Halderman, Luca Invernizzi, Michalis Kallitsis, et al. Understanding the mirai botnet. In *26th {USENIX} security symposium ({USENIX} Security 17)*, pages 1093–1110, 2017.

[ACBS17] Darren Anstee, C.F. Chui, Paul Bowen, and Gary Sockrider. 12th annual worldwide infrastructure security report. Special report, Arbor Networks Inc., 2017.

[AD94] Rajeev Alur and David L Dill. A theory of timed automata. *Theoretical computer science*, 126(2):183–235, 1994.

[AHT20] Uchenna Daniel Ani, Hongmei He, and Ashutosh Tiwari. Vulnerability-based impact criticality estimation for industrial control systems. In *2020 International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*, pages 1–8. IEEE, 2020.

[AM16] Sridhar Adepu and Aditya Mathur. An investigation into the response of a water treatment system to cyber attacks. In *2016 IEEE 17th International Symposium on High Assurance Systems Engineering (HASE)*, pages 141–148. IEEE, 2016.

[BBB+10] Ananda Basu, Saddek Bensalem, Marius Bozga, Benoît Caillaud, Benoît Delahaye, and Axel Legay. Statistical abstraction and model-checking of large heterogeneous systems. In *Formal Techniques for Distributed Systems*, pages 32–46. Springer, 2010.

[BCD+07] Gerd Behrmann, Agnes Cougnard, Alexandre David, Emmanuel Fleury, Kim G Larsen, and Didier Lime. Uppaal-tiga: Time for playing games! In *International Conference on Computer Aided Verification*, pages 121–125. Springer, 2007.

[BCSS11] David Basin, Srdjan Capkun, Patrick Schaller, and Benedikt Schmidt. Formal reasoning about physical properties of security protocols. *ACM Transactions on Information and System Security (TISSEC)*, 14(2):1–28, 2011.

[BDL04] Gerd Behrmann, Alexandre David, and Kim G Larsen. A tutorial on uppaal. In *Formal methods for the design of real-time systems*, pages 200–236. Springer, 2004.

[BDL+12] Peter Bulychev, Alexandre David, Kim Gulstrand Larsen, Marius Mikučionis, Danny Bøgsted Poulsen, Axel Legay, and Zheng Wang. UPPAAL-SMC: Statistical model checking for priced timed automata. *arXiv preprint arXiv:1207.1272*, 2012.

[BFH+01] Gerd Behrmann, Ansgar Fehnker, Thomas Hune, Kim Larsen, Paul Pettersson, and Judi Romijn. Efficient guiding towards cost-optimality in uppaal. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 174–188. Springer, 2001.

[BHV00] Gerd Behrmann, Thomas Hune, and Frits Vaandrager. Distributing timed model checking—how the search order matters. In *International Conference on Computer Aided Verification*, pages 216–231. Springer, 2000.

[BJLY98] Johan Bengtsson, Bengt Jonsson, Johan Lilius, and Wang Yi. Partial order reductions for timed systems. In *International Conference on Concurrency Theory*, pages 485–500. Springer, 1998.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT press, 2008.

[BL04] Eric Byres and Justin Lowe. The myths and facts behind cyber security risks for industrial control systems. In *Proceedings of the VDE Kongress*, volume 116, pages 213–218. Citeseer, 2004.

[BLP+99] Gerd Behrmann, Kim G Larsen, Justin Pearson, Carsten Weise, and Wang Yi. Efficient timed reachability analysis using clock difference diagrams. In *International Conference on Computer Aided Verification*, pages 341–353. Springer, 1999.

[BLR04] Gerd Behrmann, Kim G Larsen, and Jacob I Rasmussen. Priced timed automata: Algorithms and applications. In *International symposium on*

formal methods for components and objects, pages 162–182. Springer, 2004.

[Can09] Public Safety Canada. National strategy for critical infrastructure, 2009.

[CAS08] 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*, pages 495–500. IEEE, 2008.

[CAS+09] Alvaro Cardenas, Saurabh Amin, Bruno Sinopoli, Annarita Giani, Adrian Perrig, Shankar Sastry, et al. Challenges for securing cyber physical systems. In *Workshop on future directions in cyber-physical systems security*, volume 5. Citeseer, 2009.

[Cat10] Daniele Catteddu. Cloud computing: Benefits, risks and recommendations for information security. In Carlos Serrão, Vicente Aguilera Díaz, and Fabio Cerullo, editors, *Web Application Security*, pages 17–17. Springer, 2010.

[CB06] Alvaro A Cárdenas and John S Baras. Evaluation of classifiers: Practical considerations for security applications. In *Proc. AAAI Workshop Evaluation Methods for Machine Learning*, pages 777–780, 2006.

[CbO17] Daniela Soares Cruzes and Lotfi ben Othmane. Threats to validity in empirical software security research. *Empirical Research for Software Security: Foundations and Experience*, 2017.

[CBPK13] Bo Chen, Karen L Butler-Purry, and Deepa Kundur. Impact analysis of transient stability due to cyber attack on FACTS devices. In *2013 North American Power Symposium*, pages 1–6. IEEE, 2013.

[CCC19] Raymond Chan, Kam-Pui Chow, and Chun-Fai Chan. Defining attack patterns for industrial control systems. In *International Conference on Critical Infrastructure Protection*, pages 289–309. Springer, 2019.

[CFL+08] Edmund M Clarke, James R Faeder, Christopher J Langmead, Leonard A Harris, Sumit Kumar Jha, and Axel Legay. Statistical model checking in BioLab: Applications to the automated analysis of T-cell receptor signaling pathway. In *International Conference on Computational Methods in Systems Biology*, pages 231–250. Springer, 2008.

[CFN+18] Carmen Cheh, Ahmed Fawaz, Mohammad A Noureddine, Binbin Chen, William G Temple, and William H Sanders. Determining tolerable attack surfaces that preserves safety of cyber-physical systems. In *2018 IEEE 23rd Pacific Rim International Symposium on Dependable Computing (PRDC)*, pages 125–134. IEEE, 2018.

[CL07] Henrik Christiansson and Eric Luiijf. Creating a european scada security testbed. In *International Conference on Critical Infrastructure Protection*, pages 237–247, Boston, Massachusetts, 2007. Springer.

[CMM+14] Hasan Cam, Pierre Mouallem, Yilin Mo, Bruno Sinopoli, and Benjamin Nkrumah. Modeling impact of attacks, recovery, and attackability conditions for situational awareness. In *2014 IEEE International Inter-Disciplinary Conference on Cognitive Methods in Situation Awareness and Decision Support*, pages 181–187. IEEE, 2014.

[CPE+17] Oana Chenaru, Dan Popescu, Dragos Enache, Loretta Ichim, and Florin Stoican. Improving operational security for web-based distributed control systems in wastewater management. In *2017 25th Mediterranean Conference on Control and Automation*, pages 1089–1093. IEEE, 2017.

[CRFAF17] Sujit Rokka Chhetri, Nafiul Rashid, Sina Faezi, and Mohammad Abdullah Al Faruque. Security trends and advances in manufacturing systems in the era of industry 4.0. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1039–1046. IEEE, 2017.

[Cyb12] Cybersecurity and Infrastructure Security Agency. ICS-CERT Monthly Monitor, October–December 2012.

[Dag01] Jeffrey Dagle. Vulnerability assessment activities [for electric utilities]. In *2001 IEEE Power Engineering Society Winter Meeting. Conference Proceedings (Cat. No. 01CH37194)*, volume 1, pages 108–113. IEEE, 2001.

[DD08] Tore Dybå and Torgeir Dingsøyr. Strength of evidence in systematic reviews in software engineering. In *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, pages 178–187, 2008.

[DDL+12] Alexandre David, Dehui Du, Kim G Larsen, Axel Legay, Marius Mikučionis, Danny Bøgsted Poulsen, and Sean Sedwards. Statistical model checking for stochastic hybrid systems. *arXiv preprint arXiv:1208.3856*, 2012.

[DDL+13] Alexandre David, Dehui Du, Kim Guldstrand Larsen, Axel Legay, and Marius Mikučionis. Optimizing control strategy using statistical model checking. In *NASA Formal Methods Symposium*, pages 352–367. Springer, 2013.

[DJL+15] Alexandre David, Peter Gjøl Jensen, Kim Guldstrand Larsen, Marius Mikučionis, and Jakob Haahr Taankvist. Uppaal stratego. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 206–211. Springer, 2015.

[DLL+15] Alexandre David, Kim G Larsen, Axel Legay, Marius Mikučionis, and Danny Bøgsted Poulsen. Uppaal smc tutorial. *International Journal on Software Tools for Technology Transfer*, 17(4):397–415, 2015.

[DMY02] Alexandre David, M Oliver Möller, and Wang Yi. Formal verification of uml statecharts with real-time extensions. In *International Conference on Fundamental Approaches to Software Engineering*, pages 218–232. Springer, 2002.

[DSV15] Zakarya Drias, Ahmed Serhrouchni, and Olivier Vogel. Taxonomy of attacks on industrial control protocols. In *2015 International Conference on Protocol Engineering (ICPE) and International Conference on New Technologies of Distributed Systems (NTDS)*, pages 1–6. IEEE, 2015.

[DY83] Danny Dolev and Andrew Yao. On the security of public key protocols. *IEEE Transactions on information theory*, 29(2):198–208, 1983.

[ES09] Mathias Ekstedt and Teodor Sommestad. Enterprise architecture models for cyber security analysis. In *2009 IEEE/PES Power Systems Conference and Exposition*, pages 1–6. IEEE, 2009.

[EVM+10] Peyman Mohajerin Esfahani, Maria Vrakopoulou, Kostas Margellos, John Lygeros, and Göran Andersson. Cyber attack in a two-area power system: Impact identification using reachability. In *Proceedings of the 2010 American control conference*, pages 962–967. IEEE, 2010.

[FCW+05] Casey Fung, Yi-Liang Chen, Xinyu Wang, Joseph Lee, Richard Tarquini, Mark Anderson, and Richard Linger. Survivability analysis of distributed systems using attack tree methodology. In *2005 IEEE Military Communications Conference*, pages 583–589. IEEE, 2005.

[FKL+13] Michael D Ford, Ken Keefe, Elizabeth LeMay, William H Sanders, and Carol Muehrcke. Implementing the advise security modeling formalism in möbius. In *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 1–8. IEEE, 2013.

[FLH16] Andrew Fielder, Tingting Li, and Chris Hankin. Defense-in-depth vs. critical component defense for industrial control systems. 2016.

[FM10] Robert Feldt and Ana Magazinius. Validity threats in empirical software engineering research-an initial survey. In *Seke*, pages 374–379, 2010.

[FMC11] Nicolas Falliere, Liam O Murchu, and Eric Chien. W32. stuxnet dossier. *White paper, Symantec Corp., Security Response*, 5(6):29, 2011.

[GD14] Cesare Guariniello and Daniel DeLaurentis. Communications, information, and cyber security in systems-of-systems: Assessing the impact of attacks through interdependency analysis. *Procedia Computer Science*, 28:720–727, 2014.

[GGGAD17] Gustavo Gonzalez-Granadillo, Joaquin Garcia-Alfaro, and Hervé Debar. A polytope-based approach to measure the impact of events against critical infrastructures. *Journal of Computer and System Sciences*, 83(1):3–21, 2017.

[GKA17] Benjamin Green, Marina Krotofil, and Ali Abbasi. On the significance of process comprehension for conducting targeted ics attacks. In *Proceedings of the 2017 Workshop on Cyber-Physical Systems Security and PrivaCy*, pages 57–67, 2017.

[Gol12] Dieter Gollmann. Veracity, plausibility, and reputation. In *IFIP International Workshop on Information Security Theory and Practice*, pages 20–28. Springer, 2012.

[GTG+13] Gleb Gritsai, Alexander Timorin, Yury Goltsev, Roman Ilin, Sergey Gordeychik, and Anton Karpin. SCADA safety in numbers. *Positive technologies. Moscow (Russia)*, 10:16–23, 2013.

[GWP19] Alexander Giehl, Norbert Wiedermann, and Sven Plaga. A framework to assess impacts of cyber attacks in manufacturing. In *Proceedings of*

the 2019 11th International Conference on Computer and Automation Engineering, ICCAE 2019, pages 127–132, New York, NY, USA, 2019. Association for Computing Machinery.

[HBL+03] Martijn Hendriks, Gerd Behrmann, Kim Larsen, Peter Niebert, and Frits Vaandrager. Adding symmetry reduction to uppaal. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 46–59. Springer, 2003.

[HCSB16] Ashish R Hota, Abraham A Clements, Shreyas Sundaram, and Saurabh Bagchi. Optimal and game-theoretic deployment of security investments in interdependent assets. In *International Conference on Decision and Game Theory for Security*, pages 101–113. Springer, 2016.

[HDKK20] Saqib Hasan, Abhishek Dubey, Gabor Karsai, and Xenofon Koutsoukos. A game-theoretic approach for power systems defense against dynamic cyber-attacks. *International Journal of Electrical Power & Energy Systems*, 115:105432, 2020.

[HF+18] Kevin E Hemsley, E Fisher, et al. History of industrial control system cyber incidents. Technical report, Idaho National Lab.(INL), Idaho Falls, ID (United States), 2018.

[HGD+18] Saqib Hasan, Amin Ghafouri, Abhishek Dubey, Gabor Karsai, and Xenofon Koutsoukos. Vulnerability analysis of power systems based on cyber-attack and defense models. In *2018 IEEE Power & Energy Society Innovative Smart Grid Technologies Conference (ISGT)*, pages 1–5. IEEE, 2018.

[HK19] Li Huang and Eun-Young Kang. Formal verification of safety & security related timing constraints for a cooperative automotive system. In *International Conference on Fundamental Approaches to Software Engineering*, pages 210–227. Springer, 2019.

[HKVW15] Hannes Holm, Martin Karresand, Arne Vidström, and Erik Westring. A survey of industrial control system testbeds. In *Nordic Conference on Secure IT Systems*, pages 11–26. Springer, 2015.

[HL02] Martijn Hendriks and Kim G Larsen. Exact acceleration of real-time model checking. *Electronic Notes in Theoretical Computer Science*, 65(6):120–139, 2002.

[HQD+03] Salim Hariri, Guangzhi Qu, Tushneem Dharmagadda, Modukuri Ramkishore, and Cauligi S Raghavendra. Impact analysis of faults and attacks in large-scale networks. *IEEE Security & Privacy*, 1(5):49–54, 2003.

[HWY17] Peng Huang, Yinan Wang, and Gangfeng Yan. Vulnerability analysis of electrical cyber physical systems using a simulation platform. In *43rd Annual Conference of the IEEE Industrial Electronics Society*, pages 489–494. IEEE, 2017.

[HZQT19] Kaixing Huang, Chunjie Zhou, Yuanqing Qin, and Weixun Tu. A game-theoretic approach to cross-layer security decision-making in industrial cyber-physical systems. *IEEE Transactions on Industrial Electronics*, 67(3):2371–2379, 2019.

[HZT+18] Kaixing Huang, Chunjie Zhou, Yu-Chu Tian, Shuanghua Yang, and Yuanqing Qin. Assessing the physical impact of cyberattacks on industrial cyber-physical systems. *IEEE Transactions on Industrial Electronics*, 65(10):8153–8162, 2018.

[Ing21] Terrance R Ingoldsby. Attack tree-based threat risk analysis. Technical report, Amenaza Technologies Ltd., 2021.

[Jak11] Gabriel Jakobson. Mission cyber security situation assessment using impact dependency graphs. In *14th international conference on information fusion*, pages 1–8. IEEE, 2011.

[Jas20a] Jason Jaskolka. Identifying and analyzing implicit interactions in a wastewater dechlorination system. In *Computer Security*, pages 34–51. Springer, 2020.

[Jas20b] Jason Jaskolka. Recommendations for effective security assurance of software-dependent systems. In Kohei Arai, Supriya Kapoor, and Rahul Bhatia, editors, *Intelligent Computing, SAI 2020*, volume 1230 of *Advances in Intelligent Systems and Computing*, pages 511–531. Springer, Cham, 2020.

[JdLAF18] Ananth A Jillepalli, Daniel Conte de Leon, and Jim Alves-Foss. Operational characteristics of modern malware: Pco threats. In *Proceedings of the Fifth Cybersecurity Symposium*, pages 1–6, 2018.

[JHH12] Karel Jezernik, Robert Horvat, and Joze Harnik. Finite-state machine motion controller: servo drives. *IEEE Industrial Electronics Magazine*, 6(3):13–23, 2012.

[JJ21a] Alvi Jawad and Jason Jaskolka. Analyzing the impact of cyberattacks on industrial control systems using timed automata. In *2021 IEEE 21st International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, 2021.

[JJ21b] Alvi Jawad and Jason Jaskolka. Modeling and simulation approaches for cybersecurity impact analysis: State-of-the-art. In *2021 Annual Modeling and Simulation Conference (ANNSIM)*, pages 1–12. IEEE, 2021.

[JV17a] Jason Jaskolka and John Villasenor. An approach for identifying and analyzing implicit interactions in distributed systems. *IEEE Transactions on Reliability*, 66(2):529–546, June 2017.

[JV17b] Jason Jaskolka and John Villasenor. Identifying implicit component interactions in distributed cyber-physical systems. In *50th Hawaii International Conference on System Sciences*, HICSS-50, pages 5988–5997, January 2017.

[KBTJ20] Tomas Kulik, Jalil Boudjadar, and Peter WV Tran-Jørgensen. Security verification of industrial control systems using partial model checking. In *8th International Conference on Formal Methods in Software Engineering*, pages 98–108, 2020.

[KFL+10] Deepa Kundur, Xianyong Feng, Shan Liu, Takis Zourntos, and Karen L Butler-Purry. Towards a framework for cyber attack impact analysis of the electric smart grid. In *2010 First IEEE international conference on smart grid communications*, pages 244–249. IEEE, 2010.

[KFM+11] Deepa Kundur, Xianyong Feng, Salman Mashayekh, Shan Liu, Takis Zourntos, and Karen L Butler-Purry. Towards modelling the impact of cyber attacks on a smart grid. *International Journal of Security and Networks*, 6(1):2–13, 2011.

[KG13] Maryna Krotofil and Dieter Gollmann. Industrial control systems security: What is happening? In *2013 11th IEEE International Conference on Industrial Informatics (INDIN)*, pages 670–675. IEEE, 2013.

[KMKP16] Rajesh Kalluri, Lagineni Mahendra, RK Senthil Kumar, and GL Ganga Prasad. Simulation and impact analysis of denial-of-service attacks on power scada. In *2016 National Power Systems Conference*, pages 1–5. IEEE, 2016.

[KNP11] 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.

[KS17] Rajesh Kumar and Mariëlle Stoelinga. Quantitative security and safety analysis with attack-fault trees. In *2017 IEEE 18th International Symposium on High Assurance Systems Engineering (HASE)*, pages 25–32. IEEE, 2017.

[KZH+11] Joost-Pieter Katoen, Ivan S Zapreev, Ernst Moritz Hahn, Holger Hermanns, and David N Jansen. The ins and outs of the probabilistic model checker MRMC. *Performance evaluation*, 68(2):90–104, 2011.

[L+13] Wenyuan Li et al. *Reliability assessment of electric power systems using Monte Carlo methods*. Springer Science & Business Media, 2013.

[LFK+11] Elizabeth LeMay, Michael D Ford, Ken Keefe, William H Sanders, and Carol Muehrcke. Model-based security metrics using adversary view security evaluation (advise). In *2011 Eighth International Conference on Quantitative Evaluation of SysTems*, pages 191–200. IEEE, 2011.

[LFK14] Antoine Lemay, José Fernandez, and Scott Knight. Modelling physical impact of cyber attacks. In *2014 Workshop on Modeling and Simulation of Cyber-Physical Energy Systems*, pages 1–6. IEEE, 2014.

[LLPY97] Kim Guldstrand Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. Efficient verification of real-time systems: Compact data structure and state-space reduction. In *Proceedings Real-Time Systems Symposium*, pages 14–24. IEEE, 1997.

[LMNS05] Kim G Larsen, Marius Mikucionis, Brian Nielsen, and Arne Skou. Testing real-time embedded software using uppaal-tron: an industrial case study. In *Proceedings of the 5th ACM international conference on Embedded software*, pages 299–306, 2005.

[LNR11] 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):1–33, 2011.

[LXDW16] Weize Li, Lun Xie, Zulan Deng, and Zhiliang Wang. False sequential logic attack on SCADA system and its physical impact analysis. *Computers & Security*, 58:149–159, 2016.

[LYY+12] Jie Lin, Wei Yu, Xinyu Yang, Guobin Xu, and Wei Zhao. On false data injection attacks against distributed energy routing in smart grid. In *2012 IEEE/ACM Third International Conference on Cyber-Physical Systems*, pages 183–192. IEEE, 2012.

[MAM+18] Ronierison Maciel, Jean Araujo, Carlos Melo, Jamilson Dantas, and Paulo Maciel. Impact assessment of multi-threats in computer systems using attack tree modeling. In *2018 IEEE International Conference on Systems, Man, and Cybernetics*, pages 2448–2453. IEEE, 2018.

[Mes07] Jeanne Meserve. Mouse click could plunge city into darkness, experts say. *CNN. com*, 27, 2007.

[Mey20] Rounak Meyur. A bayesian attack tree based approach to assess cyber-physical security of power system. In *2020 IEEE Texas Power and Energy Conference*, pages 1–6. IEEE, 2020.

[MGKL19] Rômulo Meira-Góes, Raymond Kwong, and Stéphane Lafortune. Synthesis of sensor deception attacks for systems modeled as probabilistic

automata. In *2019 American Control Conference (ACC)*, pages 5620–5626. IEEE, 2019.

[Mica] Trend Micro. Industrial control system. Accessed on 3.11.2021.

[Micb] Trend Micro. Why do attackers target industrial control systems? Accessed on 9.11.2021.

[MKW+16] Stephen McLaughlin, Charalambos Konstantinou, Xueyang Wang, Lucas Davi, Ahmad-Reza Sadeghi, Michail Maniatakos, and Ramesh Karri. The cybersecurity landscape in industrial control systems. *Proceedings of the IEEE*, 104(5):1039–1057, 2016.

[Mor21] Sara Morrison. Ransomware attack hits another massive, crucial industry: Meat, Jun 2021.

[MPM20] Andrei Munteanu, Michele Pasqua, and Massimo Merro. Impact analysis of cyber-physical attacks on a water tank system via statistical model checking. In *8th International Conference on Formal Methods in Software Engineering*, pages 34–43, 2020.

[MSR06] Peter Mell, Karen Scarfone, and Sasha Romanosky. Common vulnerability scoring system. *IEEE Security & Privacy*, 4(6):85–89, 2006.

[MTT+11] Scott Musman, Mike Tanner, Aaron Temin, Evan Elsaesser, and Lewis Loren. Computing the impact of cyber attacks on complex missions. In *2011 IEEE International Systems Conference*, pages 46–51. IEEE, 2011.

[NAAA18] Aaamir Naeem, Farooque Azam, Anam Amjad, and Muhammad Waseem Anwar. Comparison of model checking tools using timed automata–PRISM and UPPAAL. In *2018 IEEE International Conference on Computer and Communication Engineering Technology (CCET)*, pages 248–253. IEEE, 2018.

[PSSS04] Tom Parker, Marcus Sachs, Eric Shaw, and Ed Stroz. *Cyber adversary characterization: Auditing the hacker mind*. Elsevier, 2004.

[Ros00] L Rossman. Epanet. software that models the hydraulic and water quality behavior of water distribution piping systems. cincinnati, oh 45268: Us environmental protection agency (epa), 2000.

[RPK01] Steven M Rinaldi, James P Peerenboom, and Terrence K Kelly. Identifying, understanding, and analyzing critical infrastructure interdependencies. *IEEE control systems magazine*, 21(6):11–25, 2001.

[RPM+16] Nageswara SV Rao, Stephen W Poole, Chris YT Ma, Fei He, Jun Zhuang, and David KY Yau. Defense of cyber infrastructures against cyber-physical attacks using game-theoretic models. *Risk Analysis*, 36(4):694–710, 2016.

[RT16] Marco Rocchetto and Nils Ole Tippenhauer. On attacker models and profiles for cyber-physical systems. In *European Symposium on Research in Computer Security*, pages 427–449. Springer, 2016.

[Rus21] Mary-Ann Russon. Us fuel pipeline hackers 'didn't mean to create problems', May 2021.

[SDF17] Bastien Sultan, Fabien Dagnat, and Caroline Fontaine. A methodology to assess vulnerabilities and countermeasures impact on the missions of a naval system. In *Computer Security*, pages 63–76. Springer, 2017.

[SEN09] Teodor Sommestad, Mathias Ekstedt, and Lars Nordstrom. Modeling security of power communication systems using defense graphs and influence diagrams. *IEEE Transactions on Power Delivery*, 24(4):1801–1808, 2009.

[SHÇ+10] James PG Sterbenz, David Hutchison, Egemen K Çetinkaya, Abdul Jabbar, Justin P Rohrer, Marcus Schöller, and Paul Smith. Resilience and survivability in communication networks: Strategies, principles, and survey of disciplines. *Computer Networks*, 54(8):1245–1265, 2010.

[SL14] Alexandru Stefanov and Chen-Ching Liu. Cyber-physical system security and impact analysis. *IFAC Proceedings Volumes*, 47(3):11238–11243, 2014.

[Slo19] Joseph Slowik. Evolution of ics attacks and the prospects for future disruptive events. *Threat Intelligence Centre Dragos Inc*, 2019.

[SM07] Jill Slay and Michael Miller. Lessons learned from the maroochy water breach. In *International conference on critical infrastructure protection*, pages 73–82. Springer, 2007.

[SM17] Gayathri Sugumar and Aditya Mathur. Testing the effectiveness of attack detection mechanisms in industrial control systems. In *2017 IEEE*

International Conference on Software Quality, Reliability and Security Companion (QRS-C), pages 138–145. IEEE, 2017.

[Smi01] Tony Smith. Hacker jailed for revenge sewage attacks. *the Register*, 31, 2001.

[SO16] Elad Salomons and Avi Ostfeld. Simulation of cyber-physical attacks on water distribution systems with epanet. In *Proceedings of the Singapore Cyber-Security Conference (SGCRC) 2016: Cyber-Security by Design*, volume 14, page 123, 2016.

[SVA05] Koushik Sen, Mahesh Viswanathan, and Gul Agha. VESTA: A statistical model-checker and analyzer for probabilistic systems. In *Second International Conference on the Quantitative Evaluation of Systems (QEST'05)*, pages 251–252. IEEE, 2005.

[SWB04] Javier Salmeron, Kevin Wood, and Ross Baldick. Analysis of electric grid security under terrorist threat. *IEEE Transactions on power systems*, 19(2):905–912, 2004.

[SWW15] Ahmad-Reza Sadeghi, Christian Wachsmann, and Michael Waidner. Security and privacy challenges in industrial internet of things. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference*, pages 1–6. Piscataway, New Jersey, Institute of Electrical and Electronics Engineers, Inc, 2015.

[TBG16] Unal Tatar, Hayretdin Bahsi, and Adrian Gheorghe. Impact assessment of cyber attacks: A quantification study on power generation systems.

In *2016 11th System of Systems Engineering Conference*, pages 1–6. IEEE, 2016.

[TGD+19] Riccardo Taormina, Stefano Galelli, HC Douglas, Nils Ole Tippenhauer, Elad Salomons, and Avi Ostfeld. A toolbox for assessing the impacts of cyber-physical attacks on water distribution systems. *Environmental modelling & software*, 112:46–51, 2019.

[TPSJ12] 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*, pages 55–64, 2012.

[UGTC16] David Urbina, Jairo Giraldo, Nils Ole Tippenhauer, and Alvaro Cardenas. Attacking fieldbus communications in ics: Applications to the swat testbed. In *Proceedings of the Singapore Cyber-Security Conference (SG-CRC) 2016*, pages 75–89. IOS Press, 2016.

[Vig12] Roberto Vigo. The cyber-physical attacker. In *International Conference on Computer Safety, Reliability, and Security*, pages 347–356. Springer, 2012.

[Wai17] Gabriel A Wainer. *Discrete-event modeling and simulation: a practitioner's approach*. CRC press, 2017.

[WCWW14] Lee J Wells, Jaime A Camelio, Christopher B Williams, and Jules White. Cyber-physical security challenges in manufacturing systems. *Manufacturing Letters*, 2(2):74–77, 2014.

[WRH+12] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in software engineering*. Springer Science & Business Media, 2012.

[YCG+18] Ercan Ylmaz, Bünyamin Ciylan, Serkan Gonen, Erhan Sindiren, and Gökçe Karacayılmaz. Cyber security in industrial control systems: Analysis of dos attacks against plcs and the insider effect. In *6th International Istanbul Smart Grids and Cities Congress and Fair (ICSG)*, pages 81–85, 04 2018.

[YLG11] Jie Yan, Chen-Ching Liu, and Manimaran Govindarasu. Cyber intrusion of wind farm scada system and its impact analysis. In *2011 IEEE/PES Power Systems Conference and Exposition*, pages 1–6. IEEE, 2011.

[You05] Hakan L Younes. Verification and planning for stochastic processes with asynchronous events. Technical Report CMU-CS-05-105, Carnegie Mellon University, Pittsburgh, PA, USA, 2005.

[YZZ14] Wei Yuan, Long Zhao, and Bo Zeng. Optimal power grid protection through a defender–attacker–defender model. *Reliability Engineering & System Safety*, 121:83–89, 2014.

[Zet16] Kim Zetter. Everything we know about ukraine's power plant hack. *Wired*, 2016.

[ZJS11] Bonnie Zhu, Anthony Joseph, and Shankar Sastry. A taxonomy of cyber attacks on scada systems. In *2011 International conference on internet*

of things and 4th international conference on cyber, physical and social computing, pages 380–388. IEEE, 2011.

[ZWW+20] Yaofang Zhang, Bailing Wang, Chenrui Wu, Xiaojie Wei, Zibo Wang, and Guohua Yin. Attack graph-based quantitative assessment for industrial control system security. In *2020 Chinese Automation Congress*, pages 1748–1753. IEEE, 2020.

[ZXW+18] Xiaojun Zhou, Zhen Xu, Liming Wang, Kai Chen, Cong Chen, and Wei Zhang. Apt attack analysis in scada systems. In *MATEC Web of Conferences*, volume 173, page 01010. EDP Sciences, 2018.